An Algorithmic Walk from Static to Dynamic Graph Clustering

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

der Fakultät für Informatik
des KIT

genehmigte

Dissertation

von

Robert Görke

aus Ruit, Ostfildern

Tag der mündlichen Prüfung: 11.02.2010

Erster Gutachter: Frau Prof. Dr. Dorothea Wagner

Zweiter Gutachter: Herr Prof. Dr. Michael Kaufmann

Acknowledgments

"This thesis is the fruit of stumbling into, and then enjoying academia." The frightening number of acknowledgments in PhD theses which commence with a statement similar to this makes me feel an urge to not echo it. However, it clearly applies to me as well.

My principal thanks go to my advisor Dorothea Wagner for accepting me into her fabulous team. Shaped by her style of leadership, which is above all friendly, respectful and subtly austere, throughout my years as a member, this team never departed even a bit from being a productive, fun and enjoyable working environment. At this point I must not forget to thank Alexander Wolff, who, by advising my diploma thesis, introduced me to the world of science, became a friend and sponsored me into Dorothea Wagner's team. Looking further back I also thank Paul Bonnington and Cristian Calude of the University of Auckland in New Zealand, who revealed to me that theoretical computer science, discrete mathematics and graph theory in particular are in fact fascinating, something that had largely eluded me prior to my time abroad in Aotearoa. After thanking my friend and former fellow student Daniel Friedrich for coming up with the idea of going to the *land of the long white cloud* in the first place, I guess I should stop this recursion. My sincere thanks go to Michael Kaufmann, my second reviewer, who took on the burden of reading through my work and appreciated it. After years of assembling a thesis and then handing it in for evaluation, a favorable independent opinion on it was a very important acknowledgment for me. I also thank Frank Bellosa, Gregor Snelting, Bernhard Beckert, and Sebastian Abeck for contributing to my successful defense.

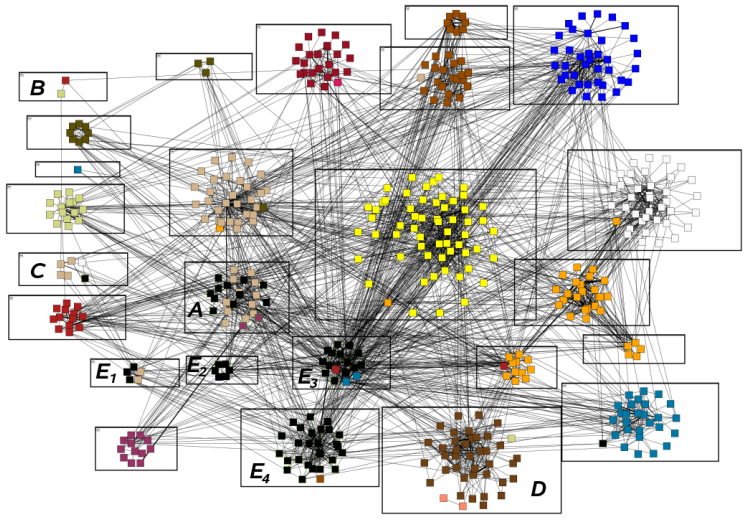
I consider my self lucky to have had the opportunity to share an office with my dear colleague Bastian Katz, whose sense of humor would lighten up even the most desperate and gloomy workday. Moreover, being addicted to solving problems of all kinds, he often was an invaluable second brain thinking about my stuff, the much needed computer science teacher I never had, my Java instructor, and the guy who frequently reminded me that a sip of freshly brewed coffee will help. Thank you. I also thank all the other fabulous colleagues I had, and in particular Thomas Wilhalm, who was the first to welcome me on my first workday, Karin Höthker, my introductory mentor to being a PhD student, Silke Wagner, from whom I took over the position, Étienne Schramm, who was my roommate at my very first conference, Mark Benkert, my exemplar in what to make of conferences abroad by having a bicycle in the luggage, Frank Schulz, who proved that a defense can be done wearing jeans, Thomas Schank, who knew how to layout those changing graphs, Marco Gaertler, who taught me nothing less than how to cluster, José Ignacio Alvarez-Hamelin, who brewed the best *mate* I ever tasted, Michael Baur, thanks for not laughing out too loud when I asked you in my early days what on earth a *heap* is, Martin Holzer, without whom I'd probably have vanished somewhere in Shanghai, Steffen Mecke, my lost PhD-brother-in-arms and my Dungeons & Dragons mate, Martin Nöllenburg, who always was a wise and prudential guide, Daniel Delling, the fast-coding, trash-talking spare-time clusterer, Reinhard Bauer, who was my resource for statistics, Sascha Meinert, my contracted and trusty m&m's caterer, Marcus Krug, my mountain bike soulmate, Ignaz Rutter, who knew the answer to any question about planar graphs, Markus Völker, with whom I could discuss the psyche of Walter Bishop and the agenda of Chloe O'Brian, Tanja Hartmann, the cunning former student of mine whom I am proud to have recruited, Andrea Schumm, in whom I finally found somebody to gossip with about all that clustering business, Thomas Pajor, whose working hours guaranteed I was never alone in the office near midnight, Andreas Gemsa, who keeps Thomas' plush horrors at bay with his giant T4-bacteriophage, and, of course, our angel of tidiness, Lilian Beckert, the best secretary imaginable and Bernd Giesinger, our dependable soft- and hardware wrestler who was always available for a system update or a chat. I also send my thanks to my great students, but to this I dedicated a separate page at the end of this thesis.

I deeply thank my family Carolyn, Siegfried and Martin Görke for their adamant confidence in me and for all the support they provided. Finally, without the staunch moral support and the sometimes much-needed encouragement from Kirsten by my side, my time as a PhD would not have been what it was, a good time.

Deutsche Zusammenfassung (German Summary)

Der Begriff *Netzwerk* begegnet uns im Alltag inzwischen unweigerlich und regelmäßig. Konzepte wie Bekanntschaftsbeziehungen und Straßensysteme geben diesem Begriff seit jeher Relevanz, jedoch ist es der Technisierung einerseits, und der massiven Verfügbarkeit von Daten andererseits zuzuschreiben, dass heute zahllose Sachverhalte mit Hilfe von Netzwerken modelliert werden. Politische Zusammenhänge, wissenschaftliche Kollaborationen in der Forschungsliteratur und bei Patenten, Kommunikationsnetze, Abhängigkeiten in der Ökonomie, Proteininteraktion in Organismen, Räuber-Beute-Beziehungen in Ökosystemen oder Freundschaften bei Facebook, all diese Sachverhalte stellen nur einen bescheidenen Auszug dessen dar, was heute alles als Netzwerk verstanden wird. Doch es kommt nicht von ungefähr, dass ein solches Spektrum mit Hilfe von Netzwerken beschrieben und gehandhabt wird. Netzwerke sind bestens dazu geeignet, komplexe Zusammenhänge verwertbar zu repräsentieren.

Instanzen von Netzwerken wie oben genannt bestehen oft aus Hunderten oder sogar Millionen von Knoten und zumeist noch mehr Relationen zwischen diesen. In zahlreichen Anwendungen ist es von großem Interesse grobe Inhomogenitäten und dicht verbundene Subnetzwerke zu identifizieren, um Zusammenhänge, Interaktionen und Funktionsweisen besser zu verstehen und gezielter Einfluss auf das Netzwerk nehmen zu können. Verfahren die dieses leisten sind Algorithmen zum Clustern von Graphen. Graphen sind dabei die mathematische Formalisierung der Netzwerke.



Der E-Mail Verkehr innerhalb der Fakultät für Informatik an der Universität Karlsruhe (TH) bildet unmittelbar ein sich entwickelndes Netzwerk aus Kollaborationen und sozialen Kontakten. Seit Oktober 2006 arbeiten wir mit der Abteilung Technische Infrastruktur (ATIS) zusammen und sammeln anonymisierte Statistiken über versandte E-Mails innerhalb der Fakultät. Ein großer Vorteil dieses Datensatzes besteht darin, dass die Datenquelle und -verarbeitung sehr verlässlich ist und viel Hintergrundwissen dazu vorhanden ist. Wir betrachten Mitarbeiter als die Knoten eines Netzwerkes und verbinden zwei Knoten mit einer Kante wenn diese im Kontakt via E-Mail stehen, gewichtet mit der Anzahl ausgetauschter Nachrichten in einem festgelegten Zeitraum. Das hier dargestellte Netzwerk berücksichtigt die E-Mails des ersten Quartals 2007. Die Gruppierung (Kästen) teilt die Mitarbeiter der Fakultät in die einzelnen Lehrstühle auf, während die Knotenfarben eine Clusterung repräsentieren, welche ein Algorithmus ohne Hintergrundwissen gefunden hat.

Für die Algorithmik stellt sich die Herausforderung, effiziente und praktikable Algorithmen zur Clusterung von Graphen zur Verfügung zu stellen. Dabei geht es nicht allein darum, gut funktionierende Algorithmen für konkrete Anwendungen oder Datensätze zu entwickeln, sondern um den systematischen Entwurf von Algorithmen für formal sauber gefasste Probleme und deren Analyse und Evaluation unter Betrachtung angemessener Qualitätskriterien. Zentraler gemeinsamer Nenner sind Clusterungen, die auf der Intuition beruhen, dichte Teilgraphen, die untereinander nur lose verbunden sind, zu identifizieren. Ein noch weitgehend unbearbeitetes, wenngleich naheliegendes Feld ist die Übertragung auf dynamische Szenarien. Diese Arbeit behandelt sowohl Themen aus dem statischen als auch aus dem dynamischen Graphenclustern. Dabei spielt die praktische Anwendbarkeit von Maßen und Verfahren eine

ebensogroße Rolle, wie deren theoretische Fundierung. Experimentelle Evaluationen stützen sich sowohl auf Fallbeispiele und Anwendungsdaten als auch auf systematisch generierte Zufallsinstanzen. Im Folgenden wird ein Überblick über die Ergebnisse dieser Arbeit geschaffen.

Statisches Graphenclustern

Verfahren zum Clustern von Graphen basieren zum Teil auf der direkten Identifikation von dichten Teilgraphen oder einem speziellen Schnittprozess. Vorwiegende Praxis bilden jedoch Verfahren, welche die Maximierung eines bestimmten Qualitätsmaßes anstreben, indem Knoten sukzessiv zusammengeballt werden. Ein solches Maß dient dann als Zielfunktion zur Maximierung und als Qualitätskriterium zur Messung der Güte einer Clusterung zugleich. Daher muss es wohlgewählt sein, denn es muss auch ein sinnvolles Verfahren ermöglichen, welches eine effiziente Maximierung dieses Maßes erzielen kann. Das allgemeine Paradigma für Clusterungen, dichte Cluster mit schwacher Verbindung zu finden, lässt jedoch vielerlei Formalisierungen zu, jeweils mit Stärken und Schwächen. Eine wesentliche Rolle in dieser Arbeit spielt ein inzwischen weit verbreitetes Qualitätsmaß für Graphenclusterungen, *Modularity*, dessen Anwendung in verschiedenen Feldern Einzug gehalten hat, bevor jegliche theoretische Analyse davon vorlag. Auf dem Gebiet des statischen Graphenclusterns sind die wesentlichen Ergebnisse, welche in dieser Dissertation erarbeitet werden, die folgenden:

Qualitätskriterium Modularity. In einer theoretischen Untersuchung dieses Maßes wird unter anderem der Beweis geliefert, dass das Entscheidungsproblem, ob eine gegebene Graphenclusterung optimal bezüglich *Modularity* unter allen möglichen Clusterungen ist, NP-vollständig ist. Dies ist eine Bestärkung der gängigen Praxis eine sogenannte gierige Heuristik zur Identifikation von Clusterungen mit hoher *Modularity* zu nutzen. Für diesen Ansatz wird jedoch gezeigt, dass er im Allgemeinen keine relative Approximationsgüte zulässt. Das Konzept auf dem die Definition von *Modularity* beruht ist die Normierung eines einfachen Qualitätsmaßes für Clusterungen mit der erwarteten Qualität bei zufälliger Kantenstruktur. Das Zufallsmodell welches diesen Erwartungswert zulässt wird aufgedeckt und ein Vergleich mit alternativen Umsetzungen vollzogen. In einer systematischen experimentellen Evaluation wird durch einen Vergleich mit anderen etablierten Qualitätsmaßen und Algorithmen zum Clustern von Graphen eine Grundlage für die Nutzung von *Modularity* und dessen gieriger Maximierung geschaffen, welche nicht von einzelnen Fallbeispielen abhängt.

Exaktes und Schnelles Clustern. Es werden zwei spezielle Ausrichtungen des Graphenclusterns behandelt. Zum einen wird ein Rahmenwerk für ganzzahlige lineare Programme dargelegt. Dadurch lassen sich zahlreiche Qualitätsmaße als Zielfunktionen formulieren, und verschiedene Nebenbedingungen wie die maximale Größe von Clustern oder deren Anzahl so formulieren, dass ein optimales Ergebnis berechnet werden kann, wenngleich mit hohem Berechnungsaufwand. Zum anderen wird ein sehr schneller Algorithmus zur Graphenclusterung entwickelt, der – im Gegensatz zu dem vorherrschenden Prinzip – kein einzelnes Qualitätsmaß maximiert, sondern auf lokalen strukturellen Argumenten basiert und trotz dieser Unabhängigkeit von einzelnen Maßen, Clusterungen von hoher messbarer Qualität liefert. Mit diesem Algorithmus lassen sich Graphen clustern, deren Größe gegen eine Milliarde Elemente strebt, eine bislang unerreichte Größenordnung.

Vergleichen von Clusterungen. Lässt ein Sachverhalt mehrere Modellierungen als Graph zu, oder ist durch externe Information eine Einteilung der Knoten bekannt, so besteht oft der Bedarf zu messen, wie ähnlich sich zwei Graphenclusterungen sind. Es wird aufgezeigt welche Nachteile die gängige Praxis hat, rein mengenbasierte Vergleichsmaße zu nutzen. Darüberhinaus wird eine systematische Erweiterung solcher Maße auf graphenbasierte Maße vorgeschlagen, und die Übereinstimmung deren Verhaltens mit intuitiven Forderungen in einer Evaluation bestätigt.

Ein Abstecher in die Netzwerkanalyse

Selbstverständlich ist das Graphenclustern ein wesentlicher Bestandteil der Netzwerkanalyse, doch Synergien mit anderen Teilgebieten sind offensichtlich: Hat man eine gute Clusterung eines Graphen gefunden, so helfen Visualisierungen diese zu begreifen. Zudem hilft die gleichzeitige Betrachtung weiterer Eigenschaften der Knoten über die Clusterzugehörigkeit hinaus, wie zum Beispiel deren Wichtigkeit im Graphen, die Clusterung zu interpretieren. Der Bedarf einer Visualisierungsmethode für große Graphen mit einer Partitionierung der Knotenmenge motiviert diesen Umweg durch die Netzwerkanalyse, der die folgenden Ergebnisse liefert:

Analytische Visualisierungen Partitionierter Graphen. Es wird ein Verfahren zur Zeichnung großer Graphen vorgestellt, welches den Fokus auf einer Gruppierung der Knotenmenge hat. Durch ein kräftebasiertes Verfahren zur Positionierung von Knoten wird einerseits die Struktur innerhalb jeder Gruppe lesbar dargestellt, und andererseits werden auch Tendenzen von Verbundenheit zwischen Gruppen oder Teilen von solchen berücksichtigt. Kombiniert wird dies mit der Art der Darstellung von Elementen, beispielsweise deren Größe und Farbe, welche netzwerkanalytische Eigenschaften wie Zentralität anzeigt. Der Nutzen dieses Verfahrens wird dann in einem Anwendungsbeispiel demonstriert, in dem der Einfluss von Peer-To-Peer Netzwerken im Internet auf die Netzlast untersucht wird. Insbesondere die sogenannte Core-Dekomposition eines Graphen, eine Einteilung der Knoten bezüglich dem Grad ihrer Verbundenheit im Graphen, erweist sich dabei als relevantes Merkmal von Knoten.

Zufallsgraphen mit Festgelegter Core-Dekomposition. Motiviert durch die Beobachtung dass die Core-Dekomposition eine große Bedeutung für die Funktionsweise von Netzwerken hat, wird ein Algorithmus vorgestellt und evaluiert, der Zufallsgraphen mit festgelegter Core-Dekomposition erzeugt und zudem die typische Gradverteilung von echten, unüberwacht wachsenden Netzwerken aufzeigt.

Clustern Zeitlich Veränderlicher Graphen

Reale Netzwerke sind in vielen Fällen von veränderlicher Natur, so dass sowohl Kanten als auch Knoten in einem zeitlichen Verlauf aus dem Netzwerk gelöscht, oder in das Netzwerk eingefügt werden. Ein solches Szenario wirft zum einen kanonische Forderungen auf, wie beispielsweise ein ressourcenschonendes Update von Clusterungen aber auch neue Fragestellungen: Kann man eine gute Clusterung eines großen Netzwerks so pflegen, dass eine Veränderung des zugrundeliegenden Graphen schnell in eine sinnvolle Änderung der Clusterung umgesetzt wird? Kann man dabei garantieren, dass die gepflegte Clusterung stets eine gewisse Qualität hat? Kann man aus der zeitlichen Entwicklung einer Clusterung schließen, wie sich Trends in dem Netzwerk in Zukunft verhalten werden? Beweisbare theoretische Resultate spielen hierbei eine ebensogroße Rolle wie die experimentelle Evaluation neuer Konzepte. Die wesentlichen Ergebnisse zum dynamischen Clustern sind:

Ein Zufallsgenerator für Dynamische Graphen. Um experimentelle Evaluationen fundiert durchführen zu können werden auch in dynamischen Szenarien systematisch erzeugte Zufallsinstanzen benötigt. Es wird ein Generator für dynamische Netzwerke mit Clusterstruktur entwickelt, dessen Dynamik auf einem konsistenten Wahrscheinlichkeitsmodell bezüglich einer veränderlichen Basis-Clusterung beruht.

Online-Dynamische Clusterverfahren. Es werden zwei Ansätze verfolgt, mit denen eine Clusterung eines Graphen nach einer Veränderung des Graphen gepflegt werden kann. Zum einen wird eines der wenigen statischen Clusterverfahren, welche eine gewisse Qualitätsgarantie der Clusterung erfüllen, voll dynamisiert. Dabei ergeben sich interessante Einsichten in die Dynamisierung von minimalen Schnittbäumen, auf denen das Verfahren basiert. Zum

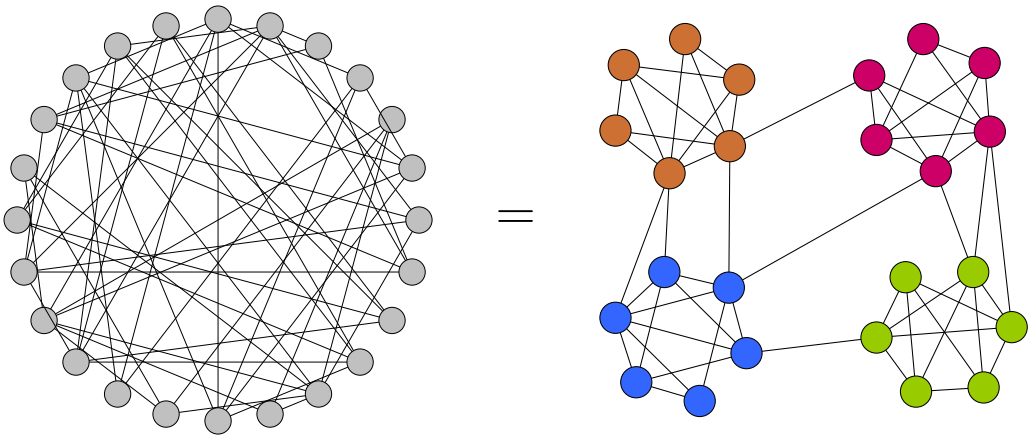
anderen werden sowohl der gängigste als auch der derzeit schnellste bekannte Clusteralgorithmus zur statischen Maximierung des Qualitätsmaßes *Modularity* dynamisiert. Dabei wird gezeigt, dass die vorgeschlagenen Algorithmen in der Praxis drei wesentliche Kriterien erfüllen: Im Vergleich zu Clusterungen welche mit Hilfe der statischen Algorithmen gefunden werden, haben die Clusterungen der dynamischen Algorithmen eine höhere Qualität, sie werden schneller gefunden, und die Ähnlichkeit aufeinanderfolgender Clusterungen ist größer. Für eine Anwendung bedeutet dies, dass zwischen zwei aufeinanderfolgenden Zeitschritten keine großen Veränderungen der Clusterung erwartet und verarbeitet werden müssen.

Offline-Dynamische Clusterverfahren. Im Gegensatz zum vorigen Fall steht in einem Offline-Szenario eine vergangene zeitliche Sequenz von Graphen zur Verfügung. Diese kann beispielsweise eine Sammlung monatlicher Zusammenfassungen eines dynamischen Netzwerkes darstellen. Es steht also beim Clustern eines Zeitschrittes mehr Information zur Verfügung als bei einem Online-Szenario, denn alle Zeitschritte können nun zugleich bearbeitet werden. Hier stellt sich die Frage nach einer dynamischen Clusterung dieser Sequenz, für die einerseits die Kriterien aus dem Online-Szenario gelten, also die Güte der einzelnen Clusterungen pro Graph aus der Sequenz und die Ähnlichkeit aufeinanderfolgender Clusterungen – welche im Offline-Szenario noch besser realisiert werden kann. Andererseits soll nun zusätzlich ein Verfolgen einzelner Cluster über die Zeit hinweg ermöglicht werden. Damit wären Trends in der Entwicklung von Clustern beobachtbar. Es wird ein Rahmenwerk zum optimalen Lösen vielerlei Formalisierungen dieser Offline-Problemstellungen dargelegt, welches auf ganzzahliger linearer Programmierung beruht. Desweiteren wird ein schnelles und praktikables Verfahren vorgestellt, welches diese Aufgabenstellung löst. An einem Fallbeispiel wird der Nutzen dieses Verfahrens demonstriert.

Contents

Chapter 1

Introduction



Graphs excel at hiding their structure. Graph clustering aims at revealing their structure. A decent algorithm for graph clustering can find the four densely knit groups in this graph.

Contents

Section 1.1

Preface

Fly, you fools!

(Gandalf the Grey,
at the Bridge of Khazad-Dûm,
The Lord of the Rings, J.R.R. Tolkien, 1954)

What is the structure of a network? There are many ways to answer this question, depending on the point of view. And considering the myriad issues modeled by networks, there are many points of view. An integral trait of most networks are groupings within. In this thesis we shall take on a view which focuses on such groupings and considers them a defining property of the structure of a network.

We inevitably encounter the term *network* on a regular basis. Roads and social relations always gave relevance to this concept, however, it is due to mechanization and today's massive availability of data that nowadays countless circumstances are modeled by networks. Scientific collaboration, communication grids, economic or political dependencies, protein interaction and friends at facebook are but examples of what we understand as networks. Yet it is no coincidence that networks serve to describe such a variety of issues, this concept is very suitable for representing complex interrelations.

Clusters in networks are areas where elements are rather densely interconnected. This intuition leads to the general paradigm describing *graph clusterings*, which is *intra-cluster density and inter-cluster sparsity*. Depending on the field of application, the literature on networks knows many names for clusters and clusterings[1], such as natural groups, modules, community structure or large scale inhomogeneities. Also depending on the application is the meaning of a clustering. Consider a sports club and the network of friendship among its members. Suppose the club is split due to a dispute between the manager and the trainer, then it is quite likely that a cluster of members, i.e., a group of close friends, decides en bloc to whom to affiliate [230]. In the network of facebook contacts, the most recent headline will spread quickly inside a cluster, before propagating further. By the same principle, a computer virus will quickly spread inside a dense infrastructure. In turn, however, if only few connections to other clusters exist, it might be feasible to contain the virus by guarding or cutting off those few links. A fascinating variant of this idea is being tackled in the field of biochemistry. Proteins are part of any living organism, they consist of chains of amino acids and their blueprints are encoded in the genes of an organism. Proteins serve as the main protagonists within and between cells and one of their major tasks is signal transduction. Networks based on this function are called *protein-protein interaction* networks [22]. Recent advances in the detection and the measurement of these interactions lead to the hope that inhibiting certain such interactions can stop diseases from spreading through an organism. Clustering methods in protein-protein interaction networks [155] help to identify those interactions that are critical to spreading.

[1] In order to avoid confusion, note that the term *clustering* refers to both the set of clusters and to the activity of finding such a set, as a gerund of the verb *to cluster*.

As another example from biology, consider an ecosystem consisting of many species. If we model this as a network such that connections between species represent predator-prey relationships and other dependencies in terms of sustenance, then the extinction of one species will immediately and most heavily affect its cluster. A final example are logical expressions. A SAT-instance is a formula consisting of a number of true-false variables, connected by ands, ors and negations. To see whether there is a configuration of these variables such that the overall instance yields a logical true is a very fundamental and hard problem. If we find a cluster in the network of variables, where connections represent dependencies of variables, we can attempt to solve this cluster, which represents a smaller and easier problem, separately. Figure 1.1.1 shows a network derived from a SAT-instance, colors represent clusters of variables.

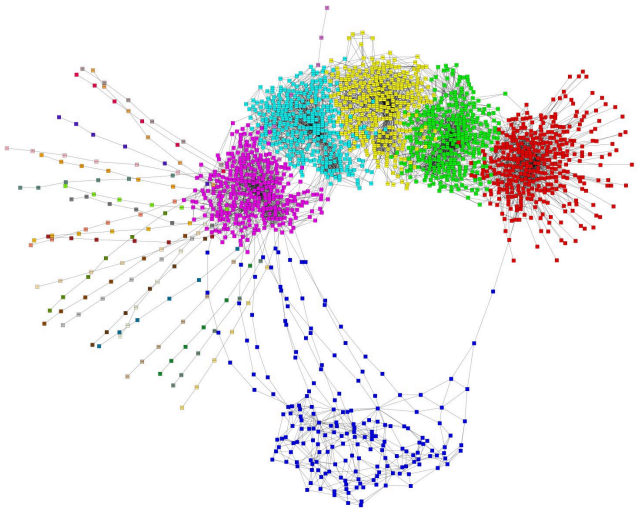


Figure 1.1.1. Clustered network model for a SAT-instance of the hardware requirements of a multiplication unit

Graph clustering has become a central tool for the analysis and the exploration of networks in general, with applications ranging from the field of social sciences to biology and to the growing field of complex systems. The principal requirement for graph clustering is an algorithm which identifies the clustering. Challenges abound for such an algorithm, with the most obvious being a sound formalization of what a good clustering actually is. The paradigm of *intra-cluster density and inter-cluster sparsity*[2] leaves much room for possible mathematical formalizations, and as a result of that, many have been proposed. Frankly speaking, there is neither a universal formula that works best for all networks, nor an algorithm which is suitable for all instances. The composition of networks can be so diverse that no single concept has proven itself superior in general. For any algorithm one can construct a network which it fails to cluster in accordance with human intuition.

Nevertheless, clustering methods are needed and several approaches have been proposed which behave very well on a large number of instances. One particularly widespread algorithm tries to find a clustering which yields a high value of *modularity*. *Modularity* quantifies the abovementioned paradigm as a *quality measure* for clusterings, it has recently been coined by Michelle Girvan and Mark Newman [178]. This measure and a simple algorithm based on it have quickly spread into diverse fields, despite a lack of sound arguments of their appropriateness. However, in many areas of science, the mere availability of an algorithm which is easy to use and appears to work reasonably well silences many legitimate doubts. A major part of this thesis is dedicated to scrutinizing the theoretical properties of this measure and its behavior in practice. Two obvious points motivate looking into different approaches for clustering: On the one hand, specific properties of *modularity* introduce a subtle but undeniable bias into clusterings identified by the above algorithm. On the other hand, as soon as the size of a network approaches a million elements, the algorithm becomes infeasible. Such huge instances can be road networks, parts of the world wide web or huge social networks. These two points motivate the work on Orca, a measure-independent clustering algorithm for huge networks and one of the first attempts to cluster networks which approach billions of elements. Given a good graph clustering algorithm, a visualization such as the one shown in Figure 1.1.1 is a first means to get an impression of the result of the algorithm. In an interlude between the clustering of static and of dynamic networks, a visualization technique for clustered or otherwise partitioned networks is proposed and exemplified, and a foray into other areas of network analysis which are related to graph clustering is conducted. Most

[2] This paradigm is often coined *intra-cluster density* versus *inter-cluster sparsity*, as the two postulations generally compete.

networks listed in the examples above are no fixed instances. They are subject to changes as elements leave or join the network and ties are established or broken. A quick update of an identified clustering, after something changes in the network, is just as important as a long-term analysis of trends present in the clustering of a network. The third large part of this thesis is concerned with clusterings of changing networks. The following section gives an overview of this thesis.

Thesis Outline and Contribution

Three general topics are addressed in this dissertation, these are static graph clustering, related topics in network analysis and dynamic graph clustering. We provide fundamental work for prevalent practice in static graph clustering and establish new tools for this field. The design of a technique for analytic visualizations of graph clusterings then briefly leads to other topics of network analysis and applications thereof. We then return to graph clustering, however, considering dynamic instances and presenting advances for both theoretical and practical *online* problems and propose methods for *offline* dynamic graph clustering. Each topic constitutes a separate chapter and is subdivided into sections, which make up fairly self-contained units. Parts of this thesis have previously been published in [42, 43, 45, 44, 101, 68, 69, 70, 72, 19, 102, 116, 11, 12, 13, 14, 33, 34, 103, 120, 117, 118, 115]. We will point out the respective publications and coauthors in the corresponding chapters and sections. The following is a brief summary of the results obtained in the individual chapters and sections.

Chapter 2 – Static Graph Clustering

This chapter is concerned with algorithms for the identification of graph clusterings. In contrast to later parts, we here deal with static, unchanging graphs. In the preface we introduce the topic and discuss existing approaches.

Section 2.2 – On Modularity Clustering. We investigate the complexity of a particularly widespread approach for graph clustering, *modularity*-maximization. After we establish the NP-completeness of the corresponding decision problem, we investigate the behavior of the commonly applied greedy agglomerative heuristic for this task. We devise an integer linear programming formulation and review a few benchmark networks and previously found clusterings in the light of optimality.

Section 2.3 – Lucidity-Driven Graph Clustering. The probabilistic setup *modularity* is build upon is scrutinized, and a discrete probability space defined, which supports the definition of this measure. At the same time, this result points out that the original assumptions for *modularity* require both loops and parallel edges to be allowed. We then set up *modularity*'s underlying paradigm of "quality compared to expected quality" (*lucidity*) and derive three other implementations of it. In a systematic experimental evaluation we finally investigate the behavior of *modularity* and these variants on a ground-truth random generator and on established clustering algorithms, and then evaluate *lucidity*-driven clustering algorithms that operate as greedy agglomerative heuristics.

Section 2.4 – ILPs for Graph Clustering. In this section we assemble an overview of integer linear programming (ILP) formulations for graph clustering, using three different quality measures as objective functions. Our formulations allow different additional constraints such as bounds for cluster sizes. *Modularity*-optimal clusterings for numerous well-known example networks are computed and basic tools for engineering the ILPs are evaluated.

Section 2.5 – Orca. The focus is now turned towards huge graphs, i.e., instances comprising up to several million elements which cannot be clustered with most established algorithms.

We describe ORCA, a fast clustering algorithm based on contraction and reduction operations that are not motivated by a single quality measure for clustering, but instead rely on local density. Although we intentionally avoid the bias towards a particular quality measure, such as *modularity*, ORCA competes well with established *modularity*-based algorithms in terms of this measure. In an experimental evaluation on huge networks, we compare ORCA to other algorithms for large instances.

Section 2.6 – Comparing Clusterings. As the final part of the chapter on static graph clustering, this section veers towards changing graphs and changing clusterings; more precisely, we address comparison measures for clusterings. Examining established measures for comparing sets, we show why these measures are inadequate for the task of comparing graph clusterings. The crucial point is that the set of edges must not simply be ignored. We then design a new measure and systematically transfer whole families of set-based measures to the context of graph clusterings. Finally, we show which measures comply with basic postulations for the comparison of clusterings.

Chapter 3 – A Foray into Network Analysis

In this chapter we take a somewhat broader look into network analysis techniques other than graph clustering. The general motivation is that such techniques can help to understand the results of a clustering algorithm in the context of other structural properties of the network. In fact the tool we propose is then employed in a non-clustering application which focuses on the *core decomposition* of a network, a concept we ultimately take a proper detour to.

Section 3.2 – LunarVis—Analytic Visualizations of Large Graphs. Arguably the single most important addition to a clustering algorithm is a means to present the result in an informative way. For an exploratory setting, we propose *LunarVis*, a new layout paradigm for drawing large networks, with a focus on decompositional properties. This visualization technique employs a combination of group-internal and -external force-directed procedures to reveal the structure of connectivity in a segmented graph. In addition to this, the layouts *LunarVis* produces easily accommodate results of other measurements performed on the network as visual properties, such that a comprehensive impression of the structure of a network is allowed for.

Section 3.3 – Overlay-Underlay Exploration Driven by Analytic Visualizations. In a case study on the traffic caused by peer-to-peer networks such as *Gnutella*, we put the results of the preceeding section to good use. We employ *LunarVis* in an analysis of the overlay- (in the *Gnutella* network) and underlay-traffic (in the physical Internet) caused by *Gnutella*, thereby revealing both correlations between the two and specific discrepancies to a simulated peer-to-peer network, where peering is based on random choices. Visual analytics then guides a focused investigation.

Section 3.4 – k-Core-Driven Random Graphs using Preferential Attachment. Since the *core decomposition* of a network proved to be a crucial property in the above case study, we set our focus on it, in this section. We establish tight bounds on a number of properties of the *core decomposition* and prove how it can be preserved when altering a network. This leads us to a simple algorithm for generating random networks that precisely incorporate any predefined *core decomposition*, and can even accommodate the mechanism of *preferential attachment*. In an experimental case study on the task of simulating the Autonomous Systems graph of the Internet, we then compare our generator with others.

Chapter 4 – Clustering a Dynamic Graph

In this chapter we finally dare to tackle dynamic graph clustering. In the preface of this chapter we introduce possible problem statements, discuss related literature, and establish the necessary notational conventions.

Section 4.2 – A Generator for Dynamic Clustered Random Graphs. An integral part of most experimental evaluations are random instances. As an aide for our later experiments we here describe a ready-to-use generator for dynamic random graphs with an implanted clustering structure. The generator is driven by a gradually changing ground-truth clustering, which motivates changes in the graph, according to a sound probabilistic setup.

Section 4.3 – Modularity-Driven Clustering of Dynamic Graphs. In the tradition of Chapter 2, we now return our focus to *modularity*-driven clustering. Based on the currently fastest and the most widespread heuristics for *modularity*-maximization, we develop algorithms for dynamically updating clusterings after a graph changes. A particular focus is set on the search space of these update algorithms. In an experimental evaluation we compare dynamic and static algorithms in several setups. The results reveal that our dynamic algorithms (i) save runtime, (ii) yield higher *modularity* and (iii) much smoother clustering dynamics than their static versions, with small search spaces working best.

Section 4.4 – Dynamic Min-Cut Tree Clustering. Our second approach for dynamic graph clustering focuses on provable quality. Building upon an algorithm for static graph clustering which provides such guarantees, we develop an algorithm that efficiently maintains these guarantees for a changing graph, yielding many insights into the dynamics of *minimum s-t-cuts*. For almost all combinatorial cases, an asymptotic speed-up can be ascertained, which is confirmed by first experiments. We project how our results can be generalized to *minimum-cut trees*.

Section 4.5 – Time-Dependent Graph Clustering. Finally we turn to *offline* settings of dynamic graph clustering. We investigate feasible formulations which generally aim at a balance between the quality of individual static clusterings and a *smooth* transition between them. After we describe an ILP formulation which theoretically accommodates most such formulations, we propose *time expanded* graph clustering as a practical approach. In a case study we describe good parametric choices and show the potential of this method.

Chapter 5 – Epilogue

This concluding chapter accommodates all the remaining parts that do not fit anywhere else.

Section 5.1 – Data Sets and Applications. The many data sets that were used in this thesis, and in applications of graph clusterings that have not yet been mentioned, are briefly and informally described in this section, alongside a quick note on how they have been used.

Section 5.2 – Side Notes. On the one hand, this informal section is dedicated to the many excellent students I worked with as an advisor, on the other hand I seize the opportunity to say a few words about other activities that successfully distracted me during my time as a PhD student.

Section 5.3 – Conclusion. This final section concludes my thesis.

My Two Pennies' Worth

Boon and bane closely accompanied graph clustering as I pursued it, halfway between theory and applicability. The guiding idea to both focus on the practical performance of concepts for clustering and also investigate their theoretical properties and foundations leads to a fine line between satisfying both directions and acting as an easy target for criticism from either or even both directions. Frankly speaking, it is easier to concentrate on one direction and jettison the other. In particular, you will have an easier time increasing your cursed, all-important count of publications. That said, I do not regret having chosen this path, since one without the other is simply less than half the fun.

Writing Style. I generally *emphasize* a word when it is a term that is now being defined properly, or whenever a word is somehow crucial. I use *slanted font* for terms which have previously been defined properly, and might differ from a naïve intuition of the term in some subtle way. There is but one conclusion in this thesis, at the very end. Individual sections are introduced and summarized at their beginning, with my personal and informal two pennies' worth at the end of each such introductory part, followed by a convenient listing of the main results and an outlook. The prefaces to each of the three chapters differ from this, they introduce the respective area and then state the driving questions for that chapter and the answers found, followed by a summarizing outlook. I mostly avoid abbreviations, with a few exceptions which avoid lengthy insertions. I object to the common practice of citing, which usually consists of shaping sentences around lengthy and syntactically relevant substitutes for the cited work and adding an "invisible" key as a reference to it. For brevity, I solely use abbreviated keys, such as "[5]", for citations, and either have them serve as a syntactical element or act as a phantom. I dislike \eqref.

Section 1.2

Preliminaries, Graphs, Clusterings

I believe that the volcanic rock of Skye contributes to the pungent aroma...

(Michael Jackson, whisky writer and expert, on Talisker, 10 year old)

ANYBODY CAN IMAGINE A NETWORK. While this might sound naïve, it certainly has its part in the fact that graph theory is among the rather quickly accessible fields of mathematics. The basic tools and observations can be understood without having studied countless lemmata that deal with nigh incomprehensibly abstract matter. Quite obviously, it is a fallacy to conclude that this renders the problems one asks in graph theory less challenging. However, it does make assembling preliminaries and notation a lot easier. We will summarize general preliminary information such as notation and definitions used throughout this work in the following, but will introduce more exotic concepts in the chapters and sections where they are actually used; we even dare to repeat a few specific things here and there, if they appear indispensable to comprehension. Frankly speaking, anybody who is familiar with graph theory can skip the following subsection without any worries, as we universally stick to common notational conventions. The notation used for quantifying properties of *clusterings* in Section 1.2.2 might be worth looking at, briefly. As a final note, in these short preliminaries we try to cover the basic terms and definitions required in this thesis; however, many concepts reach far beyond what can be introduced here. In particular, we refer the reader to established literature [106, 58] and references therein for quite a few topics, such as complexity of problems, asymptotic running times of algorithms, linear programming and integer linear programming.

graph vs. network

Before we start, a word concerning the terms "network" and "graph" should be said. Generally speaking a *graph* is the mathematical formalization of a *network*. Thus, the term network usually denotes a real-world instance which stems from some application like route planning in road networks or network design for electric circuits. Since networks are nowadays "observed" in myriad contexts that are not only very diverse but sometimes also seem slightly absurd, it comes as no surprise that many people have in mind different notions when speaking of networks. The graph that corresponds to a network dispels all vagueness and ambiguity about the contained entities and yields a formal description one can work with, mathematically. In this work, however, we shall use these two terms almost interchangeably and resort to whichever is more commonly used in the context. That said, we will never handle a network without actually having a mathematical formalization of it in mind. The crucial point is that sometimes finding the correct or the most appropriate graph model for a network is more difficult and decisive than a later analysis.

1.2.1 Graphs

Good books on graphs and graph theory include [143] and [78]. However, this work largely belongs to the field of *network analysis*, the science that leaves purely theoretical issues to graph theory and concerns itself with methods and tools that help to answer questions about real networks. Such techniques thus apply graph theory. In this work we will largely adopt the notational conventions of a good book on network analysis [46].

Basic Definitions and Properties. A *graph* $G = (V, E)$ is a tuple consisting of the set V of *nodes* and the set E of *edges*. An edge e is a unordered pair $e = \{u, v\}$ of two nodes; we say that e *connects* or *links* u and v, such that u and v are the *endnodes* of e. We denote the cardinality $|V|$ of the set V of nodes as $n := |V|$, and the cardinality $|E|$ of the set E of edges as $m := |E|$. Except for some specifically mentioned cases where we make a distinction, the terms node and *vertex* are equivalent. Given an edge $e = \{u, v\}$ we say that e is *incident* to u (and to v), and that—by virtue of edge e—nodes u and v are *adjacent*. As a shorthand we often abbreviate adjacency and incidence by $e \sim v$ or $u \sim v$, respectively. Arriving at the first term which needs disambiguation as the literature on graphs does not agree about its meaning, we deal with *simple* graphs in most of this work. In this work, a simple graph neither contains edges that constitute *loops*, i.e., edges that are incident twice to the same vertex, nor *parallel* edges, i.e., E cannot contain the same edges multiple times. Needless to say, the notion of an unordered pair and a set do not allow such anyway, strictly speaking. However in some cases we will consider *non-simple* graphs and thus allow both loops and parallel edges. In that case the set E of edges will become a multiset of edges, i.e., allowing multiplicities of elements which lead to parallel edges, and an edge will also become a multiset, such that a loop $\{v, v\}$ is possible.[3] The number of edge incidences a node has is called its *degree*. While for simple graphs this is exactly the number of edges it is incident to, for non-simple graphs we have to doubly count loops.[4] The maximum degree in a graph is often denoted by Δ, however, due to conventions in the literature this variable will sometimes have a different meaning. Unless otherwise noted we will use simple graphs.

graph, node, edge
n, m
vertex
adjacent, incident
e ∼ v, u ∼ v
simple
loop, parallel edges
non-simple
degree, Δ

Connectivity and Paths in Graphs. For two nodes v_0 and v_k in a graph G, a *walk* W between u and v is a sequence of nodes and edges of G such that $W = v_0, e_1, v_1, e_2, \ldots, e_k, v_k$ with $e_i = \{v_{i-1}, v_i\}$ for all $1 \leq i \leq k$. A *path* is a walk that does not contain any edge more than once, and a path that does not even contain any node more than once is a *simple path*. A path which ends where it starts, i.e., $v_0 = v_k$, is called a *cycle*, or a *simple cycle* if no node except $v_0 = v_k$ is contained more than once. The number of used edges in a walk or a path is its *length*. The length of the shortest path between two nodes u, v of a graph G is their *distance*, it is usually defined to be infinity if there is no such path. The *diameter* of a graph is the maximum distance between nodes present in a graph. A graph that contains no cycles is a *tree*.

We call $H = (V', E')$ a *subgraph* of a graph $G = (V, E)$ if H constitutes a graph, and $V' \subseteq V$ and $E' \subseteq E$. We often use the notion of an *induced subgraph*. Such a subgraph $H = (V', E')$ is either specified by its set V' of nodes, in which case E' contains exactly those edges of E that in G are solely incident to nodes in V', or by its set E', in which case V' contains exactly those nodes of V that in G are incident to at least one edge in E'; formally speaking this translates to $E' := \{\{u, v\} \mid u, v \in V', \{u, v\} \in E\}$ and $V' := \{v \in V \mid \exists \{u, v\} \in E'\}$. We write $H = G[V']$ and $H = G[E']$ in the former and the latter case, respectively. A graph G is *connected* if there is a path between any pair of two nodes in G, i.e., any node can be reached by starting from any other node and traversing a subset of edges, otherwise G is *disconnected*. The *connected components* of a graph G are all maximal[5] subgraphs H_i

walk
(simple) path
(simple) cycle
length
distance, diameter
tree
(induced) subgraph
(dis-) connected
connected components

[3] In the literature, graphs that contain parallel edges are often called *multigraphs*, and graphs that do not contain loops are often called *loop-free*.

[4] Sadly, we have already reached a point which is beyond what some literature I dealt with cares about.

[5] The term *maximal* is used as usual, i.e., no element can be added without violating some property.

of G which are connected. A subgraph T of a graph G which is a maximal tree is a *spanning tree*. If a connected component containing edge e becomes disconnected by removing e, we call e an *isthmus* or a *bridge*. Figure 1.2.1 depicts a network, which is already cast into a graph: the nodes V represent persons and the relations between them—friendship in this example—are modeled by the set E of edges. Alice has degree 3 and the edge $\{\text{Alice}, \text{Bob}\}$ is an isthmus. The subgraph induced by Bob, Carol and Eve, that triangle, is called a *clique*, as it is edge-maximal; a *clique* on i nodes is denoted K_i.

spanning tree
isthmus, bridge
clique

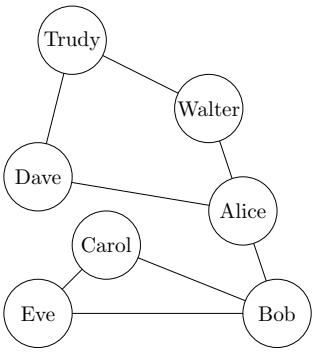


Figure 1.2.1. A social network of seven persons, an edge between two persons represents friendship.

Direction and Weight. In some rare cases we will use—and explicitly say so—the notion of a *directed graph*. An edge e in a directed graph is an *ordered pair* $e = (u, v)$ of nodes, i.e., the edge is not a mutual relationship but has a *source* and a *target*.[6] A notion we shall more often use is that of a *weighted graph*. A weighted graph G is a triple $G = (V, E, \omega)$ with V and E defined as above, and with an *edge weight* function $\omega : E \to \mathcal{D}$, with some domain $\mathcal{D}$. Modeling anything related to, e.g., the strength of a relation or the distance of two nodes, ω maps each edge e to an edge weight $\omega(e)$. We define $\omega(\{u, v\}) = 0$ if $\{u, v\} \notin E$. Instead of $\omega(\{u, v\})$ we will usually denote the weight of an edge $\{u, v\}$ by $\omega(u, v)$. The domain of ω usually is $[0, 1]$ but sometimes $\mathbb{R}_0^+$ or even $\mathbb{R}$, we shall announce any deviation from $[0, 1]$. In much of this work we will generalize assertions from unweighted to weighted graphs, in some cases simply claiming a proof to be easy to see. In most such cases it is helpful to keep in mind that a generalization must yield the result of the unweighted case if we resort to default weights such as 0 and 1. For a node v, we define the weight $\omega(v)$ of a node as the sum of the weights of incident edges, doubly counting loops. In case we allow parallel edges, $\omega(e)$ is the weight of the single edge e, whereas $\omega(\{u, v\})$ is the sum of weights of edges between nodes u and v. The analogon for unweighted graphs is $A(u, v)$ which counts the number of (parallel) edges between u and v. The sum of the weights of all edges in a graph is denoted by W. Whenever a weight immediately corresponds to a *cost* we adhere to conventions and substitute $\omega(e)$ by $c(e)$. The weight of a path in a weighted graph is the sum of the weights of the edges used. Thus, the distance between two nodes is the weight of the lightest path between them.

directed graph
source, target
weighted graph
edge weight
weight of a node, $\omega(v)$
$A(u, v)$
W
$c(e)$
weighted distance

It is important to see the difference between a weight expressing the *strength* of a tie (high values indicate an important, strong edge) and a weight quantifying the *distance* between nodes (low values indicate important edges). For the notion of path lengths, distances are required, while more often the other case is used: High edge weights correspond to a high similarity or togetherness of the incident nodes.

distance vs. similarity

Sets, Quantors and Enumerators. We often enumerate over sets of tuples of nodes. It is worth announcing three particularly notorious variants. For simplicity in enumerations we assume that V is ordered, such that for any two distinct nodes u, v either $u > v$ or $u < v$ holds. We denote the set of multisets of two nodes that can be connected by an edge in a non-simple graph as $V^\times = \{\{u, v\} \mid u \geq v, u \in V, v \in V\}$, with $\tilde{m} := |V^\times| = \binom{n}{2} + n$. The set of all 2-tuples, i.e., ordered pairs from V is $V^2 = \{(u, v) \mid u \in V, v \in V\}$. Finally, the set of all unordered pairs of nodes is denoted by $\binom{V}{2}$, and thus $E \subseteq \binom{V}{2}$ in simple graphs. We shall sometimes use one notion in an enumerator and then a different notation in the enumerand, e.g., "$\sum_{(u,v)\in V^2} \omega(\{u, v\}) < 2W \Rightarrow$ the graph is not simple", it might help to put straight that this will be done on purpose. For convenience we abbreviate extending and reducing sets: Given a set A and elements e and e', we write $A + e := A \cup \{e\}$ and $A - e := A \setminus \{e\}$.

$V^\times, \tilde{m}$
V^2
$\binom{V}{2}$
$A + e$, $A - e$

[6] Directed edges are often called *arcs* in the literature.

Simple Operations and Constructs. A node v's (standard) *neighborhood* is $N(v) := \{w \in V \mid \{v,w\} \in E\}$, and the set of vertices within distance d of v is denoted as the *d-neighborhood* $N_d(v) = \{w \in V \mid w \neq v, \text{distance}(v,w) \leq d\}$. A *contraction* of G by $N \subseteq V$ means replacing set N by a single super-node η, and leaving η adjacent to all former adjacencies u of vertices of N, with edge weight equal to the sum of all former edges between N and u. Analogously we can *contract* by a set $E' \subseteq E$.[7] A *cut* of a graph is a partition of V into two sets, it is sometimes also identified by the subset of edges which connect between those sets. A cut can also "cut" only a subset $U \subseteq V$. The *weight of a cut* is always the sum of the weights of the edges between the two sets. We often call a cut (U_1, U_2) of $U \subseteq V$ a 2-*partition* of U in order to stress that $U_1 \cap U_2 = \emptyset$ and that $U_1 \cup U_2 = U$. The *adjacency matrix* $\text{Adj}(G)$ of graph G is an $n \times n$ matrix with entries $a_{ij} = \omega(v_i, v_j)$. The *normalized adjacency matrix* $\text{AdjN}(G)$ of graph G is $D^{-1}\text{Adj}(G)$ where D is the $n \times n$ diagonal matrix with entries $d_{ii} = \omega(v_i)$. The (unnormalized) *Laplacian* L of a graph is $D - \text{Adj}(G)$.

neighborhood $N_d(v)$
contraction of G
cut
weight of a cut
(normalized) adjacency matrix $\text{Adj}(G)$, $\text{AdjN}(G)$
Laplacian L

1.2.2 Clusterings

Given an unweighted graph $G = (V, E)$. Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be a partition of V, with each C_i being non-empty. We call $\mathcal{C}$ a *clustering* of G and the elements C_i *clusters*. The cluster which contains node v is denoted by $\mathcal{C}(v)$. We identify a cluster C_i with the node-induced subgraph of G, i.e., the graph $G[C_i] := (C_i, E(C_i), \omega_{|E(C_i)})$, where $E(C_i) := \{\{v,w\} \in E : v, w \in C_i\}$. Then $E(\mathcal{C}) := \bigcup_{i=1}^{k} E(C_i)$ is the set of *intra-cluster* edges and $E \setminus E(\mathcal{C})$ the set of *inter-cluster* edges, with $|E(\mathcal{C})| =: m(\mathcal{C})$ and $|E \setminus E(\mathcal{C})| =: \overline{m}(\mathcal{C})$. The set $E(C_i, C_j)$ denotes the set of edges connecting nodes in C_i to nodes in C_j. We denote the number of non-adjacent intra-cluster pairs of nodes as $m(\mathcal{C})^c$, and the number of non-adjacent inter-cluster pairs as $\overline{m}(\mathcal{C})^c$. Further, we generalize degree $\deg(v)$ to clusters as $\deg(C) := \sum_{v \in C} \deg(v)$.

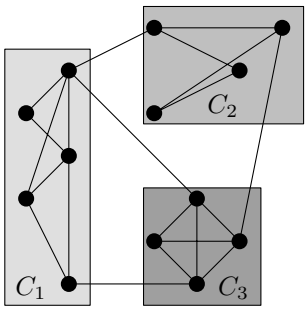


Figure 1.2.2. An example clustered graph with $k = 3$ and $m(\mathcal{C}) = 17$ and $\overline{m}(\mathcal{C})^c = 52$

clustering, cluster
intra-, inter-cluster
$m = E(\mathcal{C})$, $\overline{m}(\mathcal{C})$
$E(C_i, C_j)$
$m(\mathcal{C})^c$, $\overline{m}(\mathcal{C})^c$
$\deg(C)$

A clustering is *trivial* if either $k = 1$ ($\mathcal{C}^{\mathbf{1}}$), or all clusters contain only one element, i.e., are *singletons* ($\mathcal{C}^V$). We denote the set of all possible clusterings of a graph G with $\Psi(G)$. We call a graph of which all connected components are cliques a *clustergraph* and $F_{\mathcal{C}}$, the set of edges to be added or deleted in order to transform a given graph and clustering $\mathcal{C}$ into an according clustergraph, the *cluster editing set* of $\mathcal{C}$.

clustergraph
cluster editing set $F_{\mathcal{C}}$

Weighted Graphs. When using edge weights, all the above definitions generalize naturally by using $\omega(e)$ instead of 1 when counting edge e. For the purpose of clustering, weights are considered to represent *similarities* unless otherwise noted. Consider a weighted graph $G = (V, E, \omega)$, then $\omega(\mathcal{C})$ ($\overline{\omega}(\mathcal{C})$) denotes the sum of the weights of all intra-cluster (inter-cluster) edges, W denotes the sum of all edge weights. To further simplify notation we use $\omega(E') = \sum_{e \in E'} \omega(e)$. The maximum edge weight in a graph is called $\omega_{\max}$. Not quite so obvious are the definitions of $\omega(\mathcal{C})^c$ and $\overline{\omega}(\mathcal{C})^c$, as here a weight needs to be assigned to an absent edge, or rather, we need to quantify how much edge mass is missing, compared to "full connectivity". While this is trivial for unweighted graphs, in the weighted case we follow [46] and postulate a *reasonable* maximum weight M to compare to. We shall discuss this issue in more detail in Section 2.3, but until then we assume that $M = \max(\text{domain}(\omega))$—which often is 1. Thus we get: $\omega(\mathcal{C})^c = \sum_{C_i \in \mathcal{C}} \binom{|C_i|}{2} \cdot M - \omega(\mathcal{C})$ and $\overline{\omega}(\mathcal{C})^c = (\binom{n}{2} - \sum_{C_i \in \mathcal{C}} \binom{|C_i|}{2}) \cdot M - \overline{\omega}(\mathcal{C})$. If parallel edges in a graph are allowed, the graph should be regarded as being weighted and again a maximum weight M should be set. Otherwise the latter definitions cannot be used.

$\omega(\mathcal{C})$, $\overline{\omega}(\mathcal{C})$
$\omega(\mathcal{C})^c$, $\overline{\omega}(\mathcal{C})^c$
M

[7] This quite probably introduces a loop on η; depending on the context this must not be forgotten.

Indices. We measure the quality of clusterings with a range of quality indices, discussed, e.g., in [46], however, we set our focus on the indices *modularity* [178] (mod), *inter-cluster conductance* [48] (icc), *coverage* [46] (cov) and *performance* [213] (perf) in this work, since they are the ones studied most. In the sections to come we shall regularly mention, discuss or exhibit peculiarities of these indices, thus we try to be brief and technical at this point. For earlier and (partly) profound discussions of these indices we refer the reader to the given references and to further pointers therein. We usually indicate weighted formulae by a subscript as in, e.g., cov_ω.

coverage

Coverage. The most simple index realizing a traditional measure of clustering quality is *coverage*. The *coverage*($\mathcal{C}$) of a graph clustering $\mathcal{C}$ is defined as the fraction of intra-cluster edges (or $\omega(\mathcal{C})$) within the complete set of edges (or W):

$$\mathrm{cov}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} = \frac{m(\mathcal{C})}{m(\mathcal{C}) + \overline{m}(\mathcal{C})} \qquad \mathrm{cov}_\omega(\mathcal{C}) := \frac{\omega(\mathcal{C})}{W} = \frac{\omega(\mathcal{C})}{\omega(\mathcal{C}) + \overline{\omega}(\mathcal{C})} \tag{1.2.1}$$

Intuitively, large values of coverage correspond to a good quality of a clustering. However, one principal drawback of coverage is, that the converse is not necessarily true: *Coverage* takes its largest value of 1 in the trivial case where there is only one cluster. Finding a clustering with $k \geq 3$ clusters with optimal *coverage*($\mathcal{C}$) is equivalent to finding a k-mincut, which is NP-hard [24];[8] moreover, requiring clusters to adhere to certain size constraints, such as some minimum size, is also NP-hard [216].

Figures 1.2.3-1.2.4 exhibit a general problem of quality indices: Although *coverage* is normed to the interval $[0, 1]$, it is still often hard to associate a specific value with a meaningful intuition of goodness. Usually some comparison or a rough idea about how well other graphs of a given family can be clustered is helpful, we shall see below how the index *modularity* attempts this. *Coverage* favors *coarse clusterings*, and its domain remains $[0, 1]$ for non-simple graphs and for weighted graphs. Despite all criticism, *coverage*'s simplicity and indisputability do make it a useful base measure, if one keeps in mind its penchant for coarseness.

coarse clusterings

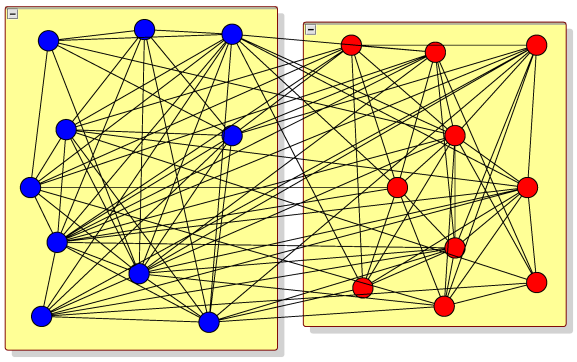


Figure 1.2.3. A random split of a $G(20, 0.5)$, i.e., taking 20 nodes and adding each possible edge $e \in \binom{V}{2}$ with probability 0.5; *coverage* is 0.66.

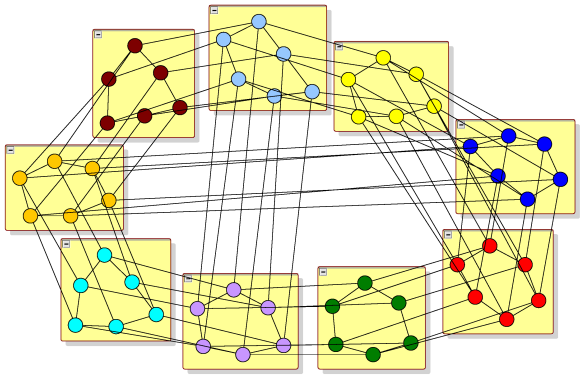


Figure 1.2.4. A meaningful clustering of a six-sided tube, which contains a few additional irregular interconnections; it yields a coverage of 0.43.

performance

Performance. The index *performance* partly remedies the main drawback of *coverage*. It is defined as the fraction of node pairs, that are clustered correctly, i.e. those connected

[8] Since this is an optimization problem, the corresponding decision problem is NP-hard and the actual problem is NPO-hard. For simplicity we omit this distinction, and shall do so in all of this work without further notice, aside from a somewhen reminder.

node pairs that are in the same cluster and those non-connected node pairs that are separated by the clustering. The unweighted case yields:

$$performance(\mathcal{C}) := \frac{m(\mathcal{C}) + \overline{m}^c(\mathcal{C})}{\frac{1}{2}n(n-1)} \tag{1.2.2}$$

The range of unweighted *performance* is $[0, 1]$ for simple graphs. Loops do not change things, but parallel edges do, as now the case mentioned in the above section applies: Sticking to the normalization in Equation 1.2.2 yields an unbounded domain of *performance*, and trying to adhere to the domain $[0, 1]$ requires harsher normalization by means of some maximum adjacency between nodes. This strongly suggests viewing this as a weighted problem, using a "reasonable" maximum weight M (or adjacency) between nodes, as follows:

M

$$performance_\omega = \frac{\omega(\mathcal{C}) + M\overline{m}(\mathcal{C})^c + M\overline{m}(\mathcal{C}) - \overline{\omega}(\mathcal{C})}{\frac{1}{2}n(n-1)M} \tag{1.2.3}$$

In the terms $\omega(\mathcal{C})^c$ and $\overline{\omega}(\mathcal{C})^c$ the weight between a non-adjacent pair of nodes is M, and the weight between (weakly) adjacent nodes is the gap between the weight of the corresponding edge and M. The literature does not agree on a value for M, however we strongly advocate using $\max(\text{domain}(\omega))$—or $\omega_{\max}$ if the former is not at hand—we shall make a case for this in Section 2.3; then the range of *performance* always is $[0, 1]$. Maximizing *performance* is NP-hard [202] as it is reducible to graph partitioning.

The drawback of performance is that in sparse networks, which most real-world networks indeed are, the value of $\overline{m}^c(\mathcal{C})$ clearly dominates the formula, supporting rather *fine clusterings*. For a rough impression, the clustering in Figure 1.2.2 has a *performance* of $(17 + 52)/78 \approx 0.89$; Figure 1.2.3 and Figure 1.2.4 yield 0.66 and 0.89, respectively.

fine clusterings

Inter-Cluster Conductance. *Inter-cluster conductance* (or *inter-cc*) measures the worst bottleneck constituted by cutting off a cluster from the graph, normalized by the degree sums thereby cut off. *Inter-cc* is based on the measure *conductance* [145], which seeks the "cheapest" cut $(S, V \setminus S)$ (with $S \subseteq V$) in a graph (measured by φ, the fractional term of Equation 1.2.4). The *conductance* of a clustering is then defined as the minimum *conductance* of each cluster. However, determining the minimum *conductance* cut in a graph is NP-hard [24], and thus this measure is ill-suited for measuring clustering quality. In turn, the cut induced by a cluster should have a very low *conductance* in a good clustering. Following [48] we can thus examine how good bottlenecks induced by clusters are (instead of all cuts *inside* a cluster), which yields the meaningful (and computable) formula given in Equation 1.2.4. We shape this measure such that it yields 1 for good clusterings. For brevity we only give a weighted formula, which can canonically be used for unweighted graphs ($\omega(v) \to \deg(v)$, $\omega(C) \to |E(C)|$):

inter-cluster conductance

bottleneck

conductance φ of a cut

$$\text{icc}_\omega(\mathcal{C}) := 1 - \max_{C \in \mathcal{C}} \underbrace{\frac{\omega(E(C, V \setminus C))}{\min\left(\sum_{v \in C} \omega(v), \sum_{v \in V \setminus C} \omega(v)\right)}}_{\text{conductance } \varphi \text{ of cut } (C, V \setminus C)} \tag{1.2.4}$$

Inter-cc is a worst-case measure, and this fact should be kept in mind. Thus, a clustering has a small *inter-cc*, if there exists at least one cluster C_0, that is rather strongly connected to $V \setminus C_0$, compared to the density of C_0 and $V \setminus C_0$. A non-isolated singleton cluster immediately results in a value of 0. In Figures 1.2.2, 1.2.3 and 1.2.4 *inter-cc* yields 0.80, 0.61 and 0.40, respectively. The range of *inter-cluster conductance* is always $[0, 1]$, even for unweighted and non-simple graphs. We refer the reader to [48] for a discussion on why not to use *intra-cluster conductance*, which is the analogon to *inter-cc* inside clusters.

worst-case measure

For clusterings of large real-world graphs it is rather common that *inter-cc* equals 0, due to some small degeneracy that may occur someplace. For this reason we also define the less capricious measure *average inter-cc*, in order to still have a meaningful measure of bottlenecks:

average inter-cc

$$\mathrm{icc}^{\mathrm{av}}_{\omega}(\mathcal{C}) := 1 - \frac{1}{|\mathcal{C}|}\sum_{C\in\mathcal{C}} \varphi(C, V \setminus C) \tag{1.2.5}$$

modularity

Modularity. Since in the following we shall dedicate two whole sections to exploring the intricacies of this index, we here keep the introduction of the measure *modularity* brief. *Modularity* has recently been proposed [178] in an attempt to find an apt remedy to the disadvantages of *coverage*. Citing the authors, the driving idea for *modularity* was to take *coverage "minus the expected value of the same quantity in a network with the same community divisions, but random connections between the vertices."* The stated formula was shortly after clarified in [57] and translates to

measure – expectation

$$\mathrm{mod}(\mathcal{C}) := \sum_{\{u,v\}\in V^{\times}} \left(\frac{A(u,v)}{m}\delta_{uv}\right) - \sum_{(u,v)\in V^2} \left(\frac{\deg(u)\cdot\deg(v)}{4m^2}\delta_{uv}\right) , \tag{1.2.6}$$

$$\text{with } \delta_{uv} = \begin{cases} 1 & \text{if } \mathcal{C}(u) = \mathcal{C}(v) \\ 0 & \text{otherwise} \end{cases} ,$$

and $A(u,v)$ = number of (parallel) edges between u and v.

using Kronecker's symbol δ_{uv} as an indicator function. However, the original formulation *did not* take into account loops and used as an enumerator for the first term $u \in V$, $v \in V$, thus miscounting *coverage* for non-simple graphs. On simple graphs the formulae coincide. Certainly this can be neglected in many practical cases, but we strongly suggest using the above formulation since the probabilistic model behind *modularity* requires loops and parallel edges to be sound. A simple way to equivalently rephrase Equation 1.2.6 is:

$$\mathrm{mod}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} - \frac{1}{4m^2}\sum_{C\in\mathcal{C}} \left(\sum_{v\in C} \deg(v)\right)^2 , \tag{1.2.7}$$

The literature has seen quite a few obfuscated or straight-out wrong formulations for *modularity*, due to the above issue or some otherwise sloppy enumeration. Since originally (and by a few follow-up studies) loops and parallel edges were disregarded, the original definitions are inconsistent, if such were allowed. Even worse, there have been studies that use a loop-agnostic definition of *modularity*, but then discuss under which preconditions loops can be simplified [23] without changing the *modularity*-optimal clustering. Mildly phrased, the results in that paper should not be trusted. Since the founding probabilistic assumptions for *modularity* are not sound without loops and parallel edges, as we shall see in a more thorough discussion in Section 2.3, it is meaningful to faithfully generalize the formulations in [178] and [57], as is done in Equations 1.2.6 and 1.2.7.

weighted edges

A generalization of *modularity* to weighted edges, such that its restriction to weights 0 and 1 yields the unweighted version, is straightforward, as proposed in [172]. We again state the formula we use, in order to disambiguate between formulations in previous works and to settle the loop-issue. Again our formula coincides with that of [172] for simple graphs.

$$\mathrm{mod}_{\omega}(\mathcal{C}) := \underbrace{\frac{\omega(\mathcal{C})}{W}}_{\mathrm{cov}_{\omega}} - \underbrace{\frac{1}{4W^2}\sum_{C\in\mathcal{C}} \left(\sum_{v\in C} \omega(v)\right)^2}_{\mathbb{E}(\mathrm{cov}_{\omega})} \tag{1.2.8}$$

The formula of *modularity* reveals an inherent trade-off: To maximize the first term, many edges should be contained in clusters, whereas the minimization of the second term is achieved by splitting the graph into many clusters with small total degrees each, or at least with a rather balanced total degree. The range of *modularity* is $[-0.5, 1]$ (see Lemma 2.2.1) for all graphs, where the least values are attained by bipartite graphs (and the obvious clustering) and 1 is approached by disjoint cliques. In Figures 1.2.2, 1.2.3 and 1.2.4 *modularity* yields

0.46, 0.13 and 0.32, respectively. The second value already indicates that *modularity* does not see much deviation from randomness in Figure 1.2.3, however we leave deeper discussions on *modularity* for the sections to come.

1.2.3 Basic Comparison Measures

In this thesis we will regularly have to quantify the similarity of two sets or of two vectors. For this basic task a number of formulae exist, and we briefly list the relevant ones in the following; for deeper insights and a much broader view we recommend [209]. Let $\vec{a}$ and $\vec{b}$ be two vectors of length n with entries $\vec{a}_i$ and $\vec{b}_j$, and let X and Y be two sets. In most cases using vectors poses a generalization of the formula for sets, however we explicitly list both cases for simplicity. The *simple matching coefficient* for $\vec{a}$ and $\vec{b}$ and for X and Y is given by

simple matching coefficient

$$\mathrm{sm}(\vec{a},\vec{b}) := \sum_{i=1}^{n}(\vec{a}_i \cdot \vec{b}_i) \qquad \mathrm{sm}(X,Y) := |X \cap Y| \ . \tag{1.2.9}$$

The lack of normalization prohibits the use of these formulae for representative quantification, which leads to the *cosine coefficient* defined by

cosine coefficient

$$\cos(\vec{a},\vec{b}) := \frac{\sum_{i=1}^{n}(\vec{a}_i \cdot \vec{b}_i)}{\sqrt{\sum_{j=1}^{n}\vec{a}_j^2}\cdot\sqrt{\sum_{\ell=1}^{n}\vec{b}_\ell^2}} \qquad \cos(X,Y) := \frac{|X \cap Y|}{\sqrt{|X|\cdot|Y|}} \ . \tag{1.2.10}$$

This measure is very commonly used. Considering the geometric interpretation of the eponymous cosine of the angle between vectors $\vec{a}$ and $\vec{b}$, we can see that this measure is bounded by the interval $[0,1]$. This measure has the property that it is length-invariant regarding vectors and that it is rather insensitive for values close to 1. The latter fact can be a drawback, and it is sometimes addressed by taking the *arccos* of $\cos(\vec{a},\vec{b})$ and renormalizing by 2π. The *overlap coefficient* takes a different point of view and is defined by

overlap coefficient

$$\mathrm{ov}(\vec{a},\vec{b}) := \frac{\sum_{i=1}^{n}(\min\{\vec{a}_i,\vec{b}_i\})}{\min\{\sum_{i=1}^{n}\vec{a}_i,\sum_{i=1}^{n}\vec{b}_i\}} \qquad \mathrm{ov}(X,Y) := \frac{|X \cap Y|}{\min\{|X|,|Y|\}} \ . \tag{1.2.11}$$

The *overlap coefficient* does not take the size of the larger set (or the values of the "larger" vector) into account, a fact which must be handled with care. Collinear vectors can even yield the same value as almost unrelated vectors, if one of the latter vectors is very long. The commonly used *Jaccard coefficient* remedies this disadvantage and is defined by

Jaccard coefficient

$$\mathrm{jac}(\vec{a},\vec{b}) := \frac{\sum_{i=1}^{n}(\vec{a}_i \cdot \vec{b}_i)}{\sum_{i=1}^{n}\vec{a}_i + \sum_{i=1}^{n}\vec{b}_i - \sum_{i=1}^{n}(\vec{a}_i \cdot \vec{b}_i)} \qquad \mathrm{jac}(X,Y) := \frac{|X \cap Y|}{|X \cup Y|} \ , \tag{1.2.12}$$

which is very intuitive—at least for sets. However, for vectors an extended *Jaccard coefficient* is more commonly used, the *Tanimoto coefficient*. This measure avoids a zero denominator, which can occur in the *Jaccard coefficient*, but at the same time it yields the *Jaccard coefficient* if restricted to sets, i.e., binary vectors:

Tanimoto coefficient

$$\tan(\vec{a},\vec{b}) := \frac{\sum_{i=1}^{n}(\vec{a}_i \cdot \vec{b}_i)}{\sum_{i=1}^{n}\vec{a}_i^2 + \sum_{i=1}^{n}\vec{b}_i^2 - \sum_{i=1}^{n}(\vec{a}_i \cdot \vec{b}_i)} \tag{1.2.13}$$

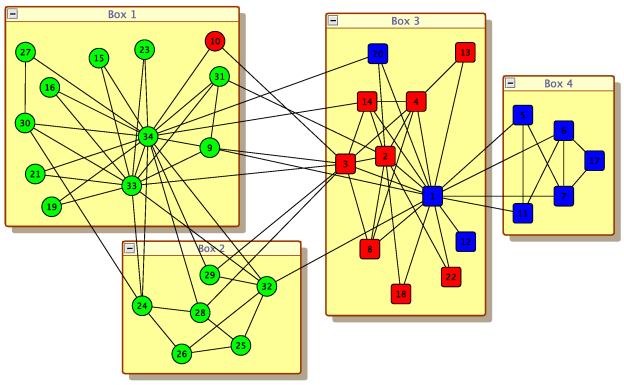
Finally, individual variants of so-called *match coefficients* for sets are commonly found in the literature. Among them are the following asymmetric and symmetric formulations, which we will explicitly point out when used:

match coefficients

$$\mathrm{ma}_{\mathrm{asym}}(X,Y) := \frac{|X \cap Y|}{|X|} \qquad \mathrm{ma}_{\mathrm{sym}}(X,Y) := \frac{|X \cap Y|}{\max\{|X|,|Y|\}} \tag{1.2.14}$$

Chapter 2

Static Graph Clustering



The fabled "karate club" network, Holy Grail of toy examples, archon of benchmark sets. Nodes model members of a university-based karate club and edges model social ties. Wayne W. Zachary assembled this data in the 70s in a sociological case study to explain why the club split up the way it did [230]. Caused by an *"unequal flow of sentiments and information across the ties"* of this social network, a *"factional division led to a formal separation of the club"*. Node shapes, left-hand circles vs. right-hand squares, correspond to the real division, boxes and colors indicate the result of *modularity* optimization and greedy maximization, respectively.

Contents

Section 2.1

Preface to Static Graph Clustering

'Oh yes,' said Frankie, 'but we'd have to get it out first. It's got to be prepared.'
'Treated,' said Benjy.
'Diced.'

(The Hitchhiker's Guide to the Galaxy, novel, Douglas Adams, Pan Books, 1979)

clustering algorithms

How can we identify a graph clustering? In short, this question is the essence of the field of graph clustering. While there are other important aspects of graph clustering such as measures to compare clusterings or means to represent a clustering in a well-perceivable way including visualizations, the core issue are algorithms for finding good graph clusterings, or *clustering algorithms*, in brief. The paradigm of *intra-cluster edge-density versus inter-cluster edge-sparsity* leads the way, but only on a very high and abstract level. Inextricably connected to this paradigm is its mathematical formalization in the shape of a quality index that measures how well the guideline is met. We have seen four such indices in Section 1.2.2. In fact, indices are an integral part of many clustering algorithms.

not graph partitioning

A more generous overview of the field calls for a few words about what graph clustering is not. The field of *graph partitioning* strongly differs from general graph clustering in that the number and possibly the size of clusters are crucial input parameters and the paradigm of graph clustering becomes a secondary criterion. As an example, an important application domain yielding very large graphs to be partitioned arises in the area of scientific computing: nodes of a problem instance are distributed evenly among a number of parallel processors in a way that—roughly speaking—minimizes the amount of delaying communication, i.e., the number of edges between the subsets [165]. For an overview and recent advances on the topic of graph partitioning we recommend [166, 165] and further references therein. Note furthermore that graph clustering is related but essentially different from the field of *data clustering* where data points are embedded in a high dimensional feature space and no explicit edge structure is present; for recent advances and a survey see, e.g., [40]. We refrain from pointing out specific discrepancies, as this has thoroughly been done in the literature [46, 89, 195]. In this thesis we exclusively treat clusterings as true partitions of the set of nodes of a graph. Thus, we do not allow clusters to *overlap* on the one hand, or leave nodes unclustered on the other hand. While in some applications this might be reasonable, it is a slightly different field indeed and thus not discussed herein. For further work on this topic see [223, 179, 76, 153], and in particular [170], where a first faithful generalization of *modularity* and other indices to overlapping clusterings is made. Another arguable point concerning the nature of graph clusterings is the paradigm of *parameter-free community discovery* [148]. In this thesis we do not fully agree that parameter-free methods are superior, although they do have undeniable advantages. A central point where a parameter can be of prime interest is granularity; the ability to explore clusterings at a coarser or a finer granularity can be essential.

not data clustering

no overlaps

parameter-free community discovery

2.1.1 Clustering Methodologies

Clustering algorithms can assume a bewildering variety of shapes. Roughly speaking, there are two dimensions in which graph clustering algorithms differ in a very general view: (i) How strongly does the method rely on a quality index? (ii) What hierarchical orientation does the algorithm take? In Figure 2.1.1 a couple of algorithms we shall encounter in this work are classified by these two criteria. On the y-axis (i), a method that does not rely on any specific quality index or a combination of indices usually performs some structural operations, such as contractions or percolation, which conform to the paradigm without actually touching an explicit, global index. Orthogonally to (i), algorithms can either iteratively agglomerate nodes into clusters ("bottom-up") or recursively cut the graph into clusters ("top-down"), which yields the x-axis (ii); either approach constructs a whole hierarchy of intermediate clusterings. Some algorithms do not work hierarchically ("non"), they either directly identify the clustering or shift nodes in a procedure of local optimization. The fact that most index-driven methods work hierarchically is partly due to the fact that most quality indices for graph clusterings have turned out to be NP-hard to optimize and rather resilient to effective approximations, see e.g., Section 2.2 and [216, 24, 145] for *modularity*, *coverage*, *performance* and *inter-cluster conductance*, respectively, allowing only heuristic approaches towards optimization. Stepwise cutting and agglomerating can easily be cast into such heuristics. The exception to this, a method based on *minimum-cut trees*, is described in Section 4.4. We clearly refrain from giving an extensive overview of the field and again refer the reader to the good overviews and introductions in [46, 89, 195]. In the following we discuss the clustering algorithms which are related to the work conducted in this chapter.

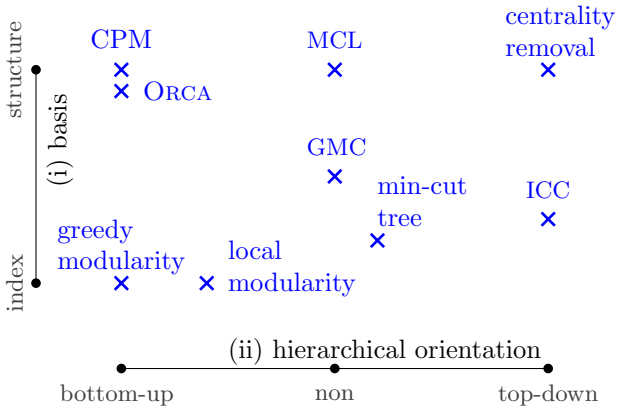


Figure 2.1.1. A rough classification of a few graph clustering algorithms described below

2.1.2 Greedy Agglomeration

greedy modularity maximization
greedy

It is common knowledge that there is no single best strategy for graph clustering, which justifies the plethora of existing approaches. An archetypical and particularly simple method, *greedy modularity maximization*, proposed in [57], ("global modularity" in Fig. 2.1.1), is probably the most widespread method applied in practice nowadays. This method clearly exemplifies an index-based bottom-up (agglomerative) algorithm, as it is solely based on *modularity* [178] and operates as follows: A temporary clustering $\tilde{\mathcal{C}}$ is initialized as the singleton clustering $\mathcal{C}^V$. Then, in an iteration of at most $n-1$ steps, those two clusters are *merged* of which a merge yields the highest increase in *modularity* among all pairs of clusters in $\tilde{\mathcal{C}}$. Figure 2.1.2 illustrates a late step of this procedure on our example graph. The grey clusters have already been assembled by a number of past merges, and the red-bordered cluster is the result of the current merge operation. The *dendrogram* (beside the graph in the figure) is a means to keep track of a clustering procedure; the bottom leaves of the *dendrogram* are the initial *singletons* and each internal node represents the merge of its two children. Since the internal nodes are ordered vertically, any horizontal line drawn through a *dendrogram* corresponds to

dendrogram

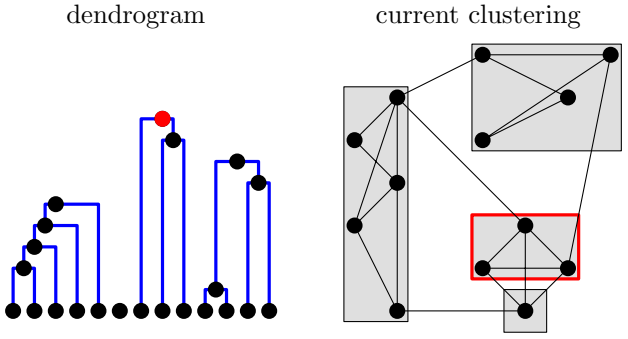


Figure 2.1.2. An intermediate step of an *index-driven* and *agglomerative* clustering approach, illustrated by the intermediate clustering $\tilde{\mathcal{C}}$ and the *dendrogram*

an intermediate clustering. Generally speaking, after $n-1$ steps, the quality of each intermediate clustering is measured and the one with the highest value is the final result. In the particular case of *modularity*, the single peak is reached as soon as no merge yields any positive increase. We shall detail this in the following two sections. In Figure 2.1.1 we coined this method "global", as it globally and greedily seeks the best agglomeration. We forego a deeper discussion of this method at this point as we return to it in Sections 2.2 and 2.3.

Variants and Similar Approaches. The technique proposed in [57] caused a surge of follow-up studies on various applications, possible adjustments and related methodologies, see, e.g., [90, 231, 168, 185]. Moreover, an array of heuristic algorithms has been proposed to optimize *modularity*. These are based on a greedy agglomeration [173, 57], on *spectral* division [175, 226], simulated annealing [126, 191], extremal optimization [81] and the Potts spin glass model [191] to name but a few prominent examples. A particularly close relative of it, which abandons global greedyness, has recently been presented in [38]. Here a significant speedup is achieved by only *locally* deciding about agglomeration and hierarchically reducing the graph repeatedly. This conceptually simple but effective local method of greedy *modularity* maximization constructs consecutive hierarchy levels of a clustering. By letting each node decide (on the basis of improving *modularity*) to which neighboring cluster/node to affiliate, clusters take shape. As soon as no node wishes to change its affiliation any more, each stable set of affiliated nodes is contracted. Then, on the graph of contracted nodes, this procedure is repeated. It is worth noting that, on a rough scale and concerning its scope of application, this approach is similar to Orca, a fast clustering technique which we shall describe and compare to the approach of [38] in Section 2.5. A crucial difference is that Orca builds a clustering without any bias towards *modularity* but instead relies on local graph-structural properties.

locally greedy

Applications of the above methods range from protein interaction dependencies to recommendation systems, social network analysis and even embeddings of neural networks (see, e.g., [231, 168, 185]). Although *modularity* has proven to be a rather reliable quality measure, it is known to behave artificially to some extent. A phenomenom that can regularly be observed in practice has been explored by Fortunato and Barthélemy [90]. They describe a *resolution limit* of *modularity*-based methods, a restrictive tendency of *modularity* to detect communities of specific size categories that depend on the size of the input. In my personal opinion, recent attempts to circumvent this *resolution limit* by either altering the index *modularity* or the agglomeration process as, e.g., presented in [62] so far failed to succeed. Some results in that work outright contradict the *resolution limit* [90]. Following a similar approach but a different aim, in [219] the authors try to speed up the agglomeration process. A balanced *dendrogram* of the clustering process is enforced by an altered objective function. The much finer resulting clustering, its quick computation and its lower *modularity* then do not come as a surprise, as they all are rather obvious consequences of an adulterated merging series, which faithfully obeys something else than *modularity*.

resolution limit

Another related approach has recently been described as a technical report in [153]. Here, clusters are built up by iterative node agglomerations according to a different fitness function, which is tunable for granularity (coarseness) and based on the ratio of connectedness within and between clusters. In a random order, nodes then build up neighborhoods around them by adding nodes that increase the fitness of their cluster. Although only unaffiliated nodes serve as new centers of neighborhoods, by intention this procedure can result in nodes belonging to more than one cluster. An interesting approach this work introduces is as follows: To find the most stable and significant clustering in a graph, the algorithm is run multiple times, each time with a slightly increased tuning parameter; then the largest range of the parameter for which the algorithm yields the same number of clusters is considered the most articulate in the graph.

2.1.3 Other Selected Clustering Approaches

In this section we point out and briefly describe a selection of algorithms for graph clustering which we will return to in this thesis. For further details on these algorithms we kindly refer the reader to the references given in the following. By any means this selection is far from comprehensive.

clique percolation method, CPM

Clique Percolation Method. Approaches which avoid the usage of a particular index for maximization often exhibit a certain elegance. Among these approaches is the *clique percolation method, CPM* [76, 182]. Although the authors allow this method to yield overlapping clusters, we briefly mention it here due to its interesting procedure: Starting with a small *clique* in the graph, e.g., a K_3 or a K_5, this subgraph "transpires" or "percolates" through the network as follows: Iteratively one node of the *clique* is swapped for another node in the graph, thus finding another neighboring *clique*. All regions identified by a connected *clique percolation* then induce clusters, and usually quite a few nodes are left unclustered. If overlaps are explicitly disallowed, this number increases. Drawbacks of this method include the difficult parametric choice of a suitable *clique* size and its rather high running time in practice; there are no proven theoretical bounds, to the best of my knowledge. An implementation of the *clique percolation method* including a GUI is available from the authors, and a reasonable extension to weighted networks is at hand. Figure 2.1.3 illustrates an outcome of CPM using a K_4.

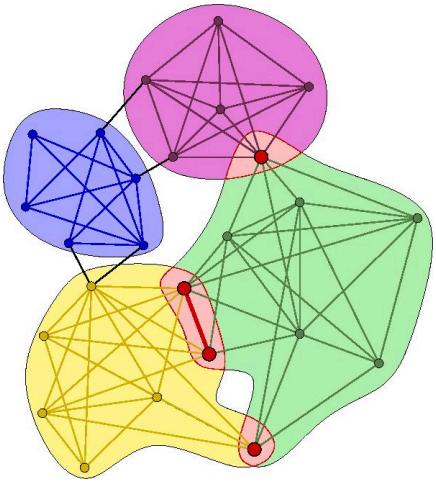


Figure 2.1.3. The clustering found by CPM, allowing overlaps (red nodes).

Removal of Central Edges. The rather exotic term *percolation* might come as a surprise for the above procedure, as it is nowhere close mathematical *percolation theory*, which investigates the properties of regular networks under a random edge-failure process. For this reason I initially mixed the above method up with the following approach, which is reminiscent of *percolation theory*: Iteratively remove the most *central* edges from a graph and stop at a predefined threshold; then, let the connected components induce the clusters. In Figure 2.1.1 this approach is named "centrality removal". It might seem counterintuitive to remove central edges, but consider *centrality measures* such as *shortest-path betweenness* (see Section 3.1.3): Here, an edge is very *central* if it lies on many shortest paths between pairs of nodes—something an *isthmus* certainly qualifies for. See Section 3.1 for details on *centrality*.

shortest-path betweenness

Recently Girvan and Newman [178] proposed algorithms which follow the above procedure. There is an interesting sidenote to their method. The authors required a good criterion for choosing at which stage during the removal process the best clustering is found. Probably unaware of existing measures, they presented *modularity* for this purpose. Only a few months later Newman et al. published a new method [57], described in Section 2.1.2 above, which solely relies on *modularity* maximization and abandons the arduous but meaningful removal process.

The running time of the removal algorithm [178] is dominated by the repeated recomputation of edge betweenness values, for which no efficient dynamization is known. Brandes' algorithm [41] computes edge betweenness most quickly, using linear space and $O(nm)$ and $O(nm + n^2 \log n)$ time in unweighted and weighted graphs, respectively; this yields a running time of $O(nm^2)$ for an unweighted clustering algorithm, which strongly limits the range of networks this method is applicable to. One remedy would be to intermit the eager recomputation of edge betweenness for a number $\ell > 1$ of edge removals. The authors of [178] claim that this seriously compromises clustering quality (*modularity* in their claim). The student thesis [37] of Abian Blome, a smart student of mine, disproved this claim to a certain extent. Blome introduced and evaluated two modifications to the vanilla algorithm: Betweenness values are recomputed every $\ell > 1$ steps, and only a certain range of the resulting clusterings

ℓ-betweenness

are actually measured, i.e., after only very few edge removals in a network of thousands of edges no measurement is performed. Roughly summarizing the results regarding this context yields that setting $\ell = \frac{n}{10}$ effects only a marginal decrease in *modularity* on a set of real-world instances and artificially generated random graphs, but a strong speed-up factor of $\frac{n}{5}$.

iterative conductance cutting ICC

Iterative Conductance Cutting. An established algorithm which partially relies on the bottleneck-based quality measure *conductance* (see Section 1.2.2) is *iterative conductance cutting*, abbreviated ICC. This top-down approach, introduced in [145], recursively splits the graph, always using a cut with low *conductance*. Since it is NP-hard to find a cut with minimum *conductance*, a polylogarithmic approximation is used instead: The nodes are ordered by their entries in the eigenvector of the second largest eigenvalue of the normalized adjacency matrix AdjN. Then that split of this order is used which yields a cut of minimum *conductance*. The algorithm stops cutting as soon as the current minimum-*conductance* cut exceeds a predefined threshold. We later use this established clustering method for comparison. The running time of this algorithm depends on the threshold and the method used for *eigenvalue computations*. We briefly jaunt into the ideas behind such *spectral* approaches below.

minimum-cut tree clustering

guaranteed bottleneck quality

Minimum-Cut Tree Clustering. We shall only be very brief here, as we dedicate Section 4.4 to this exceptional graph clustering approach [87], where we turn to a dynamic version of the algorithm. The exciting point about this approach is that it actually guarantees a certain quality of the clustering. A compact representation of the set of all minimum s-t-cuts in a modified graph G_α and certain *nesting properties* of this set are exploited such that its computation immediately yields a clustering that guarantees a meaningful bottleneck quality inside each cluster (high density) and between clusters (sparsity). An input parameter α tunes these guarantees and, in doing that, the coarseness of the clustering. Although the running time of this approach amounts to about $O(n^3\sqrt{m})$—depending on the employed method for computing minimum s-t-cuts—its algorithmic beauty and its unique feature to allow for a guarantee set this method apart from most other graph clustering approaches. In Section 4.4 we also briefly investigate how much quality we have to give up if we try to speed up the algorithm by only approximating minimum s-t-cuts.

Markov clustering, MCL

random walk

Markov Clustering. Introduced by van Dongen [213], *Markov clustering,* MCL simulates a *random walk* through a network. Such a *random walk* has a high probability to stay inside a dense cluster for a while before leaving it for a new cluster. Roughly speaking, instead of a true simulation, the transition matrix of the *random walk*, derived from $\mathrm{Adj}(G)$, is taken to the eth power in the *expansion* stage, which corresponds to computing the transition probabilities of a walk of length e. Then, in an *inflation* stage, each element of the matrix is taken to the rth power, in order to either emphasize ($r > 1$) heterogeneity in the likelihood of the *random walk* to linger at a node, or weaken it ($r < 1$). Repeatedly, these stages are performed and the matrix renormalized as to be *stochastic*, until either a fixed point or a recurrent state is reached. At that point the connected components the iterated transition matrix defines induce the clustering. It is argued [213] that in most cases such a stable condition is attained. The running time of MCL is dominated by the matrix multiplications and by the number of repetitions until convergence. With mild assumptions on the graph and an additional pruning step, a running time of $O(n\Delta^2)$ can be stated. While setting the parameters e and r to suitable values is not always trivial, we use this established and sound method in later comparisons.

GMC

Geometric Minimum Spanning Tree Clustering. The method *geometric minimum spanning tree clustering*, GMC [48], again uses a *spectral* approach. The eigenvectors of the d second largest eigenvalues of AdjN are used for a geometric embedding of the nodes as follows: In dimension ι the entries of the ιth eigenvector define the position of the nodes. Building a new graph G_g on the nodes in this geometric embedding, Euclidean distances serve as edge

weights. Then, a minimum spanning tree[1] T of G_g is built. A clustering can then be deduced by deleting all edges in T of weight (i.e., distance between nodes) greater than a threshold τ, and having the connected components of T define the clustering $\mathcal{C}_\tau$. As a neat property, the authors prove that the resulting clustering does *not* depend on the choice of T, in case this tree is not unique. Among the $n-1$ possible clusterings for different values of τ, the final clustering is then chosen to be that $\mathcal{C}_\tau$ which maximizes a given quality index (or the average of a combination of indices). We shall later include GMC in a comparison. Given only few eigenvectors are used, the running time of GMC is dominated by the computation of the minimum spanning tree which can be solved in time $O(m\alpha(m,n))$.[2]

spectral approaches

Spectral Approaches. Clustering techniques based on the *spectrum* of a graph have become numerous. Since in this thesis we use ICC, GMC and MCL, which either employ *eigenvectors* or are related to them, we here scratch the surface of *spectral* clustering. We recommend [215, 98] for good introductions and [56] for advanced topics. Considering the *Laplacian* of a graph G (see Section 1.2.1), a first and fundamental observation is as follows: The multiplicity of the *eigenvalue* 0 of L equals the number of connected components. We can see this by observing that the *indicator vector*[3] $\mathbb{1}_S$ of a connected component S is an *eigenvector* of L and that the set of all such *indicator vectors* spans the *eigenspace* of 0.

connected components

Since connected components are easy to identify anyway, consider G to be connected. Roughly speaking, finding a non-trivial cut of minimum weight in G is NP-hard [216], however, a *spectral* approach can be viewed as a *relaxed* version of such problems. As an example, consider the *ratio cut*, which is reasonably close to the paradigm of graph clustering: Find a k-partition $(A_1, \ldots, A_k)$ of V such that $\sum_{i=1}^{k}(\omega(A_i, V \setminus A_i)/|A_i|)$ is minimized. It can be shown that, if we drop the discreteness of assigning a node to a cluster A_i, we can, e.g., find an approximate *ratio cut* by the *eigenvector* of the second smallest *eigenvalue* of L.

ratio cut

The clustering algorithm MCL is based on random walks, where the transition from node v to one of its neighbors u is done with probability proportional to $\omega(v,u)$. Quite obviously, the matrix of transition probabilities is closely related to L and Adj, which again is the gist of the matter. We shall stop here and divert any further matters to the above references.

walktrap

Further Approaches. Related to MCL is an effective bottom-up strategy called *walktrap* [187] that iteratively updates a distance measure based on local random walks, which governs hierarchical agglomerations. This algorithm has been shown to be very fast in practice and to yield clusterings of good quality, which motivates our including it in a comparison to Orca in Section 2.5. A very recent and particularly interesting technique has been presented in [220]. The authors transfer the concept of *scan statistics* [111], which measure the density of data points in a sliding window, to graphs. The scan statistic the authors propose is coined *Poisson discrepancy*, and allows to assign a p-value[4] to each cluster, quantifying its statistical significance. A simple greedy algorithm is given for actually constructing a clustering, which offers a parameterized tuning of coarseness alongside an indicator how strong a chosen clustering is. Although this work makes strong assumptions on a random graph model underlying the probabilistic statements, I see much potential in this approach due to its sound setup to employ statistical significance and the success of scan statistics in the field of data mining. A comparative study of this algorithm with established methods has yet to be conducted.

scan statistics

[1] A *minimum spanning tree* is a spanning tree of minimum total edge weight.

[2] The function $\alpha(m,n)$ is the inverse Ackermann function, an extremely slowly growing function which in practice never is larger than 4.

[3] An n-dimensional vector with entry i equal to 1 if node v_i is in S, and 0 otherwise.

[4] Given a sample of a random experiment and assuming some *null hypothesis* (i.e., underlying probabilistic model), the *p-value* is the probability of obtaining a result at least as extreme (i.e., unfavorable to the null hypothesis) as the one that was observed in the experimental sample.

2.1.4 Summarizing Remarks

Countless formalizations of the paradigm of *intra-cluster edge-density versus inter-cluster edge-sparsity* exist, however, the overwhelming majority of algorithms for graph clustering relies on heuristics. The measure *modularity* and its greedy agglomerative maximization has immediately received considerable attention in several disciplines, and in particular in the complex systems literature. Apart from the fact that *modularity* turned out to be in good accordance with human intuition for a surprising variety of networks, three factors significantly sped its rise: (i) No single parameter has to (can) be tuned, such that no deeper knowledge about the algorithm is necessary when using one of the ready-to-use implementations available in the Internet. (ii) Both the formula of *modularity* and an implementation of the greedy algorithm are not at all demanding, mathematically. (iii) On reasonably modeled real-world instances, the greedy algorithm hardly ever produces nonsense clusterings. Putting aside all the drawbacks *modularity* and its greedy maximization have, these points together set them apart from a great lot of other clustering algorithms that might very well be at least as good in many applications. However, *modularity* is far from being the universal answer to graph clustering problems. Issues pointed out in previous works and in this thesis emphasize that other techniques are indispensable.

modularity's pros

questions

Motivating Questions. The main issues that motivate the work in this chapter on static graph clustering can be phrased as follows. *Modularity* has spread like wildfire, in turn both the theoretical and the systematic properties of this measure are not well understood, such that its massive use in practice as both a measure and an optimization criterion proverbially cries out for a thorough foundation. As the weird myth that the above greedy algorithm has an asymptotic running time of $O(n \log^2 n)$ in sparse graphs [57] is a mere myth indeed, a logical question is: How should we cluster graphs which this approach cannot handle due to their sheer size? Suppose we want to compare clusterings on a structural basis, either for finding a consensus, hand-pick the best solution or investigate the dynamics of a clustering, how can we quantify the distance between two clusterings?

answers

Answers in this Thesis. Addressing these questions we accomplish the following in this chapter. We provide a foundation for *modularity* and answer a number of open questions concerning this measure. In particular, we prove the conjectured hardness of optimizing *modularity* both in the general case and with the restriction to cuts. Comparing *modularity* to other quality indices and a ground truth clustering, we characterize the behavior of this measure. We provide theoretical bounds on the performance of the greedy agglomerative approach and systematically evaluate its behavior on artificially generated graphs. Moreover, we state and scrutinize the probability space which legitimates *modularity*, and integrate it into a framework of expectation-based measures, arriving at an ample corroboration of its feasibility. We give an integer linear programming formulation into which several indices fit, and perform an evaluation of first engineerings steps thereof. We propose and experimentally evaluate Orca, a clustering algorithm designed to handle huge graphs and to not be biased by a single index like *modularity*. We engineer comparison measures from the field of data mining so as to take into account the structure of a graph and systematically test their compliance to reasonable postulations for distance measures for graph clusterings, alongside that of traditional set-based measures.

Parts of this chapter have previously been published in [42, 43, 45, 44, 101, 68, 69, 70, 72, 115]. (We will point out the respective publications in the corresponding sections.)

2.1.5 Outlook

Among the many open problems we mention in the sections of this chapter, one particular issue is worth pointing out here, as it seems to be a logical next step. Although the literature has seen myriad alternative approaches towards faster agglomerative *modularity* maximization,

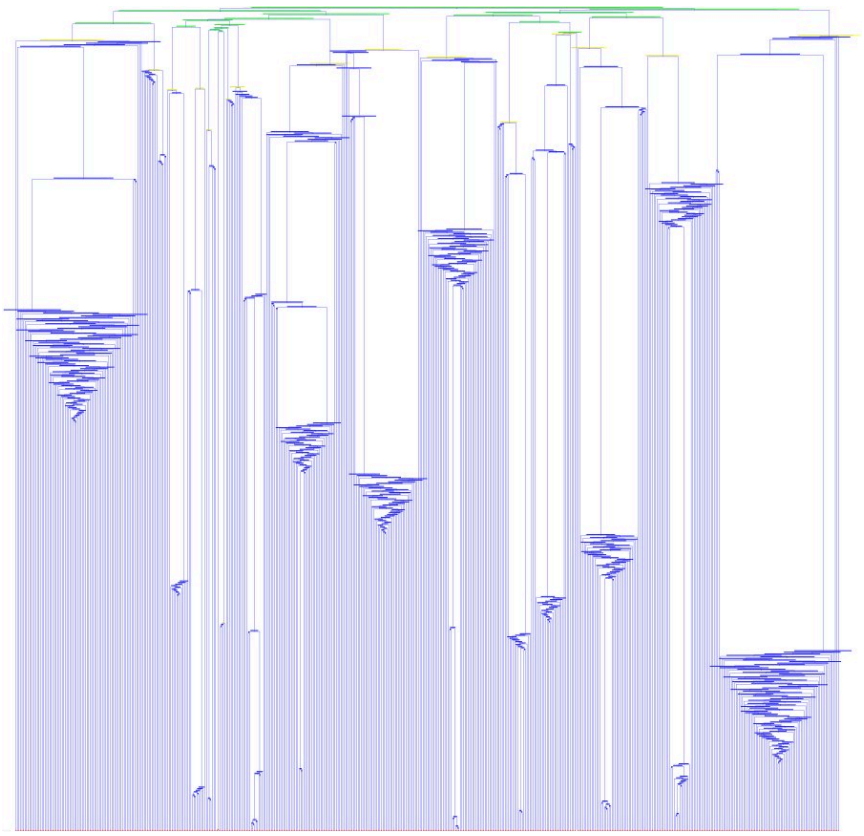


Figure 2.1.4. The beauty of a *dendrogram* should not overshadow its potential as a tool for algorithm analysis and engineering. Nodes start as singletons (red, bottom), their merges are represented by internal nodes (blue), until the algorithm's stopping criterion applies (yellow nodes); the remaining merges (green nodes) are thus not performed. Lumps in the dendrogram correspond to regions of the graph where the clustering algorithm performs numerous merge operations consecutively.

truly engineering an algorithm as to perform particularly well on specific species of graphs, such as those derived from road networks or others incorporating a strong power-law in their degree distribution, has not yet been done. Such an engineering would include the criterion speed, but also quality, as sometimes trapping local maxima of *modularity* will have to be avoided. In fact the *dendrogram* can be a valuable tool for this task, as it yields insights both about a clustering algorithm and about an instance, Figure 2.1.4 shows the *dendrogram* of the greedy *modularity* algorithm applied to a larger graph, a connected 1000-nodes excerpt from the DBLP [4] coauthorship graph (see Section 5.1.3). I see much potential in such an engineering, as it works towards more performant and reliable methods for the practitioner. I conjecture that a meta-heuristic could be included into a simple algorithm, choosing between, say, two different modes of operation depending on some quickly measurable property of the graph as, e.g., its clustering coefficient.

Taking a bird's view, the practitioner needs a very simple and decent heuristic for choosing an appropriate clustering algorithm in the first place. Using *modularity* introduces a strong bias into the result of a clustering algorithm, and for many applications, especially those with inhomogenously large and/or dense clusters, *modularity* probably fails to grasp the true community structure inside a network.

Section 2.2

On Modularity Clustering

That which is common to the greatest number has the least care bestowed on it.

(Aristotle, 384 BC – 322 BC, Greek philosopher, student of Plato, teacher of Alexander the Great; fortunately wrong for [42, 43, 44, 45])

JUST LIKE ANY OTHER QUALITY INDEX for clusterings (see, e.g., [46, 89] for *performance*, *coverage* and *intra-cluster conductance*), *modularity* certainly does have specific drawbacks such as *non-locality* and *scaling behavior* (see below) or *resolution limit* [90] as discussed in Section 1.2.2. However, being aware of these peculiarities, *modularity* can very well be considered a robust and useful measure that closely agrees with intuition on a wide range of real-world graphs, as observed by myriad studies.

In this section we study the problem of finding clusterings with maximum *modularity*, thus providing theoretical foundations for past and present work based on this measure. More precisely, we proof the conjectured [175] hardness of maximizing *modularity* both in the general case and the restriction to cuts, and give an integer linear programming formulation to facilitate optimization without enumerating all clusterings. Since the most commonly employed heuristic to optimize *modularity* is based on greedy agglomeration, we investigate its worst-case behavior. In fact, we give a graph family for which the greedy approach yields an approximation factor no better than two. In addition, our examples indicate that the quality of greedy clusterings may heavily depend on the tie-breaking strategy utilized—in the worst case, no approximation factor can be provided. These performance studies are concluded by clustering some previously considered networks optimally (via an ILP), which yields further insights.

Most of the content collected in this section has been published in at least one work I coauthored, the corresponding publications are [42, 43, 44, 45], which are based on joint work with Ulrik Brandes, Daniel Delling, Marco Gaertler, Martin Höfer, Zoran Nikoloski and Dorothea Wagner.[5] The driving force was always the long standing (at least in my personal reckoning) goal to prove the NP-completeness. In the end, of the rather large group of involved researchers, it was Martin Höfer who had the decisive idea for a reduction that worked.

Main Results

- Modularity can be defined as a normalized tradeoff between edges covered by clusters and squared cluster degree sums. (see Equation 2.2.1)

[5] Interestingly the "Symposium on Theoretical Aspects of Computer Science" (STACS) rejected our fundamental theoretical results on a measure which is massively used in practice by researchers in diverse fields as being insignificant. I couldn't disagree more.

- There is a formulation of modularity maximization as an integer linear program. (Section 2.2.2)
- There is a clustering with maximum modularity without *singleton* clusters of degree 1 and without clusters representing disconnected subgraphs. Isolated nodes have no impact on modularity. (Corollary 2.2.1, Lemmata 2.2.2, 2.2.3)
- The clustering of maximum modularity changes in a global, non-trivial fashion even for simplest graph perturbations. (Section 2.2.3.1)
- For any clustering $\mathcal{C}$ of any graph G it holds that $-\frac{1}{2} \leq \textit{modularity} \leq 1$. (Lemma 2.2.1)
- Finding a clustering with maximum modularity is NP-hard, both for the general case and when restricted to clusterings with exactly or at most two clusters. (Theorems 2.2.1 and 2.2.2)
- With a worst tie-breaking strategy, the greedy agglomeration algorithm has no worst-case approximation factor, with an arbitrary tie-breaking strategy the worst-case factor is at least 2. (Theorems 2.2.3 and 2.2.5)
- A clustering of maximum modularity for cliques of size n consists of a single cluster, for cycles of size n of approximately $\sqrt{n}$ clusters of size $\sqrt{n}$ each. (Theorems 2.2.6 and 2.2.7)

Future Work. Building upon the results of this section, the development of a clustering algorithm with provable performance guarantees should be addressed in the future. The special properties of the measure, its popularity in application domains and the absence of fundamental theoretical insights hitherto, render further mathematically rigorous treatment of *modularity* necessary, especially on specific classes of graphs.

2.2.1 Preliminaries

Throughout this section we will assume that graphs are connected, since this slightly simplifies notation; moreover, we restrict ourselves to unweighted, loop-free, simple graphs. Dropping these restrictions does not impair the hardness results which also implies that unweighted formulas for *modularity* are sufficient. Recalling the definitions from Section 1.2.2, we shall mostly be using the compact formula:

$$\operatorname{mod}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} - \frac{1}{4m^2} \sum_{C \in \mathcal{C}} \left(\sum_{v \in C} \deg(v) \right)^2 \quad , \tag{2.2.1}$$

However, for the derivation of an ILP formulation, which will be done first, the alternative formulation is more suitable, as it already incorporates Kronecker's symbol as a placeholder for a $\{0,1\}$-variable (recall $V^{\times}$ and V^2 from Section 1.2.2):

$$\operatorname{mod}(\mathcal{C}) = \sum_{\{u,v\} \in V^{\times}} \left(\frac{A(u,v)}{m} \delta_{uv} \right) - \sum_{(u,v) \in V^2} \left(\frac{\deg(u) \cdot \deg(v)}{4m^2} \delta_{uv} \right) \quad , \tag{2.2.2}$$

with Kronecker's symbol $\delta_{uv} = \begin{cases} 1 & \text{if } \mathcal{C}(u) = \mathcal{C}(v) \\ 0 & \text{otherwise} \end{cases}$,

and $A(u,v) =$ number of (parallel) edges between u and v.

2.2.2 Maximizing *Modularity* via Integer Linear Programming

The problem of maximizing *modularity* can be cast into a very simple and intuitive integer linear program (ILP). Given a graph $G = (V, E)$ with $n := |V|$ nodes, we define n^2 decision variables $X_{uv} \in \{0, 1\}$, one for every pair of nodes $u, v \in V$. The key idea is that these variables can be interpreted as an equivalence relation (over V) and thus form a clustering as follows:

equivalence relation

"Nodes u and v share a cluster iff $X_{uv} = 1$."

In order to ensure consistency, we need the following constraints, which guarantee

$$\begin{aligned} &\textit{reflexivity } \forall\, u\colon X_{uu} = 1 \ , \\ &\textit{symmetry } \forall\, u, v\colon X_{uv} = X_{vu} \ , \text{ and} \\ &\textit{transitivity } \forall\, u, v, w\colon \begin{cases} X_{uv} + X_{vw} - 2 \cdot X_{uw} & \leq & 1 \\ X_{uw} + X_{uv} - 2 \cdot X_{vw} & \leq & 1 \\ X_{vw} + X_{uw} - 2 \cdot X_{uv} & \leq & 1 \end{cases} . \end{aligned} \tag{2.2.3}$$

This formulation has been used, e.g., in [46] and in diverse variations for set partitioning problems, for a first pointer into that field see [184]. Observe that by *reflexivity*, in Equation 2.2.2 a loop on $v \in V$ always yields $\delta_{vv} = 1$, regardless of the shape of the clustering. The objective function of *modularity* can thus be simplified to the equivalent objective function

simplified objective function

$$\sum_{\{u,v\} \in \binom{V}{2}} \left(A(u, v) - \frac{\deg(u) \cdot \deg(v)}{2m} \delta_{uv} \right) \ . \tag{2.2.4}$$

This ILP can be simplified by pruning redundant variables and constraints, leaving only $\binom{n}{2}$ variables and $\binom{n}{3}$ constraints. We shall delve into further details and variant formulations in Section 2.4 and revisit an adaptation to *dynamic graphs* in Section 4.5.1.

2.2.3 Fundamental Observations

In the following, we identify basic structural properties that clusterings with maximum *modularity* fulfill. We first focus on the range of *modularity*, for which Lemma 2.2.1 gives the lower and upper bound.

mod $\in [-\frac{1}{2}, 1]$

Lemma 2.2.1 *Let G be an undirected and unweighted graph and $\mathcal{C} \in \Psi(G)$. Then $-1/2 \leq$ modularity ≤ 1 holds.*

Proof. Let $m_i = |E(C)|$ be the number of edges inside cluster C and $m_e = \sum_{C \neq C' \in \mathcal{C}} |E(C, C')|$ be the number of edges having exactly one end-node in C. Then the contribution of C to *modularity* is:

$$\frac{m_i}{m} - \left(\frac{m_i}{m} + \frac{m_e}{2m} \right)^2 \ .$$

This expression is strictly decreasing in m_e and, when varying m_i, the only maximum point is at $m_i = (m - m_e)/2$. The contribution of a cluster is minimized when m_i is zero and m_e is as large as possible. Suppose now $m_i = 0$, using the inequality $(a + b)^2 \geq a^2 + b^2$ for all non-negative numbers a and b, *modularity* has a minimum score for two clusters where all edges are inter-cluster edges. The upper bound is obvious from our reformulation in Equation (2.2.2), and has been observed previously [90, 231, 63]. It can only be actually attained in the specific case of a graph with no edges, where *coverage* (the first term) is defined to be 1. □

$E = \emptyset \Leftrightarrow$ mod $= 1$

As a result, any bipartite graph $K_{a,b}$ with the canonic clustering $\mathcal{C} = \{C_a, C_b\}$ yields the minimum *modularity* of $-1/2$. The following four results characterize the structure of a clustering with maximum *modularity*.

G bipartite $\rightsquigarrow$ mod $= -\frac{1}{2}$

Corollary 2.2.1 *Isolated nodes have no impact on modularity.*

isolated nodes don't matter

Corollary 2.2.1 directly follows from the fact that *modularity* depends on edges and degrees, thus, an isolated node does not contribute, regardless of its association to a cluster. Therefore, we exclude isolated nodes from further consideration in this work, i. e., all nodes are assumed to be of degree greater than zero.

$\deg(v) = 1$ $\Rightarrow v$ *no singleton*

Lemma 2.2.2 *A clustering with maximum modularity has no cluster that consists of a single node with degree 1.*

Proof. Suppose for contradiction that there is a clustering $\mathcal{C}$ with a cluster $C_v = \{v\}$ and $\deg(v) = 1$. Consider a cluster C_u that contains the neighbor node u. Suppose there are a number of m_i intra-cluster edges in C_u and m_e inter-cluster edges connecting C_u to other clusters. Together these clusters add

$$\frac{m_i}{m} - \frac{(2m_i + m_e)^2 + 1}{4m^2}$$

to *modularity*. Merging C_v with C_u results in a new contribution of

$$\frac{m_i + 1}{m} - \frac{(2m_i + m_e + 1)^2}{4m^2}$$

The merge yields an increase of

$$\frac{1}{m} - \frac{2m_i + m_e}{2m^2} > 0$$

in *modularity*, because $m_i + m_e \leq m$ and $m_e \geq 1$. This proves the lemma. □

disconn. clusters never necessary

Lemma 2.2.3 *There is always a clustering with maximum modularity, in which each cluster consists of a connected subgraph.*

Proof. Consider for contradiction a clustering $\mathcal{C}$ with a cluster C of m_i intra- and m_e inter-cluster edges that consists of a set of more than one connected subgraph. The subgraphs in C do not have to be disconnected in G, they are only disconnected when we consider the edges $E(C)$. Cluster C adds

$$\frac{m_i}{m} - \frac{(2m_i + m_e)^2}{4m^2}$$

to *modularity*. Now suppose we create a new clustering $\mathcal{C}'$ by splitting C into two new clusters. Let one cluster C_v consist of the component including node v, i.e., all nodes, which can be reached from a node v with a path running only through nodes of C, i.e., $C_v = \bigcup_{i=1}^{\infty} C_v^i$, where $C_v^i = \{w \mid \exists (w, w_i) \in E(C) \text{ with } w_i \in C_v^{i-1}\}$ and $C_v^0 = \{v\}$. The other nonempty cluster is given by $C - C_v$. Let C_v have m_i^v intra- and m_e^v inter-cluster edges. Together the new clusters add

create $\mathcal{C}'$ from $\mathcal{C}$

$$\frac{m_i}{m} - \frac{(2m_i^v + m_e^v)^2 + (2(m - m_i^v) + m - m_e^v)^2}{4m^2}$$

to *modularity*($\mathcal{C}'$). For $a, b \geq 0$ obviously $a^2 + b^2 \leq (a + b)^2$, and hence *modularity*($\mathcal{C}'$) $\geq$ *modularity*. □

no disconn. clusters

Corollary 2.2.2 *A clustering of maximum modularity does not include disconnected clusters.*

Corollary 2.2.2 directly follows from Lemma 2.2.3 and from the exclusion of isolated nodes.[6] Thus, the search for an optimum can be restricted to clusterings, in which clusters are connected subgraphs and there are no clusters consisting of nodes with degree 1.

[6] Observe that in the proof of Lem. 2.2.3 $\mathcal{C} = \mathcal{C}'$ can only hold if we allow isolated nodes, which we don't

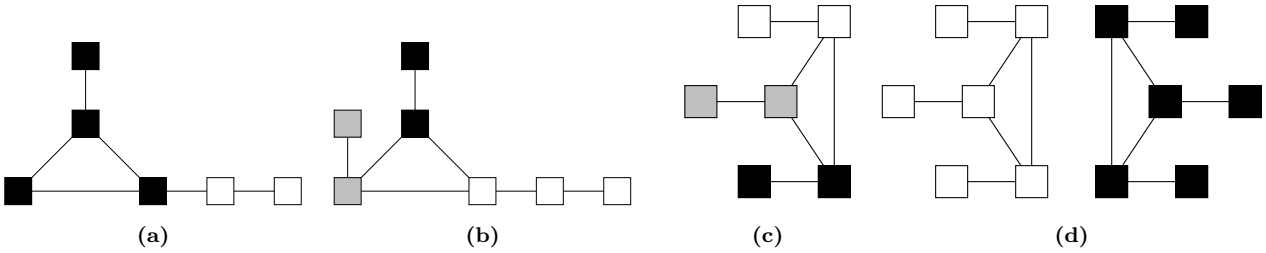


Figure 2.2.1. Clusters are represented by different grays. Comparing Figures (a) and (b) shows non-local behavior; the clique K_3 with leaves in Figure (c) is doubled in Figure (d), which shows scaling behavior.

2.2.3.1 Counterintuitive Behavior

In the last section, we listed some intuitive properties like connectivity within clusters for clusterings of maximum *modularity*. However, due to the enforced balance between *coverage* and the sums of squared cluster degrees, counter-intuitive situations arise. These are non-locality, scaling behavior, and sensitivity to satellites.

non-local behavior

Non-Locality. At first view, *modularity* seems to be a local quality measure. Recalling Equation 2.2.1, each cluster contributes separately. However, the examples presented in Figures 2.2.1a and 2.2.1b exhibit a typical non-local behavior. In these figures, clusters are represented by shades of gray. By adding an additional node connected to the leftmost node, the optimal clustering is altered completely. According to Lemma 2.2.2 the additional node has to be clustered together with the leftmost node. This leads to a shift of the rightmost black node from the black cluster to the white cluster, although locally its neighborhood structure has not changed.

singleton leaves disallowed

Sensitivity to Satellites. A *clique with leaves* is[7] a graph of $2n$ nodes that consists of a clique K_n and n *leaf* (or *satellite*) nodes of degree one, such that each node of the clique is connected to exactly one leaf node. For a clique we show in Section 2.2.6 that the trivial clustering with $k = 1$ has maximum *modularity*. For a *clique with leaves*, however, the optimal clustering changes to $k = n$ clusters, in which each cluster consists of a connected pair of *leaf* and clique nodes. Figure 2.2.1c shows an example.

doubling instances changes $\mathcal{C}$

Scaling Behavior. Figures 2.2.1c and 2.2.1d display the scaling behavior of *modularity*. By simply doubling the graph presented in Figure 2.2.1c, the optimal clustering is altered completely. While in Figure 2.2.1c we obtain three clusters each consisting of the *minor* K_2, the clustering with maximum *modularity* of the graph in Figure 2.2.1d consists of two clusters, each being a graph equal to the one in Figure 2.2.1c.

This behavior is in line with the previous observations in [90, 168], which state that the size and the structure of clusters in the optimum clustering depend on the total number of links in the network. Hence, clusters that are identified in smaller graphs might be combined to a larger cluster in a optimum clustering of a larger graph. The formulation of Equation 2.2.1 mathematically explains this observation as *modularity* optimization strives to optimize the trade-off between coverage and degree sums. This provides a rigorous understanding of the observations made in [90, 168].

[7] Typically, *leaves* are the degree-1 nodes of a *tree*.

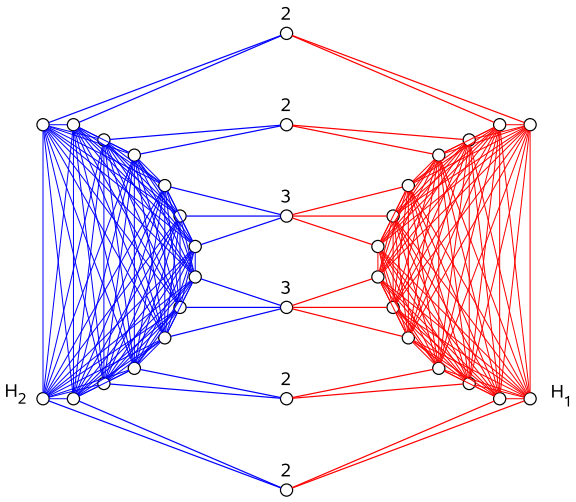


Figure 2.2.2. An example graph $G(A)$ for the instance $A = \{2, 2, 2, 2, 3, 3\}$ of 3-Partition. Node labels indicate the corresponding numbers $a_i \in A$.

2.2.4 NP-Completeness

It has been conjectured that maximizing *modularity* is hard [57], but no formal proof was provided. We next show that that decision version of modularity maximization is indeed NP-complete.

Problem 1 (Modularity) *Given a graph G and a number K, is there a clustering $\mathcal{C}$ of G, for which modularity$(\mathcal{C}) \geq K$?* Modularity

Note that we may ignore the fact that, in principle, K could be a real number in the range $[-1/2, 1]$, because $4m^2 \cdot$ *modularity*$(\mathcal{C})$ is integer for every partition $\mathcal{C}$ of G and polynomially bounded in the size of G. Our hardness result for Modularity is based on a transformation from the following decision problem.

Problem 2 (3-Partition) *Given $3k$ positive integer numbers $a_1, \ldots, a_{3k}$ such that the sum $\sum_{i=1}^{3k} a_i = kb$ and $b/4 < a_i < b/2$ for an integer b and for all $i = 1, \ldots, 3k$, is there a partition of these numbers into k sets, such that the numbers in each set sum up to b?* 3-Partition

We show that an instance $A = \{a_1, \ldots, a_{3k}\}$ of 3-Partition can be transformed into an instance $(G(A), K(A))$ of Modularity, such that $G(A)$ has a clustering with *modularity* at least $K(A)$, if and only if $a_1, \ldots, a_{3k}$ can be partitioned into k sets of sum $b = 1/k \cdot \sum_{i=1}^{k} a_i$ each.

It is crucial that 3-Partition is *strongly* NP-complete [106], i.e. the problem remains NP-complete even if the input is represented in unary coding. This implies that no algorithm can decide the problem in time polynomial even in the sum of the input values, unless $\mathcal{P} = \mathsf{NP}$. More importantly, it implies that our transformation need only be pseudo-polynomial. *strongly NP-complete*

The reduction is defined as follows. Given an instance A of 3-Partition, construct a graph $G(A)$ with k cliques (completely connected subgraphs) $H_1, \ldots, H_k$ of size $a = \sum_{i=1}^{3k} a_i$ each. For each element $a_i \in A$ we introduce a single *element node*, and connect it to a_i nodes in each of the k cliques in such a way that each clique member is connected to exactly one *element node*. It is easy to see that each clique node then has degree a and the *element node* corresponding to element $a_i \in A$ has degree ka_i. The number of edges in $G(A)$ is $m = k/2 \cdot a(a+1)$. See Figure 2.2.2 for an example. Note that the size of $G(A)$ is polynomial in the unary coding size of A, so that our transformation is indeed pseudo-polynomial. *the reduction* *element node*

Before specifying bound $K(A)$ for the instance of MODULARITY, we will show three properties of maximum *modularity* clusterings of $G(A)$. Together these properties establish the desired characterization of solutions for 3-PARTITION by solutions for MODULARITY.

no H_i is split

Lemma 2.2.4 *In a maximum modularity clustering of $G(A)$, none of the cliques $H_1, \ldots, H_k$ is split.*

reductio ad absurdum

We prove the lemma by showing that every clustering that violates the above condition can be modified in order to strictly improve *modularity*.

let $\mathcal{C}$ split H

Proof. We consider a clustering $\mathcal{C}$ that splits a clique $H \in \{H_1, \ldots, H_k\}$ into different clusters and then show how to obtain a clustering with strictly higher *modularity*. Suppose that $C_1, \ldots, C_r \in \mathcal{C}$, $r > 1$, are the clusters that contain nodes of H. For $i = 1, \ldots, r$ we denote by n_i the number of nodes of H contained in cluster C_i, $m_i = |E(C_i)|$ the number of edges between nodes in C_i, f_i the number of edges between nodes of H in C_i and *element nodes* in C_i, d_i be the sum of degrees of all nodes in C_i. The contribution of $C_1, \ldots, C_r$ to *modularity*$(\mathcal{C})$ is

$$\frac{1}{m}\sum_{i=1}^{r} m_i - \frac{1}{4m^2}\sum_{i=1}^{r} d_i^2 \ .$$

create $\mathcal{C}'$ from $\mathcal{C}$

Now suppose we create a clustering $\mathcal{C}'$ by rearranging the nodes in $C_1, \ldots, C_r$ into clusters $C', C'_1, \ldots, C'_r$, such that C' contains exactly the nodes of clique H, and each C'_i, $1 \leq i \leq r$, the remaining elements of C_i (if any). In this new clustering the number of covered edges reduces by $\sum_{i=1}^{r} f_i$, because all nodes from H are removed from the clusters C'_i. This labels the edges connecting the clique nodes to other non-clique nodes of C_i as inter-cluster edges. For H itself there are $\sum_{i=1}^{r}\sum_{j=i+1}^{r} n_i n_j$ edges that are now additionally covered due to the creation of cluster C'. In terms of degrees the new cluster C' contains a nodes of degree a. The sums for the remaining clusters C'_i are reduced by the degrees of the clique nodes, as these nodes are now in C'. So the contribution of these clusters to *modularity*$(\mathcal{C}')$ is given by

$$\frac{1}{m}\sum_{i=1}^{r}\left(m_i + \sum_{j=i+1}^{r} n_i n_j - f_i\right) - \frac{1}{4m^2}\left(a^4 + \sum_{i=1}^{r}(d_i - n_i a)^2\right) \ .$$

Δ = improvement

Setting $\Delta :=$ *modularity*$(\mathcal{C}')$ $-$ *modularity*$(\mathcal{C})$, we obtain

$$\begin{aligned}\Delta &= \frac{1}{m}\left(\sum_{i=1}^{r}\sum_{j=i+1}^{r} n_i n_j - f_i\right) + \frac{1}{4m^2}\left(\left(\sum_{i=1}^{r} 2d_i n_i a - n_i^2 a^2\right) - a^4\right) \\ &= \frac{1}{4m^2}\left((4m\sum_{i=1}^{r}\sum_{j=i+1}^{r} n_i n_j - 4m\sum_{i=1}^{r} f_i + \left(\sum_{i=1}^{r} n_i\left(2d_i a - n_i a^2\right)\right) - a^4\right) \ .\end{aligned}$$

Using the equation that $2\sum_{i=1}^{r}\sum_{j=i+1}^{r} n_i n_j = \sum_{i=1}^{r}\sum_{j\neq i} n_i n_j$, substituting $m = \frac{k}{2}a(a+1)$ and rearranging terms we get

$$\begin{aligned}\Delta &= \frac{a}{4m^2}\left(-a^3 - 2k(a+1)\sum_{i=1}^{r} f_i + \sum_{i=1}^{r} n_i\left(2d_i - n_i a + k(a+1)\sum_{j\neq i} n_j\right)\right) \\ &\geq \frac{a}{4m^2}\left(-a^3 - 2k(a+1)\sum_{i=1}^{r} f_i + \sum_{i=1}^{r} n_i\left(n_i a + 2kf_i + k(a+1)\sum_{j\neq i} n_j\right)\right) \ .\end{aligned}$$

For the last inequality we use the fact that $d_i \geq n_i a + k f_i$. This inequality holds because C_i contains at least the n_i nodes of degree a from the clique H. In addition, it contains both the clique and *element nodes* for each edge counted in f_i. For each such edge there are $k-1$ other edges connecting the element node to the $k-1$ other cliques. Hence, we get a contribution

of kf_i in the degrees of the *element nodes*. Combining the terms n_i and one of the terms $\sum_{j\neq i} n_j$ we obtain

$$
\begin{aligned}
&\Delta \\
&\geq \frac{a}{4m^2}\left(-a^3 - 2k(a+1)\sum_{i=1}^{r} f_i\right) + \frac{a}{4m^2}\left(\sum_{i=1}^{r} n_i\left(a\sum_{j=1}^{r} n_j + 2kf_i + ((k-1)a+k)\sum_{j\neq i}^{r} n_j\right)\right) \\
&= \frac{a}{4m^2}\left(-2k(a+1)\sum_{i=1}^{r} f_i + \sum_{i=1}^{r} n_i\left(2kf_i + ((k-1)a+k)\sum_{j\neq i}^{r} n_j\right)\right) \\
&= \frac{a}{4m^2}\left(\sum_{i=1}^{r} 2kf_i(n_i - a - 1)) + ((k-1)a+k)\sum_{i=1}^{r}\sum_{j\neq i}^{r} n_i n_j\right) \\
&\geq \frac{a}{4m^2}\left(\sum_{i=1}^{r} 2kn_i(n_i - a - 1) + ((k-1)a+k)\sum_{i=1}^{r}\sum_{j\neq i}^{r} n_i n_j\right) \quad .
\end{aligned}
$$

For the last step we note that $n_i \leq a-1$ and $n_i - a - 1 < 0$ for all $i = 1, \ldots, r$. So increasing f_i decreases the *modularity* difference. For each node of H there is at most one edge to a node not in H, and thus $f_i \leq n_i$. By rearranging terms and using the inequality $a \geq 3k$ we get

$$
\begin{aligned}
\Delta &\geq \frac{a}{4m^2}\sum_{i=1}^{r} n_i\left(2k(n_i - a - 1) + ((k-1)a+k)\sum_{j\neq i}^{r} n_j\right) \\
&= \frac{a}{4m^2}\sum_{i=1}^{r} n_i\left(-2k + ((k-1)a-k)\sum_{j\neq i}^{r} n_j\right) \\
&\geq \frac{a}{4m^2}((k-1)a - 3k)\sum_{i=1}^{r}\sum_{j\neq i}^{r} n_i n_j \\
&\geq \frac{3k^2}{4m^2}(3k-6)\sum_{i=1}^{r}\sum_{j\neq i}^{r} n_i n_j \quad .
\end{aligned}
$$

As we can assume $k > 2$ for all relevant instances of 3-Partition, we obtain $\Delta > 0$. This shows that any clustering can be improved by merging each clique completely into a cluster. □ Next, we observe that the optimum clustering places at most one clique completely into a single cluster.

$\Rightarrow \Delta > 0$

Lemma 2.2.5 *In a maximum modularity clustering of $G(A)$, every cluster contains at most one of the cliques $H_1, \ldots, H_k$.*

$H_i \subseteq C \Rightarrow H_j \nsubseteq C$

Proof. Consider a maximum *modularity* clustering. Lemma 2.2.4 shows that each of the k cliques $H_1, \ldots, H_k$ is entirely contained in one cluster. Assume that there is a cluster C which contains at least two of the cliques. If C does not contain any *element nodes*, then the cliques form disconnected components in the cluster. In this case it is easy to see that the clustering can be improved by splitting C into distinct clusters, one for each clique. In this way we keep the number of edges within clusters the same, however, we reduce the squared degree sums of clusters.

reductio ad absurdum

no element node ⇒ easy

Otherwise, we assume C contains $l > 1$ cliques completely and in addition some element nodes of elements a_j with $j \in J \subseteq \{1, \ldots, k\}$. Note that inside the l cliques $la(a-1)/2$ edges are covered. In addition, for every *element node* corresponding to an element a_j there are la_j edges included. The degree sum of the cluster is given by the la clique nodes of degree a and some number of *element nodes* of degree ka_j. The contribution of C to *modularity*($\mathcal{C}$) is

some element nodes

thus given by

$$\frac{1}{m}\left(\frac{l}{2}a(a-1)+l\sum_{j\in J}a_j\right)-\frac{1}{4m^2}\left(la^2+k\sum_{j\in J}a_j\right)^2 .$$

create $\mathcal{C}'$ from $\mathcal{C}$

Now suppose we create $\mathcal{C}'$ by splitting C into C'_1 and C'_2 such that C'_1 completely contains a single clique H. This leaves the number of edges covered within the cliques the same, however, all edges from H to the included *element nodes* eventually drop out. The degree sum of C'_1 is exactly a^2, and so the contribution of C'_1 and C'_2 to *modularity*($\mathcal{C}'$) is given by

$$\frac{1}{m}\left(\frac{l}{2}a(a-1)+(l-1)\sum_{j\in J}a_j\right)-\frac{1}{4m^2}\left(\left((l-1)a^2+k\sum_{j\in J}a_j\right)^2+a^4\right) .$$

Δ again

Considering the difference $\Delta = \textit{modularity}(\mathcal{C}') - \textit{modularity}(\mathcal{C})$ we note that

$$\begin{aligned}
\Delta &= -\frac{1}{m}\sum_{j\in J}a_j+\frac{1}{4m^2}\left((2l-1)a^4+2ka^2\sum_{j\in J}a_j-a^4\right)\\
&= \frac{2(l-1)a^4+2ka^2\sum_{j\in J}a_j}{4m^2}-\frac{4m\sum_{j\in J}a_j}{4m^2}\\
&= \frac{2(l-1)a^4-2ka\sum_{j\in J}a_j}{4m^2}\\
&\geq \frac{9k^3}{2m^2}(9k-1)\\
&> 0,
\end{aligned}$$

$\Delta > 0$ again

as $k > 0$ for all instances of 3-Partition. Since the clustering is improved in every case, it is not optimal. This is a contradiction. □

The previous two lemmas show that any clustering can be strictly improved to a clustering that contains k *clique clusters*, such that each one completely contains one of the cliques $H_1, \ldots, H_k$ (possibly plus some additional *element nodes*). In particular, this must hold for the optimum clustering as well. Now that we know how the cliques are clustered we turn to the element nodes. As they are not directly connected, it is never optimal to create a cluster consisting only of *element nodes*. Splitting such a cluster into *singleton* clusters, one for each *element node*, reduces the squared degree sums but keeps the *edge coverage* at the same value. Hence, such a split yields a clustering with strictly higher modularity. The next lemma shows that we can further strictly improve the *modularity* of a clustering with a *singleton* cluster of an *element node* by joining it with one of the clique clusters.

no excl. clusters for element-nodes

Lemma 2.2.6 *In a maximum modularity clustering of $G(A)$, there is no cluster composed of element nodes only.*

reductio ad absurdum

Proof. Consider a clustering $\mathcal{C}$ of maximum *modularity* and suppose that there is an *element node* v_i corresponding to the element a_i, which is not part of any clique cluster. As argued above we can improve such a clustering by creating a *singleton* cluster $C = \{v_i\}$. Suppose $C_{\min}$ is the clique cluster, for which the sum of degrees is minimal. We know that $C_{\min}$ contains all nodes from a clique H and eventually some other *element nodes* for elements a_j with $j \in J$ for some index set J. The cluster $C_{\min}$ covers all $a(a-1)/2$ edges within H and $\sum_{j\in J} a_j$ edges to *element nodes*. The degree sum is a^2 for clique nodes and $k\sum_{j\in J} a_j$ for *element nodes*. As C is a *singleton* cluster, it covers no edges and the degree sum is ka_i. This yields a contribution of C and $C_{\min}$ to *modularity*($\mathcal{C}$) of

$$\frac{1}{m}\left(\frac{a(a-1)}{2}+\sum_{j\in J}a_j\right)-\frac{1}{4m^2}\left(\left(a^2+k\sum_{j\in J}a_j\right)^2+k^2a_i^2\right) .$$

Again, we create a different clustering $\mathcal{C}'$ by joining C and $C_{\min}$ to a new cluster C'. This increases the *edge coverage* by a_i. The new cluster C' has the sum of degrees of both previous clusters. The contribution of C' to *modularity*$(\mathcal{C}')$ is given by

create $\mathcal{C}'$ from $\mathcal{C}$

$$\frac{1}{m}\left(\frac{a(a-1)}{2} + a_i + \sum_{j\in J} a_j\right) - \frac{1}{4m^2}\left(a^2 + ka_i + k\sum_{j\in J} a_j\right)^2 ,$$

so that, using $\Delta = \textit{modularity}(\mathcal{C}') - \textit{modularity}(\mathcal{C})$

Δ again

$$\begin{aligned}\Delta &= \frac{a_i}{m} - \frac{1}{4m^2}\left(2ka^2a_i + 2k^2a_i\sum_{j\in J} a_j\right)\\ &= \frac{1}{4m^2}\left(2ka(a+1)a_i - 2ka^2a_i - 2k^2a_i\sum_{j\in J} a_j\right)\\ &= \frac{a_i}{4m^2}\left(2ka - 2k^2\sum_{j\in J} a_j\right) .\end{aligned}$$

At this point recall that $C_{\min}$ is the clique cluster with the minimum degree sum. For this cluster the elements corresponding to included *element nodes* can never sum to more than a/k. In particular, as v_i is not part of any clique cluster, the elements of nodes in $C_{\min}$ can never sum to more than $(a-a_i)/k$. Thus,

$$\sum_{j\in J} a_j \le \frac{1}{k}(a-a_i) < \frac{1}{k}a ,$$

and so $\Delta > 0$. This contradicts the assumption that $\mathcal{C}$ is optimal. □

$\Delta > 0$ again

We have shown that for the graphs $G(A)$ the clustering of maximum modularity consists of exactly k clique clusters, and each element node belongs to exactly one of the clique clusters. Combining the above results, we now state our main result:

Theorem 2.2.1 MODULARITY *is strongly NP-complete.*

NP-completeness

Proof. For a given clustering $\mathcal{C}$ of $G(A)$ we can check in polynomial time whether *modularity*$(\mathcal{C}) \ge K(A)$, so clearly MODULARITY $\in$ NP.

MODULARITY $\in$ NP

For NP-completeness we transform an instance $A = \{a_1, \ldots, a_{3k}\}$ of 3-PARTITION into an instance $(G(A), K(A))$ of MODULARITY. We have already outlined the construction of the graph $G(A)$ above. For the correct parameter $K(A)$ we consider a clustering in $G(A)$ with the properties derived in the previous lemmas, i. e., a clustering with exactly k clique clusters. Any such clustering yields exactly $(k-1)a$ inter-cluster edges, so the *edge coverage* is given by

transferring K

$$\begin{aligned}\sum_{C\in\mathcal{C}} \frac{|E(C)|}{m} &= \frac{m-(k-1)a}{m}\\ &= 1 - \frac{2(k-1)a}{ka(a+1)} = 1 - \frac{2k-2}{k(a+1)} .\end{aligned}$$

Hence, the clustering $\mathcal{C} = (C_1, \ldots, C_k)$ with maximum *modularity* must minimize $\deg(C_1)^2 + \deg(C_2)^2 + \ldots + \deg(C_k)^2$. This requires a distribution of the element nodes between the clusters which is as even as possible with respect to the sum of degrees per cluster. In the optimum case we can assign to each cluster *element nodes* corresponding to elements that sum to $b = 1/k \cdot a$. In this case the sum up of degrees of *element nodes* in each clique cluster

must evenly distr. element nodes

is equal to $k \cdot 1/k \cdot a = a$. This yields $\deg(C_i) = a^2 + a$ for each clique cluster C_i, $i = 1, \ldots, k$, and gives

$$\deg(C_1)^2 + \ldots + \deg(C_k)^2 \geq k(a^2 + a)^2 = ka^2(a+1)^2.$$

Equality holds only in the case, in which an assignment of b to each cluster is possible. Hence, if there is a clustering $\mathcal{C}$ with *modularity*($\mathcal{C}$) of at least

$$K(A) = 1 - \frac{2k-2}{k(a+1)} - \frac{ka^2(a+1)^2}{k^2a^2(a+1)^2} = \frac{(k-1)(a-1)}{k(a+1)}$$

... solves 3-Partition

then we know that this clustering must split the *element nodes* perfectly to the k clique clusters. As each *element node* is contained in exactly one cluster, this yields a solution for the instance of 3-Partition. With this choice of $K(A)$ the instance $(G(A), K(A))$ of Modularity is satisfiable only if the instance A of 3-Partition is satisfiable.

if and only if

Otherwise, suppose the instance for 3-Partition is satisfiable. Then there is a partition into k sets such that the sum over each set is $1/k \cdot a$. If we cluster the corresponding graph by joining the element nodes of each set with a different clique, we get a clustering of *modularity* $K(A)$. This shows that the instance $(G(A), K(A))$ of Modularity is satisfiable if the instance A of 3-Partition is satisfiable. This completes the reduction and proves the theorem. □

holds for mod ω

This result naturally holds also for the straightforward generalization of maximizing *modularity* in weighted graphs. Instead of using the numbers of edges the definition of modularity employs the sum of edge weights for edges within clusters, between clusters and in the total graph.

2.2.4.1 Special Case: *Modularity* with a Bounded Number of Clusters

A common clustering approach is based on iteratively identifying cuts which are good with respect to some quality measures, see for example [17, 130, 146]. The general problem being NP-complete, we now complete our hardness results by proving that the restricted optimization problem is hard as well. More precisely, we consider the two problems of computing the clustering with maximum *modularity* that splits the graph into exactly or at most two clusters. Although these are two different problems, our hardness result will hold for both versions, hence, we define the problem cumulatively.

k-Modularity

Problem 3 (*k*-Modularity) *Given a graph G and a number K, is there a clustering $\mathcal{C}$ of G into exactly/at most k clusters, for which modularity($\mathcal{C}$) $\geq K$?*

We provide a proof using a reduction that is similar to the one given recently for showing the hardness of the MinDisAgree problem of correlation clustering [108]. We use the problem Minimum Bisection for Cubic Graphs (MB3) for the reduction:

MB3

Problem 4 (Minimum Bisection for Cubic Graphs) *Given a 3-regular graph G with n nodes and an integer c, is there a clustering into two clusters of $n/2$ nodes each such that it cuts at most c edges?*

the reduction

This problem has been shown to be strongly NP-complete in [51]. We construct an instance of 2-Modularity from an instance of MB3 as follows. For each node v from the graph $G = (V, E)$ we attach $n-1$ new nodes and construct an n-clique. We denote these cliques as $cliq(v)$ and refer to them as *node clique* for $v \in V$. Hence, in total we construct n different new cliques, and after this transformation each node from the original graph has degree $n+2$. Note that a cubic graph with n nodes has exactly $1.5n$ edges. In our adjusted graph there are exactly $m = (n(n-1)+3)n/2$ edges.

node clique cliq(v)

$\mathcal{C}^*$

We will show that an optimum clustering which is denoted as $\mathcal{C}^*$ of 2-Modularity in the adjusted graph has exactly two clusters. Furthermore, such a clustering corresponds to a

minimum bisection of the underlying MB3 instance. In particular, we give a bound K such that the MB3 instance has a bisection cut of size at most c if and only if the corresponding graph has 2-modularity at least K.

We begin by noting that there is always a clustering $\mathcal{C}$ with $\mathit{modularity}(\mathcal{C}) > 0$. Hence, $\mathcal{C}^*$ must have exactly two clusters, as no more than two clusters are allowed. This serves to show that our proof works for both versions of 2-modularity, in which at most or exactly two clusters must be found.

$\exists\, \mathcal{C}$ *with* mod >0

Lemma 2.2.7 *For every graph constructed from a* MB3 *instance, there exists a clustering* $\mathcal{C} = \{C_1, C_2\}$ *such that modularity* > 0. *In particular, the clustering* $\mathcal{C}^*$ *has two clusters.*

Proof. Consider the following partition into two clusters. We pick the nodes of $\mathit{cliq}(v)$ for some $v \in V$ as C_1 and the remaining graph as C_2. Then

$$\begin{aligned}\mathit{modularity} &= 1 - \frac{3}{m} - \frac{(n(n-1)+3)^2 + ((n-1)(n(n-1)+3))^2}{4m^2} \\ &= \frac{2n-2}{n^2} - \frac{3}{m} = \frac{2}{n} - \frac{2}{n^2} - \frac{3}{m} \\ &> 0 \ ,\end{aligned}$$

as $n \geq 4$ for every cubic graph. Hence $\mathit{modularity} > 0$ and the lemma follows. □

Next, we show that in an optimum clustering, all the nodes of one *node clique* $\mathit{cliq}(v)$ are located in one cluster:

$\mathit{cliq}(v)$ *en bloc*

Lemma 2.2.8 *For every node* $v \in V$ *there exists a cluster* $C \in \mathcal{C}^*$ *such that* $\mathit{cliq}(v) \subseteq C$.

reductio ad absurdum

Proof. For contradiction we assume a *node clique* $\mathit{cliq}(v)$ for some $v \in V$ is split in two clusters C_1 and C_2 of the clustering $\mathcal{C} = \{C_1, C_2\}$. Let $k_i := |C_i \cap \mathit{cliq}(v)|$ be the number of nodes located in the corresponding clusters, with $1 \leq k_i \leq n-1$. Note that $k_2 = n - k_1$. In addition, we denote the sum of node degrees in both clusters excluding nodes from $\mathit{cliq}(v)$ by d_1 and d_2:

$$d_i = \sum_{u \in C_i, u \notin \mathit{cliq}(v)} \deg(u).$$

Without loss of generality assume that $d_1 \geq d_2$. Finally, we denote by m' the number of edges covered by the clusters C_1 and C_2.

create $\mathcal{C}'$ *from* $\mathcal{C}$

We define a new clustering $\mathcal{C}'$ as $\{C_1 \setminus \mathit{cliq}(v), C_2 \cup \mathit{cliq}(v)\}$ and denote the difference of the *modularity* as $\Delta := \mathit{modularity}(\mathcal{C}') - \mathit{modularity}(\mathcal{C})$. We distinguish two cases depending in which cluster the node v was located with respect to $\mathcal{C}$: In the first case $v \in C_2$ and we obtain:

Δ

case 1: $v \in C_2$

$$\begin{aligned}\mathit{modularity}(\mathcal{C}) &= \frac{m'}{m} - \frac{(d_1 + k_1(n-1))^2}{4m^2} + \frac{(d_2 + (n-k_1)(n-1)+3)^2}{4m^2} \ , \\ \mathit{modularity}(\mathcal{C}') &= \frac{m' + k_1(n-k_1)}{m} - \frac{d_1^2 + (d_2 + n(n-1)+3)^2}{4m^2} \quad \text{and} \\ \Delta &= \frac{k_1(n-k_1)}{m} - \frac{d_1^2 + (d_2 + n(n-1)+3)^2}{4m^2} \\ &\quad + \frac{(d_1 + k_1(n-1))^2}{4m^2} + \frac{(d_2 + (n-k_1)(n-1)+3)^2}{4m^2} \ .\end{aligned}$$

We simplify expression of Δ as follows:

$$\begin{aligned}\Delta &= \frac{1}{4m^2}\Big(4mk_1(n-k_1) - d_1^2 - (d_2 + n(n-1)+3)^2 + (d_1 + k_1(n-1))^2 + (d_2 + (n-k_1)(n-1)+3)^2\Big) \\ &= \frac{1}{4m^2}\Big(4mk_1(n-k_1) + (2k_1^2 - 2nk_1)(n-1)^2 - 6k_1(n-1) + 2(d_1 - d_2)k_1(n-1)\Big) \\ &\geq \frac{k_1}{4m^2}\Big(4m(n-k_1) - 2(n-k_1)(n-1)^2 - 6(n-1)\Big) \ .\end{aligned}$$

We can bound the expression in the bracket in the following way by using the assumption that $d_1 \geq d_2$ and $1 \leq k_1 \leq n-1$:

$$\begin{aligned}&(n-k_1)\Big(4m-2(n-1)^2\Big)-6(n-1)\\ &\geq (n-k_1)\Big(\underbrace{4m-2(n-1)^2-6(n-1)}_{=:B}\Big)\end{aligned} \tag{2.2.5}$$

and, thus, it remains to show that $B > 0$. By filling in the value of m and using the facts that $2n^2(n-1) > 2(n-1)^2$ and $6n > 6(n-1)$ for all $n \geq 4$, we obtain $B > 0$ and thus *modularity* strictly improves if all nodes are moved from $cliq(v)$ to C_2.

$\Rightarrow cliq(v) \subseteq C_2$

case 2: $v \in C_1$

In the second case the node $v \in C_1$ and we get the following equations:

$$\begin{aligned}modularity(\mathcal{C}) &= \frac{m'}{m} - \frac{(d_1+k_1(n-1)+3)^2}{4m^2} + \frac{(d_2+(n-k_1)(n-1))^2}{4m^2} \quad ,\\ modularity(\mathcal{C}') &= \frac{m'+k_1(n-k_1)}{m} - \frac{d_1^2+(d_2+n(n-1)+3)^2}{4m^2} \quad \text{, and}\\ \Delta &= \frac{k_1(n-k_1)}{m} - \frac{d_1^2+(d_2+n(n-1)+3)^2}{4m^2}\\ &\quad + \frac{(d_1+k_1(n-1)+3)^2}{4m^2} + \frac{(d_2+(n-k_1)(n-1))^2}{4m^2} \quad .\end{aligned}$$

We simplify expression of Δ as follows:

$$\begin{aligned}4m^2\Delta &= 4mk_1(n-k_1) + (2k_1^2-2nk_1)(n-1)^2\\ &\quad - 6(n-k_1)(n-1) + 2(d_1-d_2)(k_1(n-1)+3)\\ &\geq 4mk_1(n-k_1) - 2k_1(n-k_1)(n-1)^2 - 6(n-k_1)(n-1))\end{aligned}$$

Recall $1 \leq k_1 \leq n-1$, and filling in the value of m, we obtain

$$4mk_1 - 2k_1(n-1)^2 - 6(n-1) = 2k_1(n^2(n-1)-(n-1)^2) + 6nk_1 - 6(n-1) > 0 \quad ,$$

which holds for all $k_1 \geq 1$ and $n \geq 4$. Also in this case, *modularity* strictly improves if all nodes are moved from $cliq(v)$ to C_2. □

$\Rightarrow cliq(v) \subseteq C_2$

The final lemma before defining the appropriate input parameter K for the 2-Modularity and thus proving the correspondence between the two problems shows that the clusters in the optimum clusterings have the same size.

$\mathcal{C}^*$ evenly divides node-cliques

Lemma 2.2.9 *In $\mathcal{C}^*$, each cluster contains exactly $n/2$ complete node cliques.*

reductio ad absurdum

Proof. Suppose for contradiction that one cluster C_1 has $l_1 < n/2$ cliques. For completeness of presentation we use m' to denote the unknown (and irrelevant) number of edges covered by the clusters. For the *modularity* of the clustering is given in Equation 2.2.6.

$$modularity(\mathcal{C}^*) = \frac{m'}{m} - \frac{l_1^2(n(n-1)+3)^2}{4m^2} - \frac{(n-l_1)^2(n(n-1)+3)^2}{4m^2} \tag{2.2.6}$$

create $\mathcal{C}'$ from $\mathcal{C}$

We create a new clustering $\mathcal{C}'$ by transferring a complete *node clique* from cluster C_2 to cluster C_1. As the graph G is 3-regular, we lose at most 3 edges in the *coverage* part of *modularity*:

$$\begin{aligned}modularity(\mathcal{C}') \quad \geq \quad & \frac{m'-3}{m} - \frac{(l_1+1)^2(n(n-1)+3)^2}{4m^2}\\ &+ \frac{(n-l_1-1)^2(n(n-1)+3)^2}{4m^2} \quad .\end{aligned} \tag{2.2.7}$$

We can bound the difference $\Delta = \mathit{modularity}(\mathcal{C}') - \mathit{modularity}(\mathcal{C})$ in the following way: Δ

$$\begin{aligned}\Delta &\geq -\frac{3}{m} + \frac{(l_1^2 + (n-l_1)^2}{4m^2} - \frac{(n-l_1-1)^2)(n(n-1)+3)^2}{4m^2} \\ &= -\frac{3}{m} + \frac{(2n-4l_1-2)}{n^2} \\ &\geq -\frac{3}{m} + \frac{2}{n^2} \\ &= \frac{2}{n^2} - \frac{6}{n^3-n^2+3n} \\ &> 0 \ ,\end{aligned}$$

for all $n \geq 4$. The analysis uses the fact that we can assume n to be an even number, so $l_1 \leq \frac{n}{2} - 1$ and thus $4l_1 \leq 2n - 4$. $\Delta > 0$

This shows that we can improve every clustering by balancing the number of complete node cliques in the clusters – independent of the loss in *edge coverage*. □
Finally, we can state theorem about the complexity of 2-Modularity:

Theorem 2.2.2 *2-*Modularity *is strongly NP-complete.* *NP-completeness*

Proof. Let (G, c) be an instance of Minimum Bisection for Cubic Graphs, then we construct a new graph G' as stated above and define $K := 1/2 - c/m$. *transferring K*

As we have shown in Lemma 2.2.9 that each cluster of $\mathcal{C}^*$ that is an optimum clustering of G' with respect to 2-Modularity has exactly $n/2$ complete node cliques, the sum of degrees in the clusters is exactly m. Thus, it is easy to see that if the clustering $\mathcal{C}^*$ meets the following inequality

$$\mathit{modularity}(\mathcal{C}^*) \geq 1 - \frac{c}{m} - \frac{2m^2}{4m^2} = \frac{1}{2} - \frac{c}{m} = K \ ,$$

then the number of inter-cluster edges can be at most c. Thus the clustering $\mathcal{C}^*$ induces a balanced cut in G with at most c cut edges. □

This proof is particularly interesting as it highlights that maximizing modularity in general is hard due to the hardness of minimizing the squared degree sums on the one hand, whereas in the case of two clusters this is due to the hardness of minimizing the edge cut.

2.2.5 The Greedy Algorithm

In contrast to the abovementioned iterative cutting strategy, another commonly used approach to find clusterings with good quality scores is based on greedy agglomeration [99, 140]. In the case of *modularity*, this approach is particularly widespread [173, 57]. In Section 2.3 we conduct a systematic evaluation of the practical behavior of this algorithm on generated graphs; here we focus on theoretical results and on a few examples from the literature. *greedy agglomeration*

The greedy algorithm starts with the *singleton* clustering and iteratively merges those two clusters that yield a clustering with the best *modularity*, i. e., the largest increase or the smallest decrease is chosen. After $n - 1$ merges, the clustering that achieved the highest *modularity* is returned. The algorithm maintains a symmetric matrix Δ with entries $\Delta_{i,j} := \mathit{modularity}(\mathcal{C}_{i,j}) - \mathit{modularity}$, where $\mathcal{C}$ is the current clustering and $\mathcal{C}_{i,j}$ is obtained from $\mathcal{C}$ by merging clusters C_i and C_j. Note that there can be several pairs i and j such that $\Delta_{i,j}$ is the maximum, in these cases the algorithm selects an arbitrary pair. The pseudo-code for the greedy algorithm is given in Algorithm 1. An efficient implementation using sophisticated data-structures requires $\mathcal{O}\left(n^2 \log n\right)$ runtime. Note that $n - 1$ iterations is an upper bound and one can terminate the algorithms when the matrix Δ contains only non-positive entries. We call this property *single-peakedness*, it is proven in [57]. Since it is NP-hard to maximize modularity in general graphs, it is unlikely that this greedy algorithm is optimal. In fact, we *single-peakedness*

Algorithm 1: Greedy Algorithm for Maximizing *Modularity*

Input: graph $G = (V, E)$
Output: clustering $\mathcal{C}$ of G
$\mathcal{C} \leftarrow$ singletons
initialize matrix Δ
while $|\mathcal{C}| > 1$ **do**
  find $\{i, j\}$ with $\Delta_{i,j}$ is the maximum entry in the matrix Δ
  merge clusters i and j
  update Δ
return clustering with highest *modularity*

sketch a graph family, where the above greedy algorithm has an approximation factor of 2, asymptotically. In order to prove this statement, we introduce a general construction scheme given in Definition 2.1. Furthermore, we point out instances where a specific way of breaking ties of merges yield a clustering with *modularity* of 0, while the optimum clustering has a strictly positive score.

Modularity is defined such that it takes values in the interval $[-1/2, 1]$ for any graph and any clustering. In particular the *modularity* of a trivial clustering placing all vertices into a single cluster has a value of 0. We use this technical peculiarity to show that the greedy algorithm has an unbounded approximation ratio.

no finite apx. factor

Theorem 2.2.3 *There is no finite approximation factor for the greedy algorithm for finding clusterings with maximum* modularity*.*

Proof. We present a class of graphs, on which the algorithm obtains a clustering of value 0, but for which the optimum clustering has value close to 1/2. A graph G of this class is given by two cliques (V_1, E_1) and (V_2, E_2) of size $|V_1| = |V_2| = n/2$, and $n/2$ matching edges E_m connecting each vertex from V_1 to exactly one vertex in V_2 and vice versa. See Figure 2.2.3 for an example with $n = 14$. Note that we can define modularity by associating weights $w(u, v)$ with every existing and non-existing edge in G as follows:

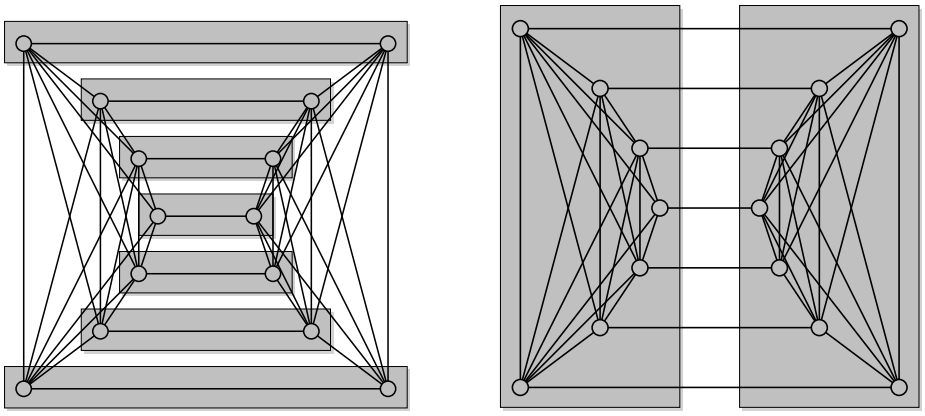
$$w(u, v) = \frac{E_{uv}}{2m} - \frac{\deg(u)\deg(v)}{4m^2} \quad ,$$

where $E_{uv} = 1$ if $(u, v) \in E$ and 0 otherwise. The *modularity* of a clustering $\mathcal{C}$ is then derived by the summing the weights of the edges covered by $\mathcal{C}$

$$\textit{modularity}(\mathcal{C}) = \sum_{C \in \mathcal{C}} \sum_{u,v \in C} w(u, v)$$

Note that in this formula we have to count twice the weight for each edge between different vertices u and v (once for every ordering) and once the weight for a non-existing self-loop for every vertex u. Thus, the change of modularity by merging two clusters is given by twice the sum of weights between the clusters.

Now consider a run of the greedy algorithm on the graph of Figure 2.2.3. Note that the graph is $n/2$-regular, and thus has $m = n^2/4$ edges. Each existing edge gets a weight of $2/n^2 - 1/n^2 = 1/n^2$, while every non-existing edge receives a weight of $-1/n^2$. As the self-loop is counted by every clustering, the initial trivial *singleton* clustering has *modularity* value of $-1/n$. In the first step each cluster merge along any existing edge results in an increase of $2/n^2$. Of all these equivalent possibilities we suppose the algorithm chooses to merge along an edge from E_m to create a cluster C'. In the second step merging a vertex with C' results in change of 0, because one existing and one non-existing edge would be included. Every other merge along an existing edge still has value $2/n^2$. We suppose the algorithm again chooses to merge two *singleton* clusters along an edge from E_m creating a



(a) **(b)**

Figure 2.2.3. (a) Clustering with modularity 0; (b) Clustering with modularity close to $\frac{1}{2}$

cluster C''. Afterwards observe that merging clusters C' and C'' yields a change of 0, because two existing and two non-existing edges would be included. Thus, it is again optimal to merge two *singleton* clusters along an existing edge. If the algorithm continues to merge *singleton* clusters along the edges from E_m, it will in each iteration make an optimal merge resulting in strictly positive increase in *modularity*. After $n/2$ steps it has constructed a clustering $\mathcal{C}$ of the type depicted in Figure 2.2.3a. $\mathcal{C}$ consists of one cluster for the vertices of each edge of E_m and has a *modularity* value of

$$\mathit{modularity}(\mathcal{C}) = \frac{2}{n} - \frac{n}{2} \cdot \frac{4n^2}{n^4} = 0.$$

Due to the *single-peakedness* of the problem [57] all following cluster merges can never increase this value, hence the algorithm will return a clustering of value 0.

On the other hand consider a clustering $\mathcal{C}^* = \{C_1, C_2\}$ with two clusters, one for each clique $C_1 = V_1$ and $C_2 = V_2$ (see Figure 2.2.3b). This clustering has a *modularity* of

$$\mathit{modularity}(\mathcal{C}^*) = \frac{n(n-2)}{n^2} - 2\frac{4n^2}{16n^2} = \frac{1}{2} - \frac{2}{n}.$$

This shows that the approximation ratio of the greedy algorithm can be infinitely large, because no finite approximation factor can outweigh a value of 0 with one strictly greater than 0. □

proof exploits non-determinism

The key observation is, that the proof considers a worst-case scenario in the sense that greedy is in each iteration supposed to pick exactly the "worst" merge choice of several equivalently attractive alternatives. If greedy chooses in an early iteration to merge along an edge from E_1 or E_2, the resulting clustering will be significantly better. As mentioned earlier, this negative result is due to formulation of *modularity*, which yields values from the interval $[-1/2, 1]$. For instance, a linear remapping of the range of *modularity* to the interval $[0, 1]$, the greedy algorithm yields a value of $1/3$ compared to the new optimum score of $2/3$. In this case the approximation factor would be 2.

w/o non-determinism

Next, we provide a decreased lower bound for a different class of graphs and no assumptions on the random choices of the algorithm.

Definition 2.1 *Let $G = (V, E)$ and $H = (V', E')$ be two non-empty, simple, undirected, and unweighted graphs and let $u \in V'$ be a node. The* product *$G \star_u H$ is defined as the*

$G \star_u H$

graph (V'', E'') *with the nodeset* $V'' := V \cup V \times V'$ *and the edgeset* $E'' := E \cup E''_c \cup E''_H$ *where*

$$\begin{aligned} E''_c &:= \big\{\{v,(v,u)\} \mid v \in V\big\} \qquad \textit{and} \\ E''_H &:= \big\{\{(v,v'),(v,w')\} \\ &\qquad \mid v \in V, v', w' \in V'', \{v',w'\} \in E\big\} \ . \end{aligned}$$

An example is given in Figure 2.2.4. The product $G \star_u H$ is a graph that contains G and for each node v of G a copy H_v of H. For each copy the node in H_v corresponding to $u \in H$ is connected to v. We use the notation (v, w') to refer to the copy of node w' of H, which is located in H_v. In the following we consider only a special case: Let $n \geq 2$ be an integer, $H = (V', E')$ be an undirected and connected graph with at least two nodes, and $u \in V'$ an arbitrary but fixed node. We denote by $\mathcal{C}^g_k$ the clustering obtained with the greedy algorithm applied to $K_n \star_u H$ starting from *singletons* and performing at most k steps that all have a positive increase in *modularity*. Furthermore, let m be the number of edges in $K_n \star_u H$.

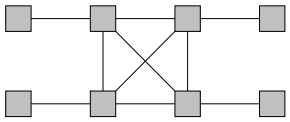


Figure 2.2.4. Example: the graph $K_4 \star_u P_1$.

Based on the merging policy of the greedy algorithm we can characterize the final clustering $\mathcal{C}^g_n$. It has n clusters, each of which includes a vertex v of G and his copy of H.

modularity of $K_n \star_u H$

Theorem 2.2.4 *Let* $n \geq 2$ *be an integer and* $H = (V', E')$ *be a undirected and connected graph with at least two nodes. If* $2|E'| + 1 < n$ *then the greedy algorithm returns the clustering* $\mathcal{C}^g := \{\{v\} \cup \{v\} \times V' \mid v \in V\}$ *for* $K_n \star_u H$ *(for any fixed* $u \in H$*). This clustering has a modularity score of*

$$4m^2 \cdot \mathit{modularity}(\mathcal{C}^g) = 4m\left((|E'|+1)\cdot n\right) - n\left(2|E'|+1+n\right)^2 \ .$$

proof omitted

The proof of Theorem 2.2.4, which relies on the graph construction described above, it can be found in an associated technical report [43], but is omitted here for brevity. The next corollary reveals that the clustering, in which G and each copy of H form individual clusters, has a greater *modularity* score. We first observe an explicit expression for *modularity*.

modularity of $\mathcal{C}^s$

Corollary 2.2.3 *The clustering* $\mathcal{C}^s$ *is defined as* $\mathcal{C}^s := \{V\} \cup \{\{v\} \times V' \mid v \in V\}$ *and, according to Equation 2.2.2, its modularity is*

$$\begin{aligned} 4m^2 \cdot \mathit{modularity}(\mathcal{C}^s) &= 4m\left(|E'|n + \binom{n}{2}\right) - n\left(2|E'|+1\right)^2 \\ &\quad - \left(n\cdot(n-1+1)\right)^2 \ . \end{aligned}$$

If $n \geq 2$ *and* $2|E'| + 1 < n$*, then clustering* $\mathcal{C}^s$ *has higher modularity than* $\mathcal{C}^g$*.*

apx.-factor ≥ 2

Theorem 2.2.5 *The approximation factor of the greedy algorithm for finding clusterings with maximum modularity is at least 2.*

The quotient $\mathit{modularity}(\mathcal{C}^s)/\mathit{modularity}(\mathcal{C}^g)$ asymptotically approaches 2 for n going to infinity on $K_n \star_u H$ with H a path of length $1/2\sqrt{n}$. The full proof of Theorem 2.2.5 is also available in [43].

2.2.6 Optimality Results

2.2.6.1 Characterization of Cliques and Cycles

In this section, we provide several results on the structure of clusterings with maximum modularity for cliques and cycles. This extends previous work, in particular [90], in which cycles and cycles of cliques were used to reason about global properties of *modularity*. For

readability of the many small results we generally postpone proofs to the mini-appendix of this section without further notice. A first observation is that *modularity* can be simplified for general d-regular graphs as follows.

proofs in appendix

modularity on d-regular graphs

Corollary 2.2.4 *Let $G = (V, E)$ be an unweighted d-regular graph and $\mathcal{C} = \{C_1, \ldots, C_k\} \in \Psi(G)$. Then the following equality holds:*

$$\textit{modularity} = \frac{|E(C)|}{dn/2} - \frac{1}{n^2}\sum_{i=1}^{k}|C_i|^2 \quad . \tag{2.2.8}$$

The correctness of the corollary can be read off the definition given in Equation 2.2.2 and the fact that $|E| = d|V|/2$. Thus, for regular graphs *modularity* only depends on cluster sizes and coverage.

Cliques We first deal with the case of complete graphs. Corollary 2.2.5 provides a simplified formulation for *modularity*. From this rewriting, the clustering with maximum *modularity* can directly be obtained.

modularity of K_n

Corollary 2.2.5 *Let K_n be a complete graph on n nodes and $\mathcal{C} := \{C_1, \ldots, C_k\} \in \Psi(K_n)$. Then the following equality holds:*

$$\textit{modularity} = -\frac{1}{n-1} + \frac{1}{n^2(n-1)}\sum_{i=1}^{k}|C_i|^2 \quad . \tag{2.2.9}$$

Thus, maximizing *modularity* is equivalent to maximizing the squares of cluster sizes. Using the general inequality $(a+b)^2 \geq a^2 + b^2$ for non-negative real numbers, the clustering with maximum *modularity* is the 1–clustering. More precisely:

modularity of K_{kn}

Theorem 2.2.6 *Let k and n be integers, K_{kn} be the complete graph on $k \cdot n$ nodes and $\mathcal{C}$ a clustering such that each cluster contains exactly n elements. Then the following equality holds:*

$$\textit{modularity} = \left(-1 + \frac{1}{k}\right) \cdot \frac{1}{kn-1} \quad .$$

For fixed $k > 1$ and as n tends to infinity, modularity is always strictly negative, but tends to zero. Only for $k = 1$ modularity is zero and thus is the global maximum.

As Theorem 2.2.6 deals with one clique, the following corollary provides the optimal result for k disjoint cliques.

modularity of $k \times K_n$

Corollary 2.2.6 *The maximum modularity of a graph consisting of k disjoint cliques of size n is $1 - 1/k$.*

The corollary follows from the definition of *modularity* in Equation 2.2.2. Corollary 2.2.6 gives a glimpse on how previous approaches have succeeded to upper bound *modularity* as it was pointed out in the context of Lemma 2.2.1.

modularity on cycles

Cycles Next, we focus on simple cycles, i. e., connected 2-regular graphs. According to Equation 2.2.8, *modularity* can be expressed as given in Equation 2.2.10, if each cluster is connected which may safely be assumed (see Corollary 2.2.2).

$$\textit{modularity} = \frac{n-k}{n} - \frac{1}{n^2}\sum_{i=1}^{k}|C_i|^2 \quad . \tag{2.2.10}$$

In the following, we prove that clusterings with maximum *modularity* are balanced with respect to the number and the sizes of clusters. First we characterize the distribution of cluster sizes for clusterings with maximum *modularity*, fixing the number k of clusters. For convenience, we minimize $F := 1 - \textit{modularity}$, where the argument of F is the distribution of the cluster sizes.

Proposition 2.2.1 *Let k and n be integers, the set $D^{(k)} := \left\{ x \in \mathbb{N}^k \,\middle|\, \sum_{i=1}^{k} x_i = n \right\}$, and the function $F\colon D^{(k)} \to \mathbb{R}$ defined as*

$$F(x) := \frac{k}{n} + \frac{1}{n^2} \sum_{i=1}^{k} x_i^2 \qquad \text{for } x \in D^{(k)} \ .$$

Then, F has a global minimum at x^* with $x_i^* = \left\lfloor \frac{n}{k} \right\rfloor$ for $i = 1, \ldots, k-r$ and $x_i^* = \left\lceil \frac{n}{k} \right\rceil$ for $i = k-r+1, \ldots, k$, where $0 \leq r < k$ and $r \equiv n \mod k$.

Proposition 2.2.1 is based on the fact, that, roughly speaking, evening out cluster sizes decreases F. Due to the special structure of simple cycles, we can swap neighboring clusters without changing the *modularity*. Thus, we can safely assume that clusters are sorted according to their sizes, starting with the smallest element. Then x^* is the only optimum. Evaluating F at x^* leads to a term that only depends on k and n. Hence, we can characterize the clusterings with maximum modularity only with respect to the number of clusters. The function to be minimized is given in Lemma 2.2.10:

$|\mathcal{C}|$ on cycles

Lemma 2.2.10 *Let C_n be a simple cycle with n nodes, $h\colon [1, \ldots, n] \to \mathbb{R}$ a function defined as*

$$h(x) := x \cdot n + n + \left\lfloor \frac{n}{x} \right\rfloor \left(2n - x \cdot \left(1 + \left\lfloor \frac{n}{x} \right\rfloor \right) \right) \ ,$$

and k^* be the argument of the global minimum of h. Then every clustering of C_n with maximum modularity has k^* clusters.

The proof of Lemma 2.2.10 builds upon Proposition 2.2.1, it can be found in the appendix. Finally we obtain the characterization for clusterings with maximum *modularity* for simple cycles.

optimal modularity on cycles

Theorem 2.2.7 *Let n be an integer and C_n a simple cycle with n nodes. Then every clustering $\mathcal{C}$ with maximum modularity has k cluster of almost equal size, where*

$$k \in \left[\frac{n}{\sqrt{n + \sqrt{n}}} - 1, \frac{1}{2} + \sqrt{\frac{1}{4} + n} \right] \ .$$

Furthermore, there are only 3 possible values for k for sufficiently large n.

The rather technical proof of Theorem 2.2.7 is based on the monotonicity of h.

2.2.7 Examples Revisited

Applying our results about maximizing *modularity* gained so far, we revisit three example networks that were used in related work [230, 177, 175]. More precisely, we compare published greedy solutions with respective optima, thus revealing two peculiarities of *modularity*. First, we illustrate a behavioral pattern of the greedy merge strategy and, second, we relativize the quality of the greedy approach.

Zachary's karate graph

The first instance is the karate club network of Zachary originally introduced in [230] and used for demonstration in [177]. The network models social interactions between members of a karate club. More precisely, friendship between the members is presented before the club split up due to an internal dispute. A representation of the network is given in Figure 2.2.5. The partition that has resulted from the split is given by the shape of the nodes, while the colors indicate the clustering calculated by the greedy algorithm and blocks refer to a optimum clustering maximizing *modularity*, that has been obtained by solving its associated ILP. The corresponding scores of *modularity* are 0.431 for the optimum clustering, 0.397 for the greedy clustering, and 0.383 for the clustering given by the split. Even though this is another example in which the greedy algorithm does not perform optimally, its score is

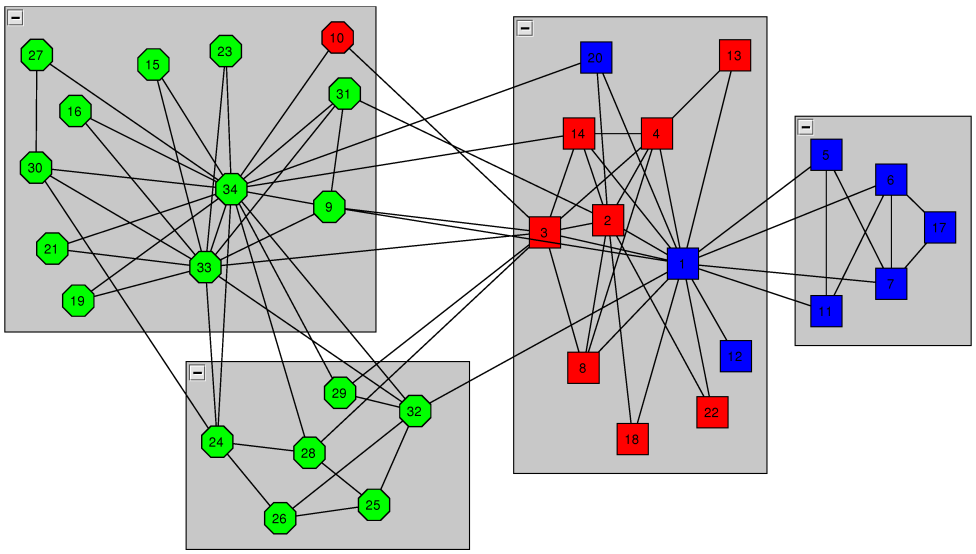


Figure 2.2.5. Karate club network of Zachary [230]. The different clusterings are coded as follows: blocks represent the optimum clustering (with respect to *modularity*), colors correspond to the greedy clustering, and shapes code the split that occurred in reality.

comparatively good. Furthermore, the example shows one of the potential pitfalls the greedy algorithm can encounter: Due to the attempt to balance the squared sum of degrees (over the clusters), a node with large degree (white square) and one with small degree (white circle) are merged at an early stage. However, using the same argument, such a cluster will unlikely be merged with another one. Thus, small clusters with skewed degree distributions occur.

Krebs' books on politics

The second instance is a network of books on politics, compiled by V. Krebs and used for demonstration in [175]. The nodes represent books on American politics bought from `Amazon.com` and edges join pairs of books that are frequently purchased together. A representation of the network is given in Figure 2.2.6. The optimum clustering maximizing *modularity* is given by the shapes of nodes, the colors of nodes indicate a clustering calculated by the greedy algorithm and the blocks show a clustering calculated by *Geometric MST Clustering* (GMC), which is introduced in [68], using the geometric mean of *coverage* and *performance*, see Section 2.1.3 for details on GMC. The corresponding scores of *modularity* are 0.527 for the optimum clustering, 0.502 for the greedy clustering, and 0.510 for the GMC clustering. Similar to the first example, the greedy algorithm is suboptimal, but relatively close to the optimum. Interestingly, GMC outperforms the greedy algorithm although it does not consider *modularity* in its calculations. This illustrates the fact that there probably are many *intuitive* clusterings close to the optimum clustering that all have relatively similar values of *modularity*. In analogy to the first example, we observe the same merge-artifact, namely the two nodes represented as dark-grey triangles.

GMC outperforms greedy

Lusseau's dolphins

As a last example, Figure 2.2.7 reflects the social structure of a family of bottlenose dolphins off the coast of New Zealand, observed by Lusseau et al. [157], who logged frequent associations between dolphins over a period of seven years. The clustering with optimum modularity (blocks) achieves a modularity score of 0.529 and, again, the greedy algorithm (colors) approaches this value with 0.496. However, structurally the two clusterings disagree on the two small clusters, whereas a clustering based on *iterative conductance cutting* [146]

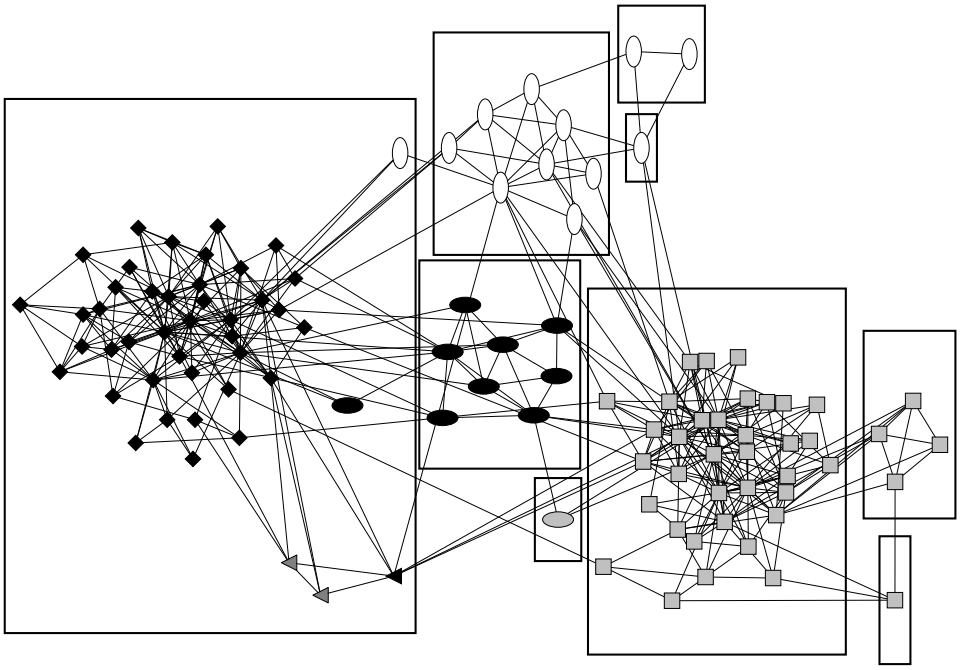


Figure 2.2.6. The networks of books on politics compiled by V. Krebs. The different clusterings are coded as follows: blocks represent the clustering calculated with GMC, colors correspond to the greedy clustering, and shapes code the optimum clustering (with respect to *modularity*).

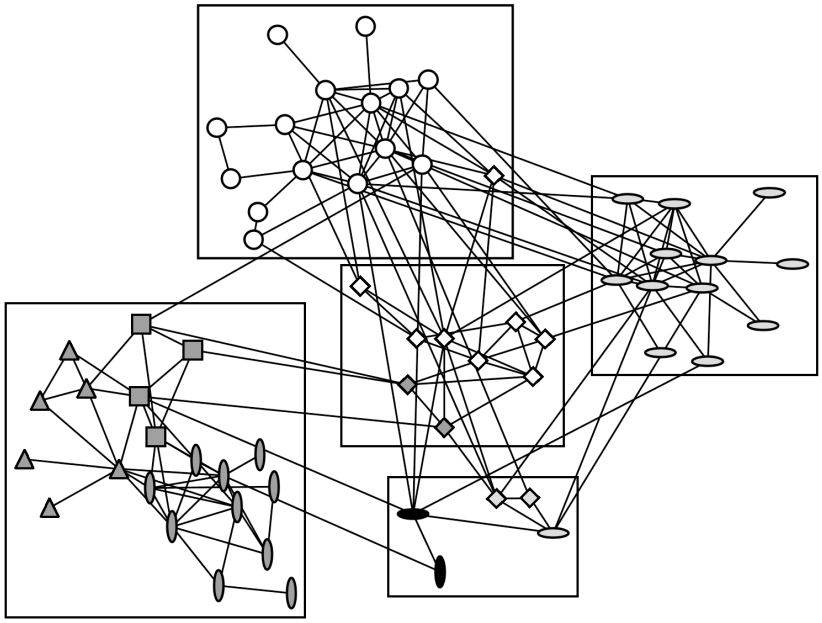


Figure 2.2.7. Social network of bottlenose dolphins introduced in [157] and clustered in [178]. The different clusterings are coded as follows: blocks represent the clustering with maximum *modularity*, colors represent the result of the greedy clustering, and shapes code the community structure identified with the *iterative conductance cut* algorithm presented in [146].

(shapes) achieves the same quality (0.492), but disagrees with the optimum only on the smallest cluster and on the refinement of the leftmost cluster.

Summarizing, the three examples illustrated several interesting facts. First of all, an artifical pattern in the optimization process of the greedy algorithm is revealed: The early merge of two nodes, one with a high and one with a low degree, results in a cluster which will not be merged with another one later on. In general, this can prevent finding the optimum clustering. Nevertheless, it performs relatively well on the given instances and is at most 10% off the optimum. However, applying other algorithms that do not optimize *modularity*, we observe that the obtained clusterings have similar scores. Thus, achieving good scores of *modularity* does not seem to be too hard on these instances. On the one hand, these clusterings roughly agree in terms of the overall structure, on the other hand, they differ in numbers of clusters and even feature artifacts such as small clusters of size one or two. Considering that all three examples exhibit significant community structure, we thus predict that there are many intuitive clusterings being structurally close (with respect to lattice structure) and that most suitable clustering algorithms probably identify one of them.

Appendix of Omitted Proofs

Proof. [of Corollary 2.2.5] Coverage of $\mathcal{C}$ can be expressed in terms of cluster sizes as follows:

$$\begin{aligned}
|E(\mathcal{C})| = \binom{n}{2} - \sum_{i=1}^{k} \prod_{j>i} |C_i| \cdot |C_j| \quad &= \binom{n}{2} - \frac{1}{2} \sum_{i=1}^{k} \prod_{j \neq i} |C_i| \cdot |C_j| \\
= \binom{n}{2} - \frac{1}{2} \sum_{i=1}^{k} |C_i| \cdot \sum_{j \neq i} |C_j| \ &= \binom{n}{2} - \frac{1}{2} \sum_{i=1}^{k} |C_i| \cdot (n - |C_i|) \\
= \binom{n}{2} - \frac{1}{2} \left(n^2 - \sum_{i=1}^{k} |C_i|^2 \right) &= -\frac{n}{2} + \frac{1}{2} \sum_{i=1}^{k} |C_i|^2 \ .
\end{aligned}$$

Thus, we obtain

$$\begin{aligned}
\textit{modularity} &= -\frac{1}{n-1} + \frac{1}{n(n-1)} \sum_{i=1}^{k} |C_i|^2 - \frac{1}{n^2} \sum_{i=1}^{k} |C_i|^2 \\
&= -\frac{1}{n-1} + \frac{1}{n^2 \cdot (n-1)} \sum_{i=1}^{k} |C_i|^2 \ ,
\end{aligned}$$

which proves the equation. □

Proof. [of Proposition 2.2.1] Since k and n are given, minimizing F is equivalent to minimizing $\sum_i x_i^2$. Thus let us rewrite this term:

$$\begin{aligned}
\sum_{i=1}^{k} \left(x_i - \frac{n}{k} \right)^2 &= \sum_{i=1}^{k} x_i^2 - 2\frac{n}{k} \sum_{i=1}^{k} x_i + k \cdot \left(\frac{n}{k} \right)^2 \\
&= \sum_{i=1}^{k} x_i^2 - 2\frac{n^2}{k} + \frac{n^2}{k} \\
\iff \qquad \sum_{i=1}^{k} x_i^2 &= \underbrace{\sum_{i=1}^{k} \left(x_i - \frac{n}{k} \right)^2}_{=:h(x)} + \frac{n^2}{k}
\end{aligned}$$

Thus minimizing F is equivalent to minimizing h. If r is 0, then $h(x^*) = 0$. For every other vector y the function h is strictly positive, since at least one summand is positive. Thus x^* is a global optimum.

Let $r > 0$. First, we show that every vector $x \in D^{(k)}$ that is close to $(\frac{n}{k}, \ldots, \frac{n}{k})$ has (in principle) the form of x^*. Let $x \in D \cap [\lfloor \frac{n}{k} \rfloor, \lceil \frac{n}{k} \rceil]^k$, then it is easy to verify that there are $k - r$ entries that have value $\lfloor \frac{n}{k} \rfloor$ and the remaining r entries have value $\lceil \frac{n}{k} \rceil$. Any 'shift of one unit' between two variables having the same value, increases the corresponding cost: Let $\epsilon := \lceil \frac{n}{k} \rceil - \frac{n}{k}$ and $x_i = x_j = \lceil \frac{n}{k} \rceil$. Replacing x_i with $\lfloor \frac{n}{k} \rfloor$ and x_j with $\lceil \frac{n}{k} \rceil + 1$, causes an increase of h by $5 + 2\epsilon > 0$. Similarly, in the case of $x_i = x_j = \lfloor \frac{n}{k} \rfloor$ and the reassignment $x_i = \lceil \frac{n}{k} \rceil$ and $x_j = \lfloor \frac{n}{k} \rfloor - 1$, causes an increase of h by $2 > 0$.

Finally, we show that any vector of $D^{(k)}$ can be reach from x^* by 'shifting one unit' between variables. Let $x \in D^{(k)}$ and with loss of generality, we assume that $x_i \leq x_{i+1}$ for all i. We define a sequence of elements in $D^{(k)}$ as follows:

1. $x^{(0)} := x^*$
2. if $x^{(i)} \neq x$, define $x^{(i+1)}$ as follows

$$x_j^{(i+1)} := \begin{cases} x_j^{(i)} - 1 & \text{if } j = \min\{\ell \mid x_\ell^{(i)} > x_\ell\} =: L \\ x_j^{(i)} + 1 & \text{if } j = \max\{\ell \mid x_\ell^{(i)} < x_\ell\} =: L' \\ x_j^{(i)} & \text{otherwise} \end{cases}$$

Note that all obtained vectors $x^{(i)}$ are elements of $D^{(k)}$ and meet the condition of $x_j^{(i)} \leq x_{j+1}^{(i)}$. Furthermore, we gain the following formula for the cost:

$$\sum_j \left(x_j^{(i+1)}\right)^2 = \sum_j \left(x_j^{(i)}\right)^2 + 2\left(x_{L'}^{(i)} - x_L^{(i)} + 1\right) \ .$$

Since $L < L'$, one obtains $x_{L'}^{(i)} \geq x_L^{(i)}$. Thus x^* is a global optimum in $D^{(k)}$. □

Proof. [of Lemma 2.2.10] Note, that $h(k) = F(x^*)$, where F is the function of Proposition 2.2.1 with the given k. Consider first the following equations:

$$\begin{aligned} \sum_{i=1}^{k} (x_i^*)^2 &= (k-r) \cdot \left\lfloor \frac{n}{k} \right\rfloor^2 + r \cdot \left\lceil \frac{n}{k} \right\rceil^2 \\ &= (k-r)\frac{(n-r)^2}{k^2} + r\left(\frac{(n-r)}{k} + 1\right)^2 \\ &= \frac{n-r}{k}\left((n-r) + 2r\right) + r = \frac{n^2 - r^2}{k} + r \\ &= \frac{1}{k}\left(n^2 - \left(n - \left\lfloor \frac{n}{k} \right\rfloor k\right)^2\right) + n - \left\lfloor \frac{n}{k} \right\rfloor k \\ &= 2n\left\lfloor \frac{n}{k} \right\rfloor - k\left\lfloor \frac{n}{k} \right\rfloor^2 + n - \left\lfloor \frac{n}{k} \right\rfloor k \\ &= n + \left\lfloor \frac{n}{k} \right\rfloor \left(2n - k\left(\left\lfloor \frac{n}{k} \right\rfloor + 1\right)\right) \end{aligned}$$

Since maximizing *modularity* is equivalent to minimize the expression $k/n + 1/n^2 \sum_i x_i^2$ for $(x_i) \in \bigcup_{j=1}^{n} D^{(j)}$. Note that every vector (x_i) can be realized as clustering with connected clusters. Since we have characterized the global minima for fixed k, it is sufficient to find the global minima by varying k. □

Proof. [of Theorem 2.2.7] First, we show that the function h can be bounded by the inequalities given in 2.2.11 and is monotonically increasing (decreasing) for certain choices of k.

$$kn + \frac{n^2}{k} \leq h(k) \leq kn + \frac{n^2}{k} + \frac{k}{4} \ . \tag{2.2.11}$$

In order to verify the Inequalities 2.2.11, let ϵ_k be defined as $n/k - \lfloor n/k \rfloor$ (≥ 0). Then the definition of h can be rewritten as follows:

$$\begin{aligned} h(k) &= kn + n + \left\lfloor \frac{n}{k} \right\rfloor \left(2n - \left(1 + \left\lfloor \frac{n}{k} \right\rfloor\right) k\right) \\ &= kn + n + \left(\frac{n}{k} - \epsilon_k\right) \left(2n - \left(1 + \frac{n}{k} - \epsilon_k\right) k\right) \\ &= kn + n + \frac{2n^2}{k} - (1-\epsilon_k)n - \frac{n^2}{k} - 2n\epsilon_k + (1-\epsilon_k)k\epsilon_k + n\epsilon_k \\ &= kn + \frac{n^2}{k} + (1-\epsilon_k)\epsilon_k k \ . \end{aligned}$$

Replacing the term $(1-\epsilon_k)\epsilon_k k$ by a lower (upper) bound of 0 ($k/4$) proves the given statements.

Second, the function h is monotonically increasing for $k \geq 1/2 + \sqrt{1/4+n}$ and monotonically decreasing for $k \leq n/\sqrt{n+\sqrt{n}} - 1$. In order to prove the first part, it is sufficient to show that $h(k) \leq h(k+1)$ for every suitable k.

$$\begin{aligned} h(k+1) - h(k) = (k+1)n + n + \left\lfloor \frac{n}{k+1} \right\rfloor \left(2n - \left(1 + \left\lfloor \frac{n}{k+1} \right\rfloor\right)(k+1)\right) \\ - kn - n - \left\lfloor \frac{n}{k} \right\rfloor \left(2n - \left(1 + \left\lfloor \frac{n}{k} \right\rfloor\right) k\right) \\ = n + 2n\left(\left\lfloor \frac{n}{k+1} \right\rfloor - \left\lfloor \frac{n}{k} \right\rfloor\right) - \left(1 + \left\lfloor \frac{n}{k+1} \right\rfloor\right) \left\lfloor \frac{n}{k+1} \right\rfloor \\ + k\left(\left(1 + \left\lfloor \frac{n}{k} \right\rfloor\right) \left\lfloor \frac{n}{k} \right\rfloor - \left(1 + \left\lfloor \frac{n}{k+1} \right\rfloor\right) \left\lfloor \frac{n}{k+1} \right\rfloor\right) \end{aligned}$$

Since $\lfloor \cdot \rfloor$ is discrete and $|\lfloor x \rfloor - \lfloor x-1 \rfloor| \leq 1$, one obtains:

$$h(k+1) - h(k) = \begin{cases} n - \left\lfloor \frac{n}{k} \right\rfloor^2 - \left\lfloor \frac{n}{k} \right\rfloor & \text{if } \left\lfloor \frac{n}{k} \right\rfloor = \left\lfloor \frac{n}{k-1} \right\rfloor \\ 3n - \left\lfloor \frac{n}{k} \right\rfloor^2 - \left\lfloor \frac{n}{k} \right\rfloor + 2k \left\lfloor \frac{n}{k} \right\rfloor & \text{otherwise} \end{cases} \tag{2.2.12}$$

Since $3n - \lfloor n/k \rfloor^2 - \lfloor n/k \rfloor + 2k \lfloor n/k \rfloor > n - \lfloor n/k \rfloor^2 - \lfloor n/k \rfloor$, it is sufficient to show that $n - \lfloor n/k \rfloor^2 - \lfloor n/k \rfloor \geq 0$. This inequality is fulfilled if $n - (n/k)^2 - n/k \geq 0$. Solving the quadratic equations leads to $k \geq 1/2 + \sqrt{1/4+n}$.

Using the above bound, for the second part, it is sufficient to show that

$$kn + \frac{n^2}{k} - (k+1)n - \frac{n^2}{k+1} - \frac{k+1}{4} \geq 0 \ , \tag{2.2.13}$$

since this implies that the upper bound of $h(k+1)$ is smaller than (the lower bound of) $h(k)$. One can rewrite the left side of Inequality 2.2.13 as:

$$kn + \frac{n^2}{k} - (k+1)n - \frac{n^2}{k+1} - \frac{k+1}{4} = -n + \frac{n^2}{k(k+1)} - \frac{k+1}{4} \ .$$

Since $h(k) - h(k+1)$ is monotonically decreasing for $0 \leq k \leq \sqrt{n}$, it is sufficient to show that $h(k) - h(k+1)$ is non-negative for the maximum value of k. We show that the lower bound $h_-(k) := -n + n^2/(k+1)^2 - (k+1)/4$ is non-negative.

$$h_-\left(\frac{n}{\sqrt{n+\sqrt{n}}} - 1\right) = -n - \frac{n}{4\sqrt{n+\sqrt{n}}} + \frac{n^2(n+\sqrt{n})}{n^2} = \sqrt{n} - \underbrace{\frac{n}{4\sqrt{n+\sqrt{n}}}}_{\leq \frac{1}{4}\sqrt{n}} \geq 0$$

Summarizing, the number of clusters k (of an optimum clustering) can only be contained in the given interval, since outside the function h is either monotonically increasing or decreasing. The length of the interval is less than

$$\frac{1}{2} + \underbrace{\sqrt{\frac{1}{4}+n} - \frac{n}{\sqrt{n+\sqrt{n}}}}_{=:\ell(n)} + 1 \ .$$

The function $\ell(n)$ can be rewritten as follows:

$$\ell(n) = \frac{\sqrt{\left(\frac{1}{4}+n\right)\left(\sqrt{n+\sqrt{n}}\right)} - n}{\sqrt{n+\sqrt{n}}} \leq \frac{\left(n + \frac{1+\epsilon}{2}\sqrt{n}\right) - n}{\sqrt{n+\sqrt{n}}} \leq \frac{1+\epsilon}{2}\sqrt{\frac{n}{n+\sqrt{n}}} \ , \qquad (2.2.14)$$

for every positive ϵ. Inequality 2.2.14 is due to the fact that

$$\begin{aligned}\left(\frac{1}{4}+n\right)\left(\sqrt{n+\sqrt{n}}\right) &\leq n^2 + n\sqrt{n} + \frac{1}{4}\left(n+\sqrt{n}\right) \qquad \leq n^2 + 2\frac{1+\epsilon}{2}n\sqrt{n} + \frac{(1+\epsilon)^2}{4}n \\ &= \left(n + \frac{1+\epsilon}{2}\sqrt{n}\right)^2 \ ,\end{aligned}$$

for sufficiently large n. □

Section 2.3

Lucidity-Driven Graph Clustering

When I consider what people generally want in calculating, I found that it always is a number. I also observed that every number is composed of units, and that any number may be divided into units. Moreover, I found that every number which may be expressed from one to ten, surpasses the preceding by one unit: afterwards the ten is doubled or tripled just as before the units were: thus arise twenty, thirty, etc. until a hundred: then the hundred is doubled and tripled in the same manner as the units and the tens, up to a thousand;
. . . so forth to the utmost limit of numeration.

(Abū Ja'far Muḥammad ibn Mūsā Al-Khwārizmī, ca. 820, eponym of the term "algorithm")

BEING AWARE THAT optimality and even reasonable approximability are out of reach, when trying to find a graph clustering with good *modularity* (see Section 2.2), is certainly crucial and justifies the use of heuristics for this purpose. However, if we take a more practical view, other questions are more pressing. Why should *modularity* be based on *coverage*, an index which is almost infamous for its simplicity and its downside (see Section 1.2.2)? What happens if we plug in a different index, and shouldn't we normalize by division instead of subtraction? Are the—rather sloppily stated—probabilistic assumptions *modularity* is based upon supported by an actual probability space? Finally and most importantly, does the behavior of *modularity* agree with human intuition and with that of other established quality indices for clusterings? Can a heuristic maximization of it compete with established algorithms?

In this section we answer these questions to a large extent. We formally state and investigate the founding paradigm for *modularity*, which we coin the *lucidity*[8] of a clustering, as the trade-off between the achieved quality and the expected quality for random networks incorporating the intrinsic properties of the original network. Furthermore we explore a probability space for random networks that fulfills the assumptions underlying *modularity*. Using this space, exchanging *coverage* by the more meaningful index *performance* as the base measure leads to an equivalent *lucidity*-index—a fact which corroborates the feasibility of *modularity*. As a byproduct of the derivation thereof, the ILP formulation (see Sections 2.2.2, 2.4) leads

[8]Being synonymous to *clarity, distinctness*, the term *lucidity*, stemming from the Latin root *lux* (light), *lucidus* (bright, clear), was chosen for this measure which, states how "clearly" a clustering is represented by a graph's structure.

to the NP-hardness of the problem MinMixedMultiPartition, which is a generalization of the cut-type outlook onto *modularity*.

As a prequel to our experimental evaluation, we show how a geometric interpretation allows us to harness the capabilities of a data structure for fully dynamic convex hulls in order to greedily maximize *lucidity* with a divisive normalization in a way similar to that discussed in Section 2.2.5 and in $O(n^2 \log n)$ time. Perhaps the part of most practical relevance is the systematic experimental evaluation of three realizations of the *lucidity* paradigm, including *modularity*. Our results confirm that *lucidity* (and *modularity*) does behave in strong agreement with human intuition and with other indices, thus supporting its usage. With results that support the feasibility of *lucidity* as a quality index we then systematically let greedy maximization algorithms find clusterings. We compare the goodness of these algorithms in terms of clustering quality to that of other clustering algorithms on a set of random pre-clustered graphs and complement our findings with results on real data. Our results indicate the feasibility of the paradigm in that, on the whole, the proposed algorithms surpass the benchmark algorithms, and in that the generality of the approach is justified by specific realizations of *lucidity* excelling on diverse tasks and on real-world data. In particular, we suggest $\mathrm{L}^{\div}_{\mathrm{perf}}$ as a strong community detection algorithm if a low or constant number of clusters is to be expected, L^{-}_{*} (*modularity*) as a good *all-round* measure and reject $\mathrm{L}^{\div}_{\mathrm{cov}}$ as it is too sensitive to graph density.

Summarizing, this section together with the preceeding one clarifies much of which has never been known or clearly stated about *modularity* and unfurls a sound theoretical and probabilistic background and a founding paradigm for this quality index, which has already spread into many fields of science. On the whole, my work on *modularity* did not really lead to evidence that this index is inferior, a prevalent opinion among quite a few colleagues of mine when I started my work. A few parts of this section have been published in [101] in less detail, based on joint work with Marco Gaertler and Dorothea Wagner. However, at that time our idea was to find a foundation of *modularity* which truly coincides with that of *statistical significance*. Despite the fact that we soon realized that this was not possible without throwing away most of the original ideas, the name *significance* stuck and manifested in the name of that publication. It took a while to actually do away with it. At the same time as this thesis, most of this section will be published in [115], based on joint work with Marco Gaertler, Florian Hübner and Dorothea Wagner.

Main Results

- We state the founding clustering paradigm of *modularity*, *lucidity* $L^{\odot}_{\mathcal{M}}(\mathcal{C})$, which considers the trade-off between achieved quality and expected quality. (Section 2.3.2)
- There is a discrete probability space (Ω, p) that fully supports *modularity* in its original spirit, and that yields a closed expression for the probabilities of all graphs in Ω. (Section 2.3.2.1, Equation 2.3.6 and Lemma 2.3.2)
- (Ω, p) must allow loops and parallel edges, otherwise some assumptions about *modularity* need to be dropped. (Section 2.3.2.1)
- We state sufficient conditions for a probability space to support *modularity*. For a weighted version of *modularity* we give a random process which yields a space that fulfills these sufficient conditions. (Lemma 2.3.3, Algorithm 2 and Lemma 2.3.4)
- By solely dropping the postulation for expected edge degrees we can state a probability space for loop-free graphs, confer most previous results and state a loop-free *modularity*. (Section 2.3.2.1)
- We derive four implementations of the *lucidity*-paradigm $\mathrm{L}^{-}_{\mathrm{cov}}$, $\mathrm{L}^{\div}_{\mathrm{cov}}$, $\mathrm{L}^{-}_{\mathrm{perf}}$ and $\mathrm{L}^{\div}_{\mathrm{perf}}$using *coverage*, *performance*, *subtraction* and *division*. (Section 2.3.2.2)

- $L^{-}_{cov}$is equivalent to L^{-}_{perf}, i.e., substituting *coverage* by *performance* in the concept of *modularity* yields an equivalent index. (Section 2.3.2.3 and Lemma 2.3.6)
- It is NP-complete to find an L^{-}_{perf}-optimal clustering. (Corollary 2.3.3)
- By the cut-view onto *modularity*, the problem MinMixedMultiPartition is NP-hard, and its restrictions to smaller partitions need not be coarsenings of an optimal MinMixedMultiPartition. (Section 2.3.2.4 and Corollary 2.3.4)
- There are algorithms and data structures (using geometry and a dynamic convex hull) that support the greedy maximization of the proposed implementations of *lucidity* in $O(n^2 \log n)$. (Section 2.3.3, Lemmata 2.3.7, 2.3.8, Algorithms 3, 4)
- A systematic experimental evaluation on generated graphs yields that *lucidity* agrees with human intuition, the *ground truth* of a graph generator and other established quality indices. (Section 2.3.4.2)
- Greedily maximizing *lucidity* competes well with established clustering algorithms in terms of *coverage*, *performance* and *inter-cluster conductance* (and *lucidity* itself), which is corroborated on real-world networks. (Sections 2.3.4, 2.3.4.3)

Future Work. An old question about *modularity* has recently resurfaced, as Ulrik Brandes asked me if it were possible to *really* transfer the concept of *statistical significance* to clustering. I still agree that this is tempting and should be addressed. Apart from the question about the computational complexity of $L^{\div}_{perf}$, a pressing issue is whether *modularity* is *fixed parameter tractable*. While Theorem 2.2.2 (NP-hardness of 2-Modularity) shatters the hope that $|\mathcal{C}|$ could serve as a good parameter for such an approach, there might be a different parameter such as $\deg_{max}$, or, ultimately, another hardness result.

2.3.1 Preliminaries

Although considering only simple graphs suffices for most insights, we require loops and parallel edges later and thus start out general straight away. We often consider only unweighted graphs but will say so explicitly. Recall from Section 1.2.1 that since we now allow non-simple graphs, we write both edges and edge sets E as multisets, such that $\{v, v\} \in E$ is allowed (a loop) and $E = \{\{u, v\}, \{u, v\}\}$ (two parallel edges). Recall further our convenient notations (u, v), $V^{\times}$, V^2, $\omega(\{u, v\})$, $\omega(e)$ and $\omega(v)$, as well as d_i (ω_i) for the degree (weight) of cluster C_i.

non-simple graphs

2.3.2 The Lucidity Paradigm

In the *lucidity paradigm* a good clustering is characterized by having a high quality compared to the value the clustering obtains for a random network that reflects specific *structural properties* that are expected to be present in the graph, as predefined in an appropriate null hypothesis. The structural properties of a graph can include characteristics such as the sequence of degrees, the number of nodes, the clustering coefficient, the degree distribution etc. These properties do not determine a graph completely but define a family of graphs incorporating them. A configuration then is a specific instantiation of these properties, i.e., a specific graph. Every realization of the *lucidity* paradigm requires a quality measure, a null hypothesis based on a set of such characteristics, and a mode of comparison of these.

lucidity paradigm

quality vs. expected quality

The concept of *lucidity* is related to the notion of p-values in statistical hypothesis testing. The p-value of a value t observed for a random variable T is the probability that under the assumption of a given null hypothesis, T assumes a value at least as unfavorable to the null hypothesis as the observed value t. In general, the null hypothesis is rejected, if the p-value is smaller than the statistical significance level. However, in our concept we do not reject a null

relation to significance

hypothesis, which we assume to reasonably describe observed graphs. Instead, we compare the achieved quality of a clustering to the expected value, in order to judge its relevance.

lucidity $L^{\odot}_{\mathcal{M}}$

Definition 2.2 *Given a quality index $\mathcal{M}$ and a clustering $\mathcal{C}$, we define the lucidity $L^{\odot}_{\mathcal{M}}$ of a clustering $\mathcal{C}$ as the corresponding quality index respecting our paradigm in the following way:*

$$L^{\odot}_{\mathcal{M}}(\mathcal{C}) := \mathcal{M}(\mathcal{C}) \odot \mathbb{E}_{\Omega}[\mathcal{M}(\mathcal{C})] \ , \tag{2.3.1}$$

where $\mathbb{E}_{\Omega}[\mathcal{M}]$ is the expected value of the quality index $\mathcal{M}$ for the clustering $\mathcal{C}$ with respect to a suitable probability space Ω and $\odot$ is a binary operator on real numbers.

mod $= L^{-}_{\text{cov}}$

The key intuition of the *lucidity paradigm* is that a clustering is *lucid*, if the edges support a good community structure that is unlikely to emerge if links were inserted at random but respecting intrinsic properties of the graph. As, in this paradigm, *modularity* (Equation 2.3.2) employs *coverage* (Equation 1.2.1) and *subtraction*, the concept of *lucidity* is a true generalization of *modularity*. For convenience we repeat the definition of *modularity* in the two formulations which we will be using in this section.

$$\text{mod}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} - \frac{1}{4m^2} \sum_{C \in \mathcal{C}} \left(\sum_{v \in C} \deg(v) \right)^2 \ , \tag{2.3.2}$$

formulas for modularity

or alternatively and equivalently:

$$\text{mod}(\mathcal{C}) = \sum_{\{u,v\} \in V^{\times}} \left(\frac{A(u,v)}{m} \delta_{uv} \right) - \sum_{(u,v) \in V^2} \left(\frac{\deg(u) \cdot \deg(v)}{4m^2} \delta_{uv} \right) \ , \tag{2.3.3}$$

$$\text{with } \delta_{uv} = \begin{cases} 1 & \text{if } \mathcal{C}(u) = \mathcal{C}(v) \\ 0 & \text{otherwise} \end{cases} \ ,$$

and $A(u,v) =$ number of (parallel) edges between u and v.

2.3.2.1 A Probabilistic Setup

probabilities for modularity

The question that motivates this subsection is: Is there a sound probability space underlying the definition of *modularity*? The random models proposed below are thus not intended to be particularly elegant or universal, but they serve as a support for *modularity* and *lucidity*.

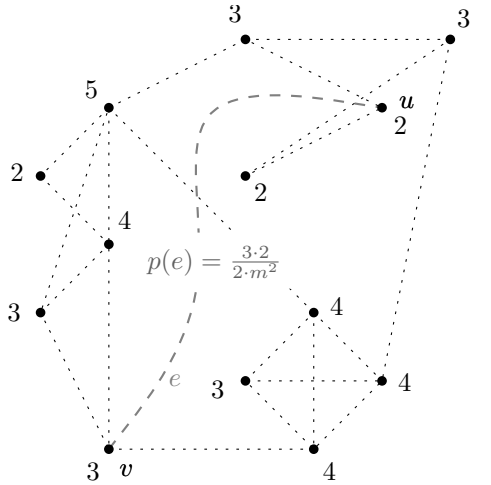


Figure 2.3.1. Input graph with original node degrees inducing probabilities.

In the following we discuss a suitable probability space (Ω, p) required for Definition 2.2, which we use throughout this paper. We restrict ourselves to the unweighted case for now and discuss a weighted setup later. In their definition of *modularity*, the authors of [178] and [57] suggest setting the probability of a randomly inserted edge to become $\{u,v\}$ $(u \neq v)$ to $\deg(v)\deg(w)/2m^2$. The motivation for this, and thus the assumed underlying principle by which the graph is built, is a random process that inserts m edges into the disconnected set of n nodes, of which both ends then connect to node x with probability proportional to the degree $\deg(x)$ of x in the input, i.e., $\deg(x)/2m$. However, the model also assigns a probability of $(\deg(v))^2/4m^2$, to a loop on v, a fact rarely mentioned explicitly. Thus we obtain

$$p(e) := \begin{cases} \frac{\deg u \deg v}{2m} & \text{if } e = \{u,v\}, u \neq v \\ \frac{(\deg v)^2}{4m} & \text{if } e = \{v,v\} \end{cases} \ . \tag{2.3.4}$$

$p(e)$ normed to 1

As follows, this setup is unbiased, i.e., probability masses of edges add up to 1:

$$\sum_{\{u,v\}\in V^\times} p[e=\{u,v\}] = \underbrace{\sum_{v>w\in V} \frac{\deg v \deg w}{2m^2}}_{\text{non-loops}} + \underbrace{\sum_{v\in V} \frac{(\deg v)^2}{4m^2}}_{\text{loops}} \tag{2.3.5}$$

$$= \sum_{v,w\in V} \frac{\deg v \deg w}{4m^2} = \frac{1}{4m^2}\left(\sum_{v\in V} \deg v\right)^2 = \frac{1}{4m^2}(2m)^2 = 1$$

In the case that edges are not allowed to form loops (a question the literature does not agree on, even for *simple* graphs) the above assumptions are incorrect and overestimate the number of intra-cluster edges, since the intra-cluster edge mass contributed by loops has to be distributed elsewhere. We discuss such a setting later in this section. However, we are not aware of a manageable solution if parallel edges were disallowed.

incorrect for simple graphs

Thus, the discrete probability space (Ω_E, p) for edge insertions uses as Ω_E all unordered pairs (two-element multisets) $\{u,v\} \in V^\times$, and $p(\{u,v\})$ is defined as above. Clearly, the probability function p is nonnegative, and the sample space Ω_E is normed to 1 by Equation 2.3.7. A trial consisting of m edges being drawn independently as elementary events from Ω_E, by symmetry and using the above probabilities, yields an expected number[9] of $(\deg(u)\deg(v))/(2m)$ (parallel) edges between u and v, for two nodes $u \neq v$, and an expected number $(\deg(v))^2/(4m)$ of self loops on v. These very values are used in the definition of *modularity* in Equation 2.3.3. We shall discuss a resulting probability space for graphs below.

(Ω_E, p)

By fixing the number m of edges and expected node degrees, the above setup is rather restrictive. In the *lucidity* paradigm, different random models are conceivable, if other or less properties of a graph are considered to be fixed according to the application. An example would be not to fix the number m of edges to be inserted in the probabilistic model, but to allow any (possible) number of non-parallel edges instead. Then, one could still use edge probabilities proportional to those used in Equation 2.3.5 in order to obtain probabilities for graphs. However, this will not yield the expected coverage as stated in the formula of *modularity* (a minimalist counterexample is easy to find). Note that this does not disprove the existence of a different setup which yields the formula of *modularity*. However, given the ideas of the founders of *modularity* and the fact that a sound probability space for graphs in accordance with its formula can be given (see below), we restrict ourselves to that setup in this work.

other setups

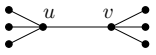
Since an edge set E' is a multiset of elementary events in Ω_E, we may build upon this setup and define the discrete probability space (Ω, p) for graphs as follows. Let Ω consist of all m-element multisets of elementary events in (Ω_E, p), which is a subset of the set of all multisets over Ω_E. We can now trivially identify the family of all graphs on n (labeled) nodes and m (unlabeled) edges with Ω. The probability for a specific graph $H = (V, E')$ in this family can then be chosen to directly reflect the edge probabilities (see Equation 2.3.4) in the definition of *modularity*: Using edge probabilities $p(e)$ as defined in (Ω_E, p) (see Equation 2.3.4), in space (Ω, p), let

(Ω, p)

$$p(H) := \underbrace{\prod_{e\in E'} p(e)}_{\substack{\text{prob. of } \textit{one} \text{ ordering} \\ \text{of the events in } E'}} \cdot \underbrace{\frac{m!}{\prod_{e\in E'} s_e!}}_{\substack{\text{number of orderings} \\ \text{which yield } E'}} \qquad (s_e = \text{multipl. of } e \text{ in } E') \tag{2.3.6}$$

be the probability of the event that the m elementary events from Ω_E result in the multiset E' and thus induce H. Particular attention has to be paid to the multiplicity s_i of elementary events (i.e., graphs) of Ω occurring in the set of all series of m elementary events of Ω_E. For

[9] This quantity was claimed to be the *probability* of an edge existing between u and v in [57], however, multiplication by m clearly yields the *expected* number of edges. The righthand graph yields $8/7 > 1$ for edge $\{u,v\}$ for this number.



our graph H it does not make a difference in which order this multiset is drawn, thus, several series can lead to the same graph (i.e., elementary event in Ω).[10]

a suitable random process

From the above construction of (Ω, p) a random process for graph creation is immediate: draw m edges independently, each according to (Ω_E, p). Equations 2.3.4 and 2.3.5 yield that this model is unbiased, yielding m expected edges. Again, since edges are drawn independently, it is easy to see that this probability space is sound, i.e. that $p(H) \geq 0$ and that $\sum_{H\in\Omega} p(H) = 1$. The former claim is trivial by Equations 2.3.6 and 2.3.4, and the latter can be seen as follows. As opposed to the above, suppose for now the drawings to be *labeled*, i.e., it matters in which order edges are drawn, and let this setup be $(\dot{\Omega}, p)$. Then we obtain $|V^\times|^m = \tilde{m}^m$ different elementary events $\dot{\delta}$ in $\dot{\Omega}$ (some of which represent identical graphs, merely with edges added in a different order). Analogous to (Ω, p) (Equation 2.3.6), we may now define $p(\dot{\delta}) = \prod_{e\in\dot{\delta}} p(e)$ for all $\dot{\delta} \in \dot{\Omega}$, and get the following lemma:

$\tilde{m} = |V^\times|$

$(\dot{\Omega}, p)$ and (Ω, p) are normed to 1

Lemma 2.3.1 *The probability spaces $(\dot{\Omega}, p)$ and (Ω, p) are normed to 1.*

Proof.

$$\sum_{\dot{\delta}\in\dot{\Omega}} \prod_{e\in\dot{\delta}} p(e) = \sum_{\substack{E'\in \\ (V^\times)^m}} \prod_{e\in E'} p(e) = \Big(\sum_{e\in V^\times} p(e)\Big)^m = 1^m = 1 \ . \tag{2.3.7}$$

The first two equalities exploit the independence of $p(e)$ and reorder terms, and the third equality holds by Equation 2.3.5. Given that $(\dot{\Omega}, p)$ is normed to 1, for (Ω, p) we only have to summarize terms that represent the same unordered multiset (graph) as shown in Equation 2.3.6 and obtain that (Ω, p) is normed to 1. □

What is left to show is that for any given graph G and clustering $\mathcal{C}(G)$, $\mathbb{E}(\mathrm{cov}(\mathcal{C}))$ in (Ω, p) equals the term in *modularity* (see Equation 2.3.3):

$(\dot{\Omega}, p)$ yields modularity

Lemma 2.3.2 *For any given graph G and clustering $\mathcal{C}(G)$, in (Ω, p) it holds that: $\mathbb{E}(\mathrm{cov}(\mathcal{C})) = \sum_{(u,v)\in V^2} \frac{\deg(u)\deg(v)}{4m^2}\delta_{uv}$. (As above s_e denotes e's multiplicity.)*

Proof.

$$\begin{aligned}
\mathbb{E}(\mathrm{cov}(\mathcal{C})) &= \mathbb{E}\left(\frac{\sum_{e\in E(\mathcal{C})} s_e}{m}\right) = \frac{1}{m}\mathbb{E}\left(\sum_{e\in E(\mathcal{C})} s_e\right) = \frac{1}{m}\sum_{e\in E(\mathcal{C})} \mathbb{E}(s_e) \\
&= \frac{1}{m}\left(\sum_{\substack{e=\{u,v\}\in E(\mathcal{C}) \\ u\neq v}} \frac{\deg(u)\deg(v)}{2m} + \sum_{\substack{e\in E(\mathcal{C}) \\ e=\{v,v\}}} \frac{(\deg(v))^2}{4m}\right) \\
&= \sum_{(u,v)\in V^2} \frac{\deg(u)\deg(v)}{4m^2}\delta_{uv}
\end{aligned} \tag{2.3.8}$$

□

suff. cond. for such a space

Examining the above proof we can see that any distribution that (i) fulfills Equation 2.3.4 and (ii) surely uses a total number m of edges, has the property described in Lemma 2.3.2. Moreover we can immediately see that the additional postulation that expected node degrees should be fixed is also fulfilled.

$\mathbb{E}(\deg(v)) = \deg(v)$

Corollary 2.3.1 *The expected edge degree of node v in (Ω, p) is $\deg(v)$ (from G).*

[10] This multiplicity is accounted for by the second factor in Equation 2.3.6. This factor can be seen as follows: there are $m!$ possibilities to order m events, but since the s_i drawings of event i are indistinguishable, $s_i!$ of these $m!$ orderings are identical; as this applies to the multiplicities of all events, we obtain the given factor. It equals $m!$ iff $s_e = 1$ for all $e \in E'$, and 1 iff $s_e = m$ for some $e \in E'$.

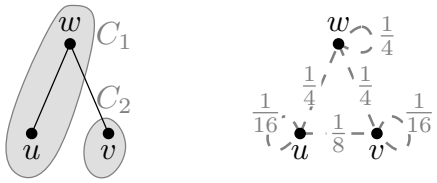
Proof.

$$\mathbb{E}_\Omega(\deg(v)) = \sum_{\substack{u \in V \\ u \neq v}} \frac{\deg(u)\deg(v)}{2m} \cdot 1 + \frac{(\deg(v))^2}{4m} \cdot 2$$
$$= \deg(v)\left(\frac{2m - \deg(v)}{2m} + \frac{deg(v)}{2m}\right) = \deg(v)$$

□

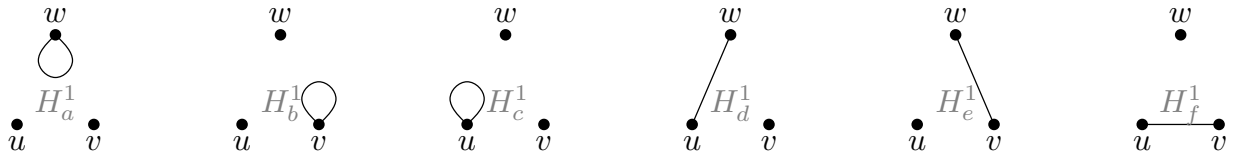
The above proof uses the discussed edge probabilities; note that a self loop (second summand) contributes 2 to the $\deg(v)$. Concluding, we now have a sound probabilistic setup for unweighted graphs for the *lucidity* paradigm.

An Instructive Example. The following tiny example illustrates this model. Let graph $G = (V, E)$ in the righthand Figure 2.3.2a be given, with $n = 3, m = 2$, alongside a clustering $\mathcal{C}$. Figure 2.3.2b states the edge probabilities according to Equation 2.3.4, comprising $\tilde{m} = \binom{n}{2} + n = 6$ possible edges. To simplify things we first consider only one random edge: The family $\mathcal{H}_1$ of the 6 graphs on three nodes and only one edge are easily listed, their probabilities match the corresponding edge probabilities.



(a) $G, \mathcal{C}(G)$ **(b)** $p(e)$

Figure 2.3.2. Given G (a), Equation 2.3.4 yields probabilities $p(e)$ (b)



(a) $p_a = \frac{1}{4}$ **(b)** $p_b = \frac{1}{16}$ **(c)** $p_c = \frac{1}{16}$ **(d)** $p_d = \frac{1}{4}$ **(e)** $p_e = \frac{1}{4}$ **(f)** $p_f = \frac{1}{8}$

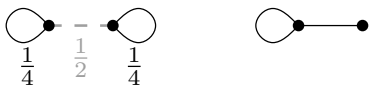
Figure 2.3.3. Family $\mathcal{H}_1$ of graphs induced by (Ω_1, p); only one random edge is inserted in Ω_1

Due to the independence of edge drawings, we can now build the required probability space (Ω_2, p) inducing the family $\mathcal{H}_2$ of graphs on 3 nodes and 2 edges by building the Cartesian product $\Omega_1 \times \Omega_1$. This yields 6^2 outcomes in Ω_2, whose probabilities are obtained by multiplying those of the participating members of Ω_1. Of these outcomes $\tilde{m}$ occur once (two parallel edges) and $\binom{\tilde{m}}{2}$ occur twice (different insertion orders lead to the same graph). Figure 2.3.4 shows two of the 21 possible graphs in $\mathcal{H}_2$. The probability in 2.3.4a is double the product of the edges, due to two possible insertion orders.

(Ω_2, p)
$= \Omega_1 \times \Omega_1$

(a) $p(H_a^2) = \frac{1}{8}$ **(b)** $p(H_c^2) = \frac{1}{16}$

Figure 2.3.4. Two examples graphs from Ω_2 and their probabilities as consistent with the formula of *modularity*.



(a) G with $p(e)$ **(b)** $p(H) = \max$

Figure 2.3.5. A graph G and one of its most likely random variants H. Outcome G has lower probability.

Consider now the clustering $\mathcal{C}$ of G depicted in Figure 2.3.2a. Equation 2.3.2 yields $\mathrm{mod}(\mathcal{C}) = \frac{1}{2} - \frac{3^2+1^2}{4 \cdot 2^2} = -\frac{1}{8}$, and in particular $\mathbb{E}(\mathrm{cov}) = \frac{5}{8}$. To see that this coincides with the expected coverage in Ω_2 (i.e., $\mathcal{H}_2$) regarding $\mathcal{C}$, as theoretically proven in Lemma 2.3.2, we can list all 21 different members of $\mathcal{H}_2$ and check that $\sum_{H \in \mathcal{H}_2} p(H)\mathrm{cov}(\mathcal{C})_H = \frac{5}{8}$ (see

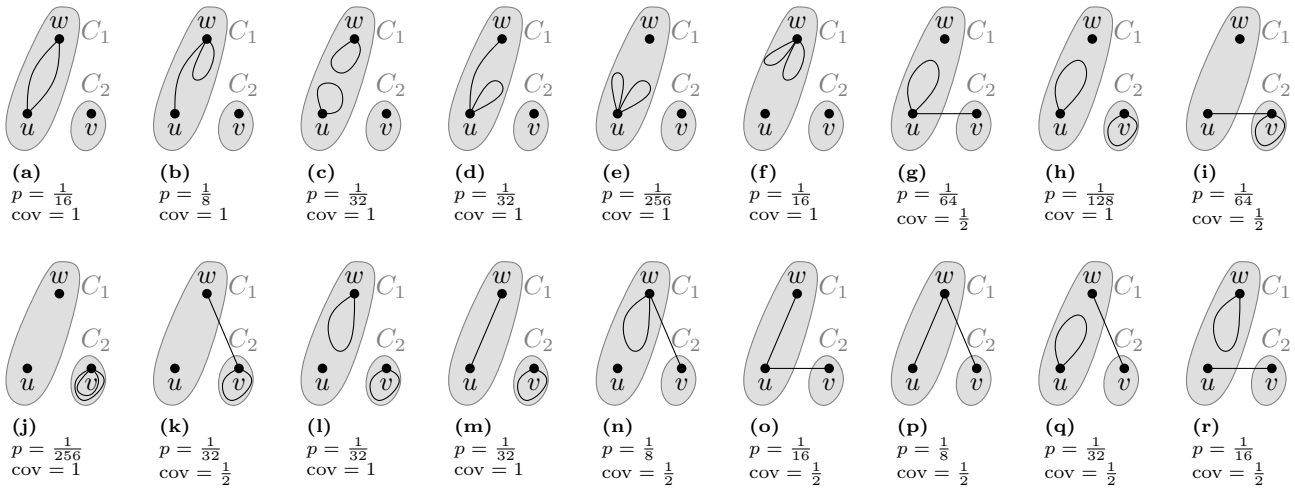


Figure 2.3.6. All graphs in $\mathcal{H}^2$ with positive *coverage* for $\mathcal{C}$, yielding $\mathbb{E}(\text{cov}(\mathcal{C})) = \frac{5}{8}$. Note that graphs with non-parallel edges occur twice in Ω_2, hence their double probability.

$p(G) \neq \max$

Figure 2.3.6 for completeness). As an interesting side note, the example in Figure 2.3.5 shows that this setup does not necessarily grant the highest probability to the very graph used as the blueprint for the probability space. In Figure 2.3.5 $p(H) = \frac{1}{4}$ and $p(G) = \frac{1}{8}$.

weighted edges

The Weighted Case. A generalization of *modularity* to weighted edges, such that its restriction to weights 0 and 1 yields the unweighted version, is straightforward, as proposed in [172]. We again state the formula we use, in order to disambiguate between formulations in previous works:

$$\text{mod}_\omega(\mathcal{C}) := \underbrace{\frac{\omega(\mathcal{C})}{W}}_{\text{cov}_\omega} - \underbrace{\frac{1}{4W^2}\sum_{C\in\mathcal{C}}\left(\sum_{v\in C}\omega(v)\right)^2}_{\mathbb{E}(\text{cov}_\omega)} \tag{2.3.9}$$

$\mathbb{E}(\omega(e))$

Analogous to unweighted edges, this formula assumes for expected edge weights

$$\mathbb{E}(\omega(e)) := \begin{cases} \frac{\omega(u)\omega(v)}{2W} & \text{if } e = \{u,v\}, u \neq v \\ \frac{(\omega(v))^2}{4W} & \text{if } e = \{v,v\} \end{cases}. \tag{2.3.10}$$

Note that for our view (but not necessarily in the application's view) parallel edges are obsolete (even disruptive, notationally) in this setting if we allow the edge weight function ω to go beyond 1 as $\omega : E \to \mathbb{R}_0^+$ and simply summarize parallel edges to one "heavier" edge. For simplicity we shall do this in the following.

$\mathbb{E}(W) = W$

Analogous to Equation 2.3.5 we can see that the choices in Equation 2.3.10 are unbiased, as the expected total edge mass $\mathbb{E}(W)$ equals W. However, we have left the field of discrete probabilities, and describing a continuous probability space for this setup is not as easy, as we cannot simply draw m edges independently but have to continuously distribute an edge mass W. Analogous to Lemma 2.3.2 we can prove the following lemma

suff. cond. for a space

Lemma 2.3.3 *A probability distribution for weighted graphs will justify Equation 2.3.9 if it fulfills the following two properties:*[11]

(i) expected edge weights are as in Equation 2.3.10,

(ii) random graphs surely have a total edge mass equal to W.

[11] As in the unweighted case this does not rule out the existence of different setups.

We will not define a probability distribution for weighted graphs, but describe a rather simple random process, which produces a distribution which fulfills these two properties. The general idea of this (arguably realistic) process is that each edge starts exactly with its expected weight (as in Equation 2.3.10). Then in an arbitrary number of handshakes between random edges, two participants contest about their combined edge mass. The mass is divided up in a new random way between the two, but such that the expected ratio of the two halves matches the ratio of their respective expected weights. Suppose the two handshaking edges are e_ℓ and e_r with expected weights a_ℓ and a_r, and actual edge weights x_ℓ and x_r, respectively. Let $c := a_\ell/(a_\ell + a_r)$ be the fraction that e_ℓ expects to get. We define a piecewise uniform density function $d(x)$ as depicted in Figure 2.3.7 as follows:[12]

no space but a random process

Figure 2.3.7. density d

$$d(x) = \begin{cases} \frac{1-c}{c} =: f_\ell & \text{if } 0 \le x \le c \\ \frac{c}{1-c} =: f_r & \text{if } c < x \le 1 \end{cases} . \tag{2.3.11}$$

Having drawn x from $d(x)$, the available weight $x_\ell + x_r$ is divided up such that e_ℓ gets a part of size $x \cdot (x_\ell + x_r)$ and e_r gets a part of size $(1-x) \cdot (x_\ell + x_r)$. Algorithm 2 summarizes this procedure.

Algorithm 2: Random Process for Weighted Graphs

1 Set a_i and x_i as in Eq. 2.3.10 $\forall e_i = \{u, v\} \in V \times V$
2 **for** $\#T$ *runs* **do**
3 unif. at rand. choose edges $\{\ell, r\} \in \binom{V}{2}$ `// choose contestants`
4 $c \leftarrow \frac{a_\ell}{a_\ell + a_r}$ `// ℓ's expected fraction`
5 draw[12] $x \sim d(x)$ as in Equation 2.3.11 `// see Figure 2.3.7 for` $d(x)$
6 $x_\ell \leftarrow x(x_\ell + x_r)$ and $x_r \leftarrow (1-x)(x_\ell + x_r)$ `// distribute` $x_\ell + x_r$
7 **return** Graph G with edge weights x_i

random process yields modularity

Lemma 2.3.4 *Given a weighted graph G and a clustering $\mathcal{C}(G)$. Algorithm 2 yields a distribution of graphs with $\mathbb{E}(\mathrm{cov}_\omega)$ as used in Equation 2.3.9.*

Proof. We use Lemma 2.3.3 for the proof. Property (ii) is trivially fulfilled as edge mass W is introduced in line 1 and only moved between edges later. To see property (i) we use induction over the number of runs as coupled experiments.
Ind. start: In the beginning $x_i = a_i$ for all e_i by line 1.
Ind. hypothesis: For all t' up to some $t \le T$: $\mathbb{E}(x_i) = a_i$ for all e_i.
Ind. step: Given $\mathbb{E}(x_i) = a_i$ for all e_i after run t. For the expected value of x in line 1 we get:

$$\begin{aligned} \mathbb{E}(x) &= \int_0^1 x d(x) dx = \int_0^c x f_\ell dx + \int_c^1 x f_r dx = f_\ell \frac{c^2}{2} + f_r \frac{1-c^2}{2} \\ &= \frac{1-c}{c}\frac{c^2}{2} + \frac{c}{1-c}\frac{1-c^2}{2} = \frac{c-c^2}{2} + \frac{c}{1-c}\frac{(1-c)(1+c)}{2} = c \end{aligned}$$

And thus after $t+1$ we get $\mathbb{E}^{t+1}(x_i) = \mathbb{E}^t(x_i)$ for all e_i not chosen in run $t+1$. For the two affected edges we get (x_r analogously):

$$\mathbb{E}(x_\ell^{t+1}) = \mathbb{E}(x \cdot (x_\ell + x_r)) = \mathbb{E}(x) \cdot (\mathbb{E}^t(x_\ell) + \mathbb{E}^t(x_r)) = \frac{a_\ell \cdot (a_\ell + a_r)}{a_\ell + a_r} = a_\ell$$

□

[12] In practice, random draws with density d can be done, e.g., as follows. First decide which side of c to use with the help of a single Bernoulli trial that chooses ℓ with prob. $p(\ell) = c \cdot f_\ell = 1 - c$ (and r with prob. $p(r) = (1-c) \cdot f_r = c$). Then, choose a value x uniformly at random within the chosen interval.

By the linearity of the expectation operator, it is easy to see that the expected node weights in this model are $\mathbb{E}(\omega(v)) = \omega(v)$ as in G, only the edge weights in Equation 2.3.10 are needed for the proof analogous to the unweighted case (see Corollary 2.3.1).

loop-free case

The Loop-Free Case. As discussed above, the expected edge weights (see Equations 2.3.4 and 2.3.10) in the formula of *modularity* assume that loops are possible (see Equation 2.3.5). Suppose now we disallow loops, but still adopt the intuition that a randomly inserted edge should become incident with node v with probability proportional to $\deg(v)$ in the unweighted case. Analogous to Figure 2.3.1 and the derivation of *modularity*, we can now derive the probability of a random edge in a loop-free setup to become:

loop-free $p(e)$

$$p_\emptyset(\{u,v\}) = \underbrace{\frac{\deg(u)}{2m}\cdot\frac{\deg(v)}{2m-\deg(u)}}_{\substack{=p_\emptyset((u,v))\\ \text{“first connect to } u \text{ then to } v\text{”}}} + \underbrace{\frac{\deg(v)}{2m}\cdot\frac{\deg(u)}{2m-\deg(v)}}_{\substack{=p_\emptyset((v,u))\\ \text{“first connect to } v \text{ then to } u\text{”}}}$$
$$= \frac{\deg(u)\deg(v)\cdot(\overline{\deg}(u)+\overline{\deg}(v))}{2m\cdot\overline{\deg}(u)\overline{\deg}(v)} \qquad \text{using } \overline{\deg}(v) = 2m-\deg(v) \tag{2.3.12}$$

normed

Analogous to Equation 2.3.5 we can observe that this setup is normed, i.e., that the sum of all edge probabilities sum up to one. For easier summation we suppose for a moment that the graph was directed, and we write the above probability for edge $\{u,v\}$ as the sum of the probabilities of the two directed edges (u,v) and (v,u), as in the derivation of Equation 2.3.12. Note that without loops we now do not use $V^\times$ but rather $\{\{u,v\} \subseteq V \mid u \neq v\}$.

$$\sum_{\substack{\{u,v\}\in V\\ u\neq v}} p_\emptyset(\{u,v\}) = \sum_{\substack{\{u,v\}\subseteq V\\ u\neq v}} (p_\emptyset((u,v)) + p_\emptyset((v,u)))$$
$$= \sum_{v\in V}\sum_{\substack{u\in V\\ u\neq v}} p_\emptyset((v,u)) = \sum_{v\in V}\sum_{\substack{u\in V\\ u\neq v}} \frac{\deg(v)\deg(u)}{2m\cdot\overline{\deg}(v)}$$
$$= \frac{1}{2m}\sum_{v\in V}\frac{\deg(v)}{\overline{\deg}(v)}\sum_{\substack{u\in V\\ u\neq v}}\deg(u) = \frac{1}{2m}\sum_{v\in V}\frac{\deg(v)}{\overline{\deg}(v)}\overline{\deg}(v) = 1$$

$(\Omega_\emptyset, p_\emptyset)$

Using arguments from the previous sections we can now setup a discrete probability space $(\Omega_\emptyset, p_\emptyset)$ for loop-free graphs in an analogous way. By drawing m independent edges according to Equation 2.3.12 we obtain probabilities for graphs similar to Equation 2.3.6 and a lemma analogous to Lemma 2.3.1. Even our arguments concerning a weighted version (using total weight W) and a random process for weighted graphs in Section 2.3.2.1 carry over, yielding an expected edge weight of $\mathbb{E}_\emptyset(\omega(u,v)) = \omega(u)\omega(v)\cdot(\overline{\omega}(u)+\overline{\omega}(v))/(2\overline{\omega}(u)\overline{\omega}(v))$, using $\overline{\omega}(v) := 2W-\omega(v)$ (compare to Eq. 2.3.12). A variant $\mathit{modularity}_\emptyset$ for loop-free graphs could thus be defined as (compare to Formulas 2.3.3 and 2.3.9):

loop-free modularity

$$\mathrm{mod}_\emptyset(\mathcal{C}) := \sum_{\substack{\{u,v\}\subseteq V\\ u\neq v}} \left(\frac{\omega(u,v)}{W} - \frac{\omega(u)\omega(v)\cdot(\overline{\omega}(u)+\overline{\omega}(v))}{2W\cdot\overline{\omega}(u)\overline{\omega}(v)}\right)$$

$\mathbb{E}(\deg(v)) \nsim \deg(v)$!

It is important to note that this formulation does *not* fulfill Corollary 2.3.1: the intuition of making random edges incident to v with probability proportional to $\deg(v)$ (and its weighted analogon) does not generally preserve expected node degrees (weights) in the loop-free model. Devising such a model is much harder, simply using $p(\{u,v\}) = \deg(u)\deg(v)/(\text{normalized})$ does not work. Excluding parallel edges makes issues even worse, thus we stop here and postpone such thoughts to future work.

2.3.2.2 Implementations of the Lucidity Paradigm

The building blocks presented above enable us to study four implementations of the *lucidity* paradigm, namely, *coverage* and *performance* as quality indices and *subtraction* and *division* as the binary operators. Using *coverage* and subtraction, *modularity* is one of these implementations. For a discussion of *performance* [213] in weighted graphs we refer the reader to [46]. However, one aspect needs particular attention: *Performance* evaluates node pairs based on their being connected or not. Switching to weighted edges now requires a meaningful assumption (see [46]) about a maximum edge weight M to compare to, in order to measure, e.g., how missing inter-cluster edges contribute. The above main references for *performance* are not specific about M and thus three possible choices for M are immediate: $\omega_{\max}$ of G, 1 (being the maximum allowed edge weight), or W. The canonic formulation is (compare Equation 1.2.3)

weighted performance

$$\text{perf}_\omega = \frac{\omega(\mathcal{C}) + M\overline{m}(\mathcal{C})^c + M\overline{m}(\mathcal{C}) - \overline{\omega}(\mathcal{C})}{\frac{1}{2}n(n-1)M} \quad . \tag{2.3.13}$$

We first need to derive its expected value, which is slightly more laborious compared to using *coverage*. In particular any choice for M which is a parameter dependent on G (such as $\omega_{\max}$) becomes a random variable in (Ω, p). Even worse, it is not independent of other edge weights, which renders the expected *performance* something we cannot easily specify, using our models from Section 2.3.2.1. However, there is a more fundamental objection against using $\omega_{\max}$: in a random model it is a highly questionable assumption that for each drawn graph weights are compared to the maximum edge weight occurring in this particular draw. On the other hand, keeping the value of $\omega_{\max}$ in G as a fixed constant for all draws raises the question why exactly G should pose the maximum edge weight for the whole probability space—a value any other single draw from the space will attain with zero probability. As a better choice for M, the range of the weight function ω should be used. This can be 1 if the application yields this limit, or W (we shall see later, that the exact choice does not have a decisive influence). Thus, in the following we assume M to be some choice of a constant, which enables us to compute the expected *performance*. This leads to the following lemma:

Lemma 2.3.5 *Using the probability space described in Section 2.3.2.1 and an arbitrary but fixed constant M, the expected value of performance is (for unweighted edges set $\omega(e) \equiv 1 = M$)* $\mathbb{E}(\text{perf})$

$$\frac{\sum_{C\in\mathcal{C}}(\sum_{v\in C}\omega(v))^2/W + M(n^2 - \sum_{C\in\mathcal{C}}|C|^2) - 2W}{n(n-1)M}$$

Proof. For a simpler representation we split Equation 2.3.13 into edges (first term in numerator) and non-edges (last three terms in numerator). Again we use (Ω, p), i.e., Equation 2.3.10 for expected edge weights.

$$\underbrace{\mathbb{E}\left(\frac{\omega(\mathcal{C})}{\frac{1}{2}n(n-1)M}\right)}_{\mathbb{E}_1} = \frac{\frac{1}{4W}\sum_{C\in\mathcal{C}}\left(\sum_{v\in C}\omega(v)\right)^2}{\frac{1}{2}n(n-1)M} = \frac{\frac{1}{2W}\sum_{C\in\mathcal{C}}\left(\sum_{v\in C}\omega(v)\right)^2}{n(n-1)M}$$

$$\underbrace{\mathbb{E}\left(\frac{M\overline{m}(\mathcal{C})^c + M\overline{m}(\mathcal{C}) - \overline{\omega}(\mathcal{C})}{\frac{1}{2}n(n-1)M}\right)}_{\mathbb{E}_2} = \frac{\mathbb{E}\left(\sum\limits_{\substack{e=\{u,v\}\in E\\ \mathcal{C}(u)\neq\mathcal{C}(v)}} (M-\omega(e)) + \sum\limits_{\substack{\{u,v\}\notin E\\ \mathcal{C}(u)\neq\mathcal{C}(v)}} M\right)}{\frac{1}{2}n(n-1)M}$$

$$= \frac{\frac{1}{2}\sum\limits_{C\in\mathcal{C}}\sum\limits_{C'\in\mathcal{C}\setminus C}\sum\limits_{(v,w)\in C\times C'}\left(M - \frac{\omega(v)\omega(w)}{2W}\right)}{\frac{1}{2}n(n-1)M}$$

$$= \frac{M(n^2-\sum_{C\in\mathcal{C}}|C|^2)}{n(n-1)M} - \frac{\frac{1}{4W}\sum_{C\in\mathcal{C}}\sum_{v\in C}\omega(v)(2W-\sum_{v\in C}(\omega(v)))}{\frac{1}{2}n(n-1)M}$$

$$= \frac{M(n^2-\sum_{C\in\mathcal{C}}|C|^2)}{n(n-1)M} - \frac{\frac{1}{4W}4W^2 - \frac{1}{4W}\sum_{C\in\mathcal{C}}(\sum_{v\in C}\omega(v))^2}{\frac{1}{2}n(n-1)M}$$

$$= \frac{M(n^2-\sum_{C\in\mathcal{C}}|C|^2)}{n(n-1)M} - \frac{2W - \frac{1}{2W}\sum_{C\in\mathcal{C}}(\sum_{v\in C}\omega(v))^2}{n(n-1)M}$$

$$\mathbb{E}(\text{perf}_\omega) = \mathbb{E}_1 + \mathbb{E}_2 = \frac{\frac{1}{W}\sum\limits_{C\in\mathcal{C}}\left(\sum\limits_{v\in C}\omega(v)\right)^2 - 2W + M\left(n^2 - \sum\limits_{C\in\mathcal{C}}|C|^2\right)}{n(n-1)M} \tag{2.3.14}$$

□

four implem. of lucidity

We can now state an overview summarizing the formulas of the resulting four implementations of the *lucidity* paradigm in Table 2.3.1. Note that the weighted versions of *lucidity* are true generalizations of the unweighted cases, since setting each weight to 1 yields the unweighted formulas. Thus, we restrict our analyses to the weighted case. The straightforward weighted variant of $\mathrm{L}^-_{\mathrm{cov}}$ has been described by [172]. Based on Table I we now define the following implementations:

$$\mathrm{L}^-_{\mathrm{cov}} := \mathrm{cov} - \mathbb{E}[\mathrm{cov}] \quad \text{(equals modularity)} \qquad \mathrm{L}^{\div}_{\mathrm{cov}} := \frac{\mathrm{cov}}{\mathbb{E}[\mathrm{cov}]} \tag{2.3.15}$$

$$\underbrace{\mathrm{L}^-_{\mathrm{perf}} := \mathrm{perf} - \mathbb{E}[\mathrm{perf}]}_{\text{absolute variants (subtractive)}} \qquad \underbrace{\mathrm{L}^{\div}_{\mathrm{perf}} := \frac{\mathrm{perf}}{\mathbb{E}[\mathrm{perf}]}}_{\text{relative variants (divisive)}} \tag{2.3.16}$$

measure	*coverage*	*performance*
$\mathcal{M}$	$\frac{m(\mathcal{C})}{m}$	$\frac{m(\mathcal{C})+\overline{m}(\mathcal{C})^c}{0.5\cdot n(n-1)}$
$\mathbb{E}[\mathcal{M}]$	$\sum_{C\in\mathcal{C}}\left(\frac{\sum_{v\in C}\deg(v)}{2m}\right)^2$	$\frac{\sum_{C\in\mathcal{C}}((\sum_{v\in C}\deg(v))^2/m-(\sum_{v\in C}1)^2)+n^2-2m}{n(n-1)}$
$\mathcal{M}_\omega$	$\frac{\omega(\mathcal{C})}{W}$	$\frac{\omega(\mathcal{C})+M\overline{m}(\mathcal{C})^c+(M\overline{m}(\mathcal{C})-\overline{\omega}(\mathcal{C}))}{0.5\cdot n(n-1)M}$
$\mathbb{E}[\mathcal{M}_\omega]$	$\sum_{C\in\mathcal{C}}\left(\frac{\sum_{v\in C}\omega(v)}{2W}\right)^2$	$\frac{\sum_{C\in\mathcal{C}}(\sum_{v\in C}\omega(v))^2/W+M(n^2-\sum_{C\in\mathcal{C}}\lvert C\rvert^2)-2W}{n(n-1)M}$

Table 2.3.1. Quality indices and expected values (M: maximum edge weight in the model). The subscript "ω" indicates edge-weighted versions.

As we shall see in Section 2.3.4, some of these implementations differ significantly in their behavior, although they are all derived from the same paradigm. However, to our surprise we found that $\mathrm{L}^-_{\mathrm{perf}}$ and $\mathrm{L}^-_{\mathrm{cov}}$, are in fact equivalent, which we show in detail in the following subsection. However, we can already make an interesting preliminary observation towards that result:

constant M is a factor in $\mathrm{L}^-_{\mathrm{perf}}$

Corollary 2.3.2 *A constant M for weighted $\mathrm{L}^-_{\mathrm{perf}}$ is a scaling factor, which means that an observation $\mathrm{L}^-_{\mathrm{perf}}(\mathcal{C}(G)) \geq \mathrm{L}^-_{\mathrm{perf}}(\mathcal{C}'(G))$ is M-invariant.*

Proof. From Lemma 2.3.5 it it not hard to see, that some terms from perf_ω have survived in $\mathbb{E}(\mathrm{perf}_\omega)$, which for simplicity we denote by Φ: Φ

$$\Phi = M\overline{m}(\mathcal{C})^c + M\overline{m}(\mathcal{C}) = \frac{1}{2}M(n^2 - \sum_{C\in\mathcal{C}}|C|^2) \tag{2.3.17}$$

Rewriting and summarizing $\mathrm{L}^-_{\mathrm{perf}}$ yields the following term, which uses M only as a factor in the denominator, as an inverse scaling factor.

$$\begin{aligned}
\mathrm{L}^-_{\mathrm{perf}} &= \mathrm{perf}_\omega - \mathbb{E}(\mathrm{perf}_\omega) \\
&= \frac{\omega(\mathcal{C}) + \Phi - \overline{\omega}(\mathcal{C})}{\frac{1}{2}\cdot n(n-1)M} - \frac{\sum_{C\in\mathcal{C}}(\sum_{v\in C}\omega(v))^2/W + 2\Phi - 2W}{n(n-1)M} \\
&= \frac{\omega(\mathcal{C}) - \overline{\omega}(\mathcal{C}) - \frac{1}{2W}\sum_{C\in\mathcal{C}}(\sum_{v\in C}\omega(v))^2 - W}{\frac{1}{2}n(n-1)M}
\end{aligned} \tag{2.3.18}$$

□

We refrain from a discussion of the usage of *lucidity* on graphs with a fuzzy clustering, which allows clusters to overlap, i.e., nodes may belong to several clusters. However we point the reader to two recent works which consistently generalize *modularity* to the overlapping case. These are [180], which also proposes a generalization to directed graphs, and [170] which discusses the former and proposes sound improvements. Summarizing, the introduction of *belonging factors* of nodes to clusters, as proposed by these two works, can immediately be applied to *coverage* and *performance* and thus also to the implementations of *lucidity* discussed herein, but not necessarily to any implementation.

2.3.2.3 The Equivalence of $\mathrm{L}^-_{\mathrm{perf}}$ and $\mathrm{L}^-_{\mathrm{cov}}$

As we have seen in Section 2.2.2, $\mathrm{L}^-_{\mathrm{cov}}$ can be optimized via ILP[13] formulation: Constraints ensure a consistent partition of the nodes by formalizing an equivalence relation on the nodes, deciding whether two nodes are in the same cluster. The linear objective function follows directly from the weighted version of Equation 2.3.3:

$$\text{weighted } \mathrm{L}^-_{\mathrm{cov}} = \sum_{\{u,v\}\in V^\times}\left(\frac{\omega(u,v)}{W}X_{uv}\right) - \sum_{(u,v)\in V^2}\left(\frac{\omega(u)\omega(v)}{4W^2}X_{uv}\right) \tag{2.3.19}$$

$$\text{with } X_{uv} = [\delta_{uv} =] \begin{cases} 1 & \text{if } u,v \text{ in same cluster} \\ 0 & \text{otherwise} \end{cases}$$

rewriting $\mathrm{L}^-_{\mathrm{perf}}$

A similar formulation is possible for $\mathrm{L}^-_{\mathrm{perf}}$. Using the same framework of constraints it is not hard to see that a linear objective function can be derived from Table 2.3.1. We first build upon the formula for $\mathrm{L}^-_{\mathrm{perf}}$ derived in Equation 2.3.18, and rewrite it, working towards a similar shape as used in Equation 2.3.19:

[13] ILP stands for integer linear program.

$$
\begin{aligned}
\text{weighted } \mathrm{L}^-_{\text{perf}} &= \frac{\omega(\mathcal{C}) - \overline{\omega}(\mathcal{C}) - \frac{1}{2W}\sum_{C\in\mathcal{C}}(\sum_{v\in C}\omega(v))^2 - W}{\frac{1}{2}n(n-1)M} \\
&= \frac{\sum_{\{u,v\}\in V^2}\omega(u,v)X_{uv} - \sum_{\{u,v\}\in V^2}\omega(u,v)(1-X_{uv}) - \sum_{(u,v)\in V^2}\frac{\omega(u)\omega(v)}{2W}X_{uv} - W}{\frac{1}{2}n(n-1)M} \\
&= \frac{2\sum_{\{u,v\}\in V^2}\omega(u,v)X_{uv} - \frac{1}{2W}\sum_{(u,v)\in V^2}\omega(u)\omega(v)X_{uv} - 2W}{\frac{1}{2}n(n-1)M} \\
&= \frac{\sum_{\{u,v\}\in V^2}\frac{\omega(u,v)}{W}X_{uv} - \sum_{(u,v)\in V^2}\frac{\omega(u)\omega(v)}{4W^2}X_{uv}}{\underbrace{\frac{1}{4W}n(n-1)M}_{a}} - \underbrace{\frac{1}{\frac{1}{4W}n(n-1)M}}_{b}
\end{aligned}
\tag{2.3.20}
$$

We now trim Formula 2.3.20 by removing the second summand (b) and the (main) denominator (a), which are both invariant under X_{uv} and obtain Formula 2.3.19. This yields the following lemma:

$\mathrm{L}^-_{\text{perf}} \doteq \mathrm{L}^-_{\text{cov}}$

Lemma 2.3.6 (Equivalence of $\mathrm{L}^-_{\text{perf}}$ and $\mathrm{L}^-_{\text{cov}}$) *The problem of optimizing $\mathrm{L}^-_{\text{perf}}$ and that of optimizing $\mathrm{L}^-_{\text{cov}}$ are equivalent, furthermore*

$$
\mathrm{L}^-_{\text{cov}}(G,\mathcal{C}_1) > \mathrm{L}^-_{\text{cov}}(G,\mathcal{C}_2) \iff \mathrm{L}^-_{\text{perf}}(G,\mathcal{C}_1) > \mathrm{L}^-_{\text{perf}}(G,\mathcal{C}_2) \tag{2.3.21}
$$

This lemma together with the NP-completeness of optimizing *modularity* [44], immediately gives us the following corollary:

$\mathrm{L}^-_{\text{perf}}$ *NP-complete*

Corollary 2.3.3 *Given a graph G (weighted or unweighted) and a real L. It is NP-complete to decide whether there is a clustering $\mathcal{C}(G)$ with $\mathrm{L}^-_{\text{perf}}(\mathcal{C}(G)) \geq L$.*

The deduction of the equivalence in Lemma 2.3.6 implies that a linear relation between the values of $\mathrm{L}^-_{\text{perf}}$ and $\mathrm{L}^-_{\text{cov}}$ for a given instance G and an arbitrary clustering $\mathcal{C}(G)$ can be given in the form $\mathrm{L}^-_{\text{perf}} = a(G)\cdot\mathrm{L}^-_{\text{cov}} + b(G)$. Coefficients a and b both depend on the instance G and are the very terms mentioned above (see Equation 2.3.20). Together with the fact that both $\mathrm{L}^-_{\text{perf}}$ and $\mathrm{L}^-_{\text{cov}}$ can attain the value 0, even for the respective optimum clusterings, this yields that relative approximation guarantees do not easily carry over in either direction. In any way, to our best knowledge, no positive results on the approximability of either $\mathrm{L}^-_{\text{perf}}$ or $\mathrm{L}^-_{\text{cov}}$ exist.

We briefly discuss how Formula 2.3.18 can be trimmed further, such that in Formula 2.3.22 we obtain a very simple but equivalent objective function for maximizing $\mathrm{L}^-_{\text{cov}}$ (or $\mathrm{L}^-_{\text{perf}}$) in, e.g., an ILP (note that $X_{uu} \equiv 1$):

$\mathrm{L}^-_{\text{cov}}$ *simplified*

$$
\begin{aligned}
&\sum_{\{u,v\}\in V^\times}\left(\frac{\omega(u,v)}{W}X_{uv}\right) - \sum_{(u,v)\in V^2}\left(\frac{\omega(u)\omega(v)}{4W^2}X_{uv}\right) \\
&= \sum_{\{u,v\}\in\binom{V}{2}}\left(\frac{\omega(u,v)}{W}X_{uv}\right) + \underbrace{\sum_{v\in V}\left(\frac{\omega(v,v)}{W}\right)}_{X\text{-invariant}} - \sum_{\{u,v\}\in\binom{V}{2}}\left(\frac{\omega(u)\omega(v)}{2W^2}X_{uv}\right) - \underbrace{\sum_{v\in V}\left(\frac{(\omega(v))^2}{4W^2}\right)}_{X\text{-invariant}} \\
&\cong \sum_{\{u,v\}\in\binom{V}{2}}\left(\left(\omega(u,v) - \frac{\omega(u)\omega(v)}{2W}\right)X_{uv}\right)
\end{aligned}
\tag{2.3.22}
$$

2.3.2.4 The Relation to MinMixedMultiPartition

Note that the ILP formulation in Equation 2.3.22 has an equivalent *metric* version, i.e., $X_{uv} = 1$ iff nodes u and v are in *different* clusters. The problem thus changes to minimizing the same objective function: Instead of maximizing the edge contributions inside clusters, we minimize those in between. It becomes obvious that the problem of optimizing this index is equivalent to finding the minimum weight edge set inducing a (multi-)partition on the complete graph $\mathcal{K}$ on V, where edge weights g are equal to the (simplified) term in brackets in Equation 2.3.22. Given an unweighted instance of $\mathrm{L}^-_{\mathrm{cov}}$ (i.e., *modularity*), edge weights in $\mathcal{K}$ are multiples of $1/m$, thus we can assume $g \in \mathbb{Z}$. We formalize the general form of this problem as follows:

cut-type view of modularity

Definition 2.3 (MinMixedMultiPartition) *Consider an undirected graph $\mathcal{K} = (V, E)$, an edge weight function $g : E \to \mathbb{Z}$ and a rational number L. Is there a partition of V into disjoint subsets $V_1, \ldots, V_m$ ($m \geq 1$) such that the sum of weights of edges whose endpoints lie in different subsets is at most L?*

MinMixed-MultiPartition

By the fact that optimizing $\mathrm{L}^-_{\mathrm{cov}}$ is NP-hard, we thus obtain the following corollary:

Corollary 2.3.4 *The problem* MinMixedMultiPartition *is NP-hard.*

MinMixed-MultiPartition *is NP-hard*

For a weighted $\mathrm{L}^-_{\mathrm{cov}}$ instance, a similar observation holds, if we assume that original edge weights are rational, $\omega(e) \in \mathbb{Q}$. Although many similar hardness results on cuts in graphs exist, we are not aware of a proof of this particular variant. Well known hardness results in this context have been presented, e.g., by [106] for GraphPartition or MaxCut, and by [61] for MinimumMultiwayCut, where in a positively weighted graph a set of terminals $T \subseteq V$ has to be separated. Note that MinMixedMultiPartition is not, as it might seem, a straightforward generalization of the NP-hard problem MixedMinCut (i.e., MaxCut), in that the set of cut edges of a MixedMinCut is a subset of the set of edges cut by MinMixedMultiPartition, as can be disproven by the simple example in Figure 2.3.8. Moreover instances exist where the solution to MinMixedMultiPartition is the trivial partition $\{V\}$ (e.g., in the case of exclusively positive weights), such that for obvious reasons no MixedMinCut can be deduced. This emphasizes the relevance of Corollary 2.3.4.

MixedMinCut $\not\subseteq$ MinMixed-MultiPartition

2.3.3 Lucidity-Clustering Algorithms

With the optimization of $\mathrm{L}^-_{\mathrm{cov}}$ being NP-complete ([45]; [44]) encourages the usage of heuristics or approximations. In this section, we briefly describe the algorithms we use for *lucidity* maximization. Throughout our experiments, we employ a greedy heuristic approach, allowing for a consistent evaluation of the four variants of lucidity, as follows.

lucidity-greedy heuristic

For a given *lucidity* measure L the greedy algorithm starts with the singleton clustering and iteratively merges those two clusters that yield the largest increase or the smallest decrease in L. After a maximum of $n-1$ merges the intermediate clustering that achieved the highest

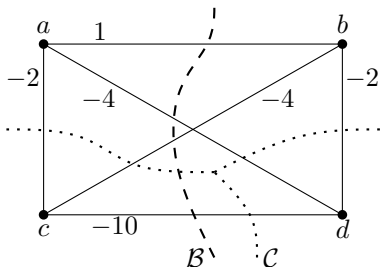


Figure 2.3.8. The (unique) minimum multi-partition $\mathcal{C}$ with $\mathrm{cost}(\mathcal{C}) = -22$ does not directly induce the (unique) minimum bipartition $\mathcal{B}$ with $\mathrm{cost}(\mathcal{B}) = -17$.

value of L is returned. Thus, this approach is parameter-free, and not even the number of clusters has to be specified as a parameter in advance. Note that this procedure can thus return a complete *hierarchy* of clusterings. The algorithm maintains a symmetric matrix ΔL with entries $\Delta L_{i,j}$ equaling $L(\mathcal{C}_{i,j}) - L(\mathcal{C})$, where $\mathcal{C}$ is the current clustering and $\mathcal{C}_{i,j}$ is obtained from $\mathcal{C}$ by merging clusters C_i and C_j. The pseudo-code for the greedy algorithm is given in Algorithm 3.[14] Apart from a number of engineering approaches on this fundamental algorithm, a recent work [181] explores, among other things, how in a postprocessing stage the result of this greedy approach can be improved. In particular, the authors investigate how large clusters can later be split up in order to make up for previous decisions of the greedy approach which were not optimal. In this work, however, we refrain from delving into the many variants of this pure greedy approach that can be found in the literature, for the sake of brevity.

hierarchical *ΔL*

Algorithm 3: Greedy Lucidity

Input: Graph $G = (V, E, \omega)$
Output: Clustering $\mathcal{C}$ of G
1 $\mathcal{C} \leftarrow$ Singletons, initialize L
2 Initialize matrix ΔL (with $\Delta L_{ij} \cong$ change in L when merging C_i and C_j)
3 **while** $|\mathcal{C}| > 1$ **do**
4 Find $\{i, j\}$ with $\Delta L_{i,j} = \arg\max \Delta L_{ij}$
5 Merge clusters C_i and C_j
6 Update ΔL
7 $L \leftarrow L + \Delta L_{i,j}$
8 Return intermediate clustering with highest *lucidity*

Let ΔL be defined by matrices $\Delta\mathcal{M}$ and $\Delta\mathbb{E}[\mathcal{M}]$, denoting the additive changes in $\mathcal{M}$ and in $\mathbb{E}[\mathcal{M}]$, respectively. For *coverage* and *performance* it is not hard to see that elements $\Delta\mathcal{M}_{ij}$ and $\Delta\mathbb{E}[\mathcal{M}]_{ij}$ only depend on $E(C_i)$, $E(C_j)$ and $E(C_i, C_j)$, i.e., on local information. Then, when merging C_i and C_j, entries $\Delta\mathcal{M}_{pq}$ of unaffected clusters do not change, while entries in rows and columns i and j are updated as follows: $\Delta\mathcal{M}_{k,(ij)} := \Delta\mathcal{M}_{k,i} + \Delta\mathcal{M}_{k,j}$, where $C_{(ij)} = C_i \cup C_j$ (and $\Delta\mathbb{E}[\mathcal{M}]$ is updated analogously). From Equations 2.3.15 and 2.3.16 it becomes clear that for the absolute variants this transfers to ΔL, while for the relative variants all entries of ΔL change, even those of unaffected clusters.

merge is local and quick

2.3.3.1 Runtime Analysis

Both absolute variants of *lucidity* share the same asymptotic running time. Employing a standard data structure for clusterings, one observes that Step 1 and 8 run in $O(n)$ time. Matrix ΔL is initialized in $O(n^2)$ time. The loop at Step 3 is executed $n - 1$ times. Step 4 runs in $O(n)$ time, if we store the rows of ΔL as heaps. Merging two clusters (Step 5) and updating L (Step 7) require at most linear time.[15] Thus, updating ΔL dominates, which consists of $O(n)$ insertions and deletions from heaps, requiring $O(\log n)$ each. This yields the following lemma:

$O(n^2 \log n)$

Lemma 2.3.7 *Algorithm 3 runs in $O(n^2 \log n)$ time for the absolute variants.*

Adapting Lemma 2.3.7 to the relative variants yields a runtime of $O(n^3)$, since a merge entails an update of n^2 matrix entries. However, in Lemma 2.3.8 and Algorithm 4 we improve this upper bound.

It is not hard to see that the first local optimum of L, that the absolute greedy heuristic attains, is its global optimum, since then the matrix ΔL is non-positive, allowing no further

[14] For compactness of representation we accept a few repetitions from Section 2.2.5.
[15] Note that we need not initialize the full matrix, as only merges along connected pairs of nodes allow for positive changes, while this tweak is helpful, practically, it does not lower the asymptotic runtime.

increase in L or any entry ΔL_{xy}. In case the number of clusters is dependent on n, i.e., $|\mathcal{C}| \in \omega(1)$, this may result in an asymptotic decrease in running time.

2.3.3.2 Quick Divisive Merge

In this section we describe how for relative variants, the running time for updating ΔL can be reduced by avoiding explicit matrix updates. We give an algorithm that updates ΔL in $O(n \log n)$ amortized time using a geometric embedding.

geometric rep. for $L_*^{\div}$

We store matrix ΔL by a point set P in the plane as follows and as depicted in Figure 2.3.9. Each entry $\{i, j\}$ is represented by a point p_{ij} with coordinates $p_{ij} := (\mathcal{M}(\mathcal{C}_{i,j}), \mathbb{E}[\mathcal{M}(\mathcal{C}_{i,j})]) = (\mathcal{M} + \Delta\mathcal{M}_{ij}, \mathbb{E}[\mathcal{M}] + \Delta\mathbb{E}[\mathcal{M}]_{ij})$. Thus, each point encodes the measure (y-axis) of a clustering and its expectation (x-axis). Since these are both non-negative, all points are in quadrant one. We additionally insert one point $R = (\mathcal{M}(\mathcal{C}), \mathbb{E}[\mathcal{M}(\mathcal{C})])$ that represents the current clustering.

Since $\mathcal{M}$ and $\mathbb{E}[\mathcal{M}]$ update additively, we can update each point p in the plane, after merging two clusters C_k and C_l, as follows: First, for a linear number of points p (i.e., those involving C_k and C_l), we set $p_{i,(kl)} \leftarrow p_{i,k} + p_{i,l} - R$. By doing so we actually both delete and introduce a linear number of points. Second, for all points P, including those newly introduced, we set $p \leftarrow p + (p_{kl} - R)$. These steps maintain the data structure.

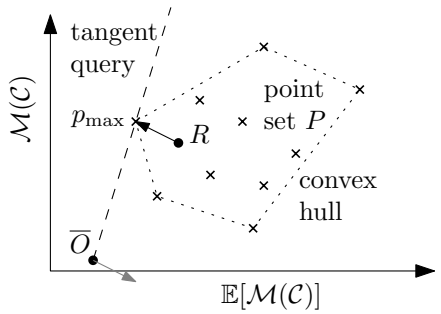


Figure 2.3.9. Each cross encodes the quality of some merge, with $p_{\max}$ yielding the highest quotient. Instead of all crosses, $\overline{O}$ moves antipodally by $R - p_{\max}$ (gray arrow). Due to some earlier step, $\overline{O}$ has already been shifted away from $(0, 0)$.

There are two crucial observations: First, instead of uniformly setting $p \leftarrow p + (p_{kl} - R)$, we can save $\Omega(n^2)$ such updates by only shifting the origin: $\overline{O} \leftarrow \overline{O} - (p_{kl} - R)$. Second, at any time, the merge maximizing $L_*^{\div}$ corresponds to the point $p_{\max}$ that maximizes $y(p)/x(p)$. Point $p_{\max}$ must lie on the convex hull of P, and can be found by a *tangent query* through the origin $\overline{O}$. Such a query reports the tangents on the hull that pass through a given point. Initially $\overline{O}$ is set to $(0, 0)$, but each merge shifts this imaginary origin, which serves as the vantage point of the tangent queries. Figure 2.3.9 illustrates these observations. We thus need a data structure that maintains the convex hull of a fully dynamic point set P and that allows for quick tangent queries.

move origin, find max. on conv. hull

In fact [49] present such a data structure, using so-called *kinetic heaps*. It uses linear space (i.e., $O(n^2)$, in our case), handles both insertions into and deletions from P, as well as tangent queries to the convex hull in amortized time $O(\log n)$. This data structure is described more extensively in the dissertation of [139], where, among other things, it is proven that the amortized performance of this data structure is in fact optimal. Since detailing this data structure is far beyond the scope of this paper, we just give a rough idea and use it as a black box. The points are stored in several instances of a semi-dynamic data structure that supports deletions. Insertions result in new instances, which are merged with *rank degree* $\log n$ by a semi-dynamic data structure that supports constant time deletions. Then, the core data structure is built which maintains the *convex hull* of two such merged sets. On top of that data structure, a *kinetic heap* is then built, which finally handles queries and operations. A *kinetic* (or *parametric*) *heap* is a generalization of a *priority queue*, such that the entries are linear functions that change over time. The authors use *interval trees* as secondary structures for answering containment queries.

a supporting data structure

Given this data structure, Algorithm 4 performs the update in Line 6 of Algorithm 3 in time $O(n \log n)$. First, a tangent query from $\overline{O}$ to the convex hull of P finds $p_{\max}$ (Line 1) in time $O(\log n)$. Then, after storing the merge of C_l and C_k (Line 2) in at most linear time, a

linear number of points $p_{i,k}$ and $p_{i,l}$ are replaced by a new point $p_{i,(kl)}$ (Line 3). After each such replacement the convex hull of P is maintained in time $O(\log n)$ by the data structure. Finally, reference point R is set to the newly improved coordinates (Line 4), and origin $\overline{O}$ is shifted (Line 5) in constant time. This last step is crucial in terms of running time, since it saves the update of *all* $\Omega(n^2)$ points in P. Thus, we arrive at the following:

$O(n^2 \log n)$

Lemma 2.3.8 *By employing* quick divisive merge *(Algorithm 4), Algorithm 3 runs in* $O(n^2 \log n)$ *time for the relative variants.*

Algorithm 4: Quick Divisive Merge

Input: $\Delta\mathcal{M}, \Delta\mathbb{E}[\mathcal{M}]$, data structure with of points P as described above, reference R, (shifted) origin $\overline{O}$

Output: Best merge, updated matrices $\Delta'\mathcal{M}, \Delta'\mathbb{E}[\mathcal{M}]$

1 Find $p_{\max} = p_{kl}$ with tangent query through R
2 Merge clusters C_k and C_l of point $p_{kl} = p_{\max}$
3 For all clusters C_i insert $p_{i,(kl)} := p_{i,k} + p_{i,l} - R$, delete $p_{i,k}$, $p_{i,l}$
4 $R \leftarrow p_{\max}$
5 $O \leftarrow O - (p_{kl} - R)$

Note that the above lemma generalizes to all implementations of lucidity, where a merge of two clusters entails an addition of corresponding entries of ΔL (or of $\Delta\mathcal{M}_w$ and $\Delta\mathbb{E}[\mathcal{M}_w]$).

2.3.4 Experimental Evaluation

two questions:

The aim of this section is to experimentally evaluate the behavior of *lucidity* and of *lucidity*-based clustering algorithms in a systematic way. We proceed in two steps and start with the measure *lucidity* itself:

1. lucidity vs. human intuition

. . . on ground truth

. . . on benchmark algorithms

1. ***Lucidity* vs. Human Intuition.** The key idea of this part is to evaluate how well *lucidity* quantifies the human intuition of the quality of a graph clustering. In a first step we examine the behavior of *lucidity* on generated *ground-truth* clusterings. These generated clusterings are built by a basic random generator which features an unarguable and intuitive mechanism for tuning the clarity of the implanted clustering. We thereby check whether our implementations of *lucidity* yield results that are in accordance with human intuition of "better" or "worse" clusterings.

   We then cluster the generated graphs with established clustering algorithms and repeat our measurements of *lucidity*. This second set of experiments is less controlled than that which uses the generator, but reduces the dependency of our findings on the generator's clustering. Next we turn to *lucidity*-driven clustering.

2. quality of greedy lucidity

. . . wrt. lucidity and other measures

2. **Quality of Greedy *Lucidity*.** The experiments described above serve to corroborate that *lucidity* may be used to quantify the goodness of a graph clustering. In this second setup, we then try to find out how well *lucidity*-driven algorithms, and the proposed greedy agglomerative algorithms in particular, work in practice. To this end, we use three established quality indices and *lucidity* itself, and systematically measure the quality of the clusterings found by our *lucidity*-based clustering algorithms from Section 2.3.3. Here we again use our generator for clustered random networks with scalable clarity. We thereby compare our algorithms to three established ones which serve as benchmarks. The question we want to answer is: How well do *lucidity*-based clustering algorithms compete with other algorithms in terms of quality?

   The reason for using several measures and algorithms for a comparison is simple: it is folklore in the field of graph clustering that there is no single best strategy or measure, thus the reader needs several vantage points for assessing our results.

In the following section we describe the general model used for the experimental evaluation, then we present and discuss the results on the two questions stated above.

2.3.4.1 The Experimental Setup

We employ an adaption of the benchmark used in [47, 48]. For further details on this experimental setup we refer the reader to these references and restrict ourselves to a brief sketch at this point. Starting with a fixed set $V = \{1, \ldots, n\}$ of nodes, a random partition generator $\mathcal{P}(n, s, \nu)$ partitions V into $(P_1, \ldots, P_k)$. For the distribution of $|P_i|$ we use $|P_i| \sim \mathcal{N}(s, \frac{s}{\nu})$, with $s = n/k$. This simple process constrains $|P_k|$ (and possibly even predecessors); this is dealt with by setting $|P_k| = n - \sum_{i<k} |P_i|$ iff this yields $||P_k| - s| < s/3$, otherwise the partition is rejected and a new one drawn. Given a partition, this is used as the clustering. Then, for all $e \in V^\times$ edges are introduced inside and between clusters with probabilities p_{in} and p_{out}, respectively. Finally a random weight ω is assigned to each edge with $\omega \sim \mathcal{U}([0, p_{\text{in}}])$[16] or $\omega \sim \mathcal{U}([0, p_{\text{out}}])$, respectively. In case the resulting graph is disconnected, additional edges between random nodes of disconnected components are drawn. In our experiments we used $n = 100$ and $n = 1000$, and choose $k \sim \mathcal{U}([\log n, \sqrt{n}])$, $\nu = 4$.

the generator

p_{in}, p_{out}

We roughly refer to combinations of p_{in} and p_{out} supporting *dense*, *sparse*, *strong* and *random* community structure by A_{dense}, A_{sparse}, A_{strong} and $A_{\text{rand.}}$, respectively, as sketched out in Figure 2.3.10.

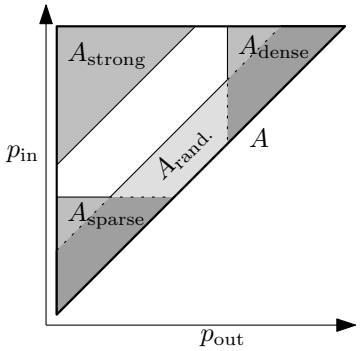


Figure 2.3.10. Combinations of p_{in}, p_{out} and their rough naming.

A

We then let the *lucidity* algorithms, based on Algorithm 3 and on the four variants (see Section 2.3.2.2) compete with reference algorithms on these instances. We restrict ourselves to *Markov Clustering* (MCL) [213], *Geometric MST Clustering* (GMC) [48] and *Iterative Conductance Cutting* (ICC)[17] [145] for comparison and to *lucidity*, *coverage*, *performance* and *inter-cluster conductance* (see [46]) for measuring clustering quality alongside structural aspects, such as the number of clusters. We keep the number of algorithms for comparison limited as they only serve as a benchmark. Although there exist a number of alternative approaches that work towards the maximization of *modularity* (and could thus also be applied to *lucidity*), we refrain from including any of them in our study as it has been repeatedly shown (please refer to the references in the introduction of this section) that all these largely similar approaches only marginally differ in both the structure of the identified clustering and the measured *modularity*.

benchmark algorithms

quality indices

We systematically conducted experiments using 100 and 1000 nodes, for all combinations of $p_{\text{in}} > p_{\text{out}}$ in steps of 0.05. We repeated each setup, until mean measured qualities were estimated to lie within a confidence interval of length $2 \cdot 0.05$ around the measured mean with an α-level (probability) of 0.95. We separately required this level of significance for each quality index measured. In total about one million total runs were conducted. Effective runtimes of our very basic Java 1.5 implementations ranged from a few milliseconds for 100 nodes using absolute variants to several seconds for 1000 nodes using relative variants, on an AMD Opteron 2.2 GHz. In addition to this systematic evaluation, we show exemplary results on two real-world networks in Section 2.3.4.3.

statistical significance

As a side note, it is worth mentioning that methodologies for the experimental evaluation of graph algorithms, either in a conceptual sense or in the context of algorithm engineering,

[16] The uniform distribution over the interval $[a, b]$ is denoted as $\mathcal{U}([a, b])$

[17] *ICC* uses a threshold which determines when the cutting strategy of the algorithm should stop, we use 0.4; for *GMC* we use embedding dimension 2 and the geometric mean of *coverage*, *performance* and *inter-cluster conductance* as the objective function; for *MCL* we use *expansion* = 2 and *reduction* = 2; note that while for *ICC* the threshold directly influences the number of clusters, the other two algorithms automatically determine this number.

are a big topic on its own right. Due to the non-generality of specific real-world networks such as the well-known network of bottlenose dolphins [157], the karate club [230], phone calls networks [38] etc., graph clustering methods have often been evaluated with artificially generated graphs, most notably using so called *ad hoc* networks (see, e.g., [173, 63]). This rather restrictive generation method divides a set of 128 nodes into four clusters of equal size and then uses parameters p_{in} and p_{out} as above. Although our approach is more general, we are aware that it is still far from comprehensive, and thus potentially subject to some bias introduced by a dependency between the generator, the algorithm and even the quality measures. Variant approaches for such evaluations have been proposed, e.g., by [91] and [25] but a setup as general and as methodologically sound as proposed by [100] is beyond the scope of this work, as is a quantitative discussion of the *distances* between found clusterings. Section 2.6 gives an overview of the latter issue.

2.3.4.2 Computational Results

In this section we discuss the outcomes of our experiments. Since results for $n = 100$ largely agreed with those for $n = 1000$ we chose to focus on the latter (larger) setup here. Due to the equivalence of $\mathrm{L}^{-}_{\text{cov}}$ and $\mathrm{L}^{-}_{\text{perf}}$, we denoted results as L^{-}_{*}. The plots use *isolines* (or *contour lines*), which are curves where the evaluated function has a constant value as denoted by labels on the isolines in the figures. This is comparable to elevation contour lines on topographic maps, giving a good impression of the behavior of a function on two variables. We first conduct experiments that evaluate how well *lucidity* is in accordance with human intuition in terms of the quality of a given clustering, then we evaluate *lucidity*-based algorithms.

isolines

1. *Lucidity*-Scores on Generator and Benchmark Algorithms. We can assume that the graph generation process described above yields clusterings whose qualities—according to human intuition—clearly scale with p_{in} and (inversely) with p_{out}. Studying the quality measures we use on the pregenerated clusterings gives some useful insights about the behavior of these indices. Roughly speaking, for our results, one would expect high values in A_{strong}, with some variety of descent towards $A_{\text{rand.}}$. Figure 2.3.11 shows the results. As postulated for a reasonable index, all indices clearly attain the highest values for A_{strong}. For most indices, the slope of the quality level decreases with higher p_{out}; since the number of inter-cluster pairs of nodes increases more quickly than the number of intra-cluster nodes in our generator.[18] The slopes for *performance* remain approx. constant, which is a favorable behavior, as it yields a better comparability of clustering qualities of different graphs. This behavior is due to the fact that both edges inside, and non-edges between clusters are considered (as compared to, e.g., *coverage*). By Figure 2.3.11f $\mathrm{L}^{\div}_{\text{perf}}$ adopts this behavior, a fact that is not obvious from the definition, but a property to keep in mind when using the index. Conversely, L^{-}_{*} does not exhibit this behavior, which is in parts explained by its strong dependence on *coverage* (remember from Section 2.3.2.2 that in $\mathrm{L}^{-}_{\text{perf}}$, the terms referring to inter-cluster edges cancel out). As one difference between *coverage* and L^{-}_{*} note that the latter is more discriminative about $A_{\text{rand.}}$, yielding values close to 0. *Inter-cc* yields high values for $A_{\text{strong}}$and low values for $A_{\text{rand.}}$, consistent with the intuition. Again, slopes decrease, but for a different reason: The index *inter-cluster conductance* is sensitive to a large cut induced by a single small cluster; since the ratio of inter- to intra-cluster edges for A_{sparse} is lower than for A_{dense}, *inter-cluster conductance* generally yields higher values for A_{sparse}. Summarizing, all three implementations of *lucidity* behave consistently in this test on the quality of a "*ground-truth*" clustering with scalable clarity.

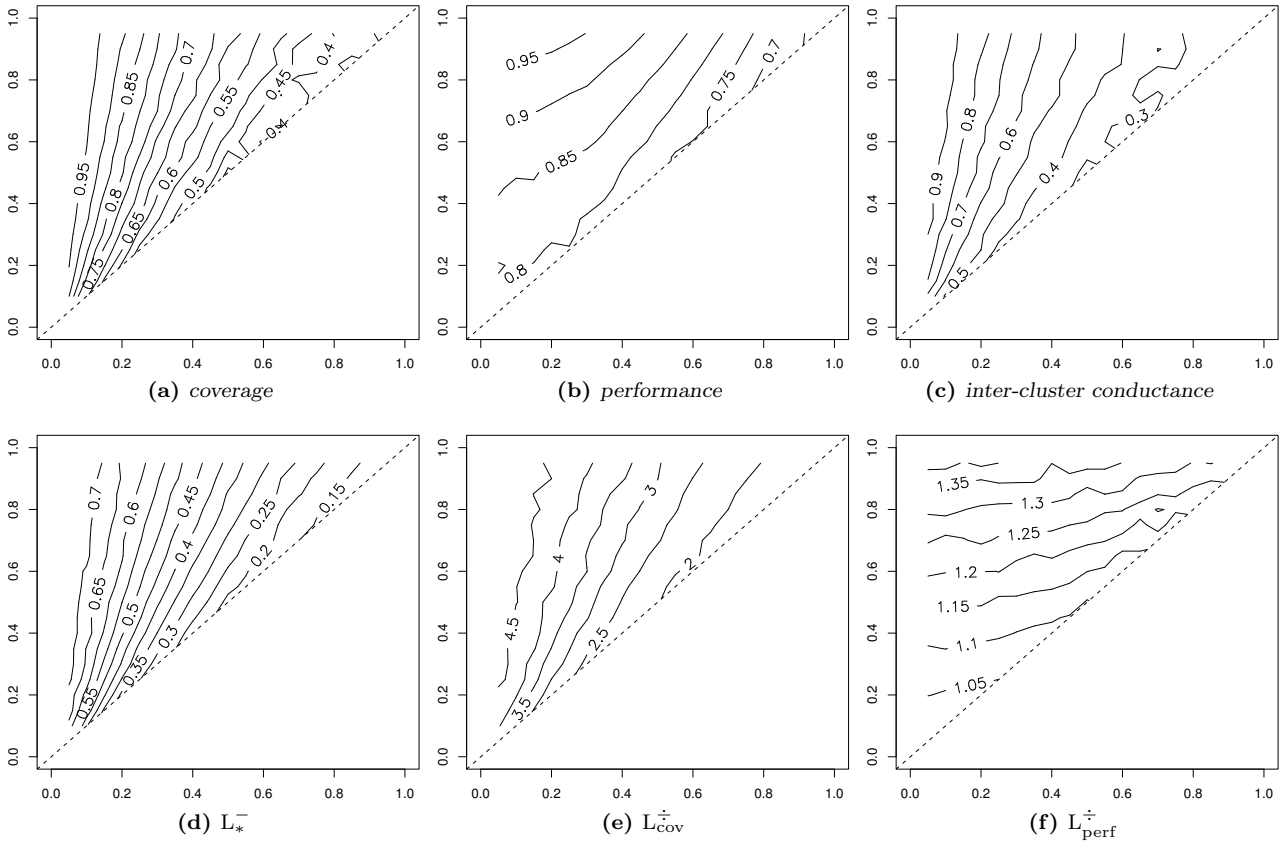
1. lucidity vs. human intuition

summary for gen.: lucidity sound

So far we know that our implementations of *lucidity* behave in a sound way on the generator's clustering. Figure 2.3.12 shows how they assess the results of the algorithms MCL and ICC (here we omit GMC for brevity), being less controlled experiments. For A_{strong} these

lucidity on bench. algorithms

[18] Roughly speaking, the ratio of intra- to inter-cluster edges is proportional to k. Thus, in our generator and in many real networks, the statement holds.



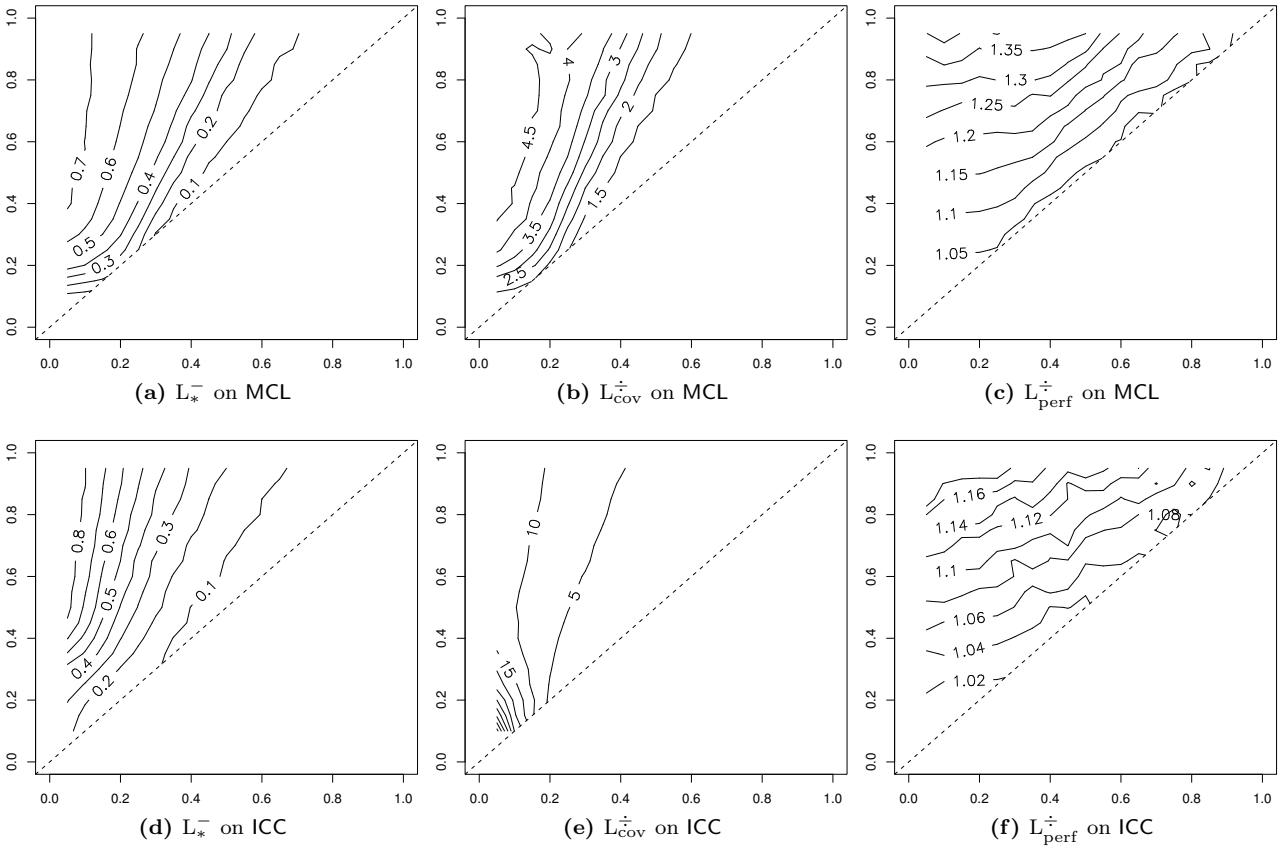
(a) *coverage* **(b)** *performance* **(c)** *inter-cluster conductance*

(d) L_*^- **(e)** $L_{\text{cov}}^{\div}$ **(f)** $L_{\text{perf}}^{\div}$

Figure 2.3.11. Plotted results for achieved quality on the underlying generator's clustering. The y-axis shows p_{in}, the x-axis shows p_{out}. The isolines indicate combinations of p_{in} and p_{out} where the same quality (value as label on isoline) has been measured.

algorithms probably identify a clustering which is very similar to the generator's. Comparing plots 2.3.12(a)-(c) to the corresponding ones in Figure 2.3.11 yields strong evidence for this. All three agree about MCL not performing very well for A_{sparse}, a fact *coverage*, *performance* and *inter-cluster conductance* also agree on (see Figures 2.3.13b, 2.3.14b and 2.3.15b, respectively). The main reason for this is that MCL tends to identify a very fine clustering for A_{sparse} (see Figure 2.3.16b). Interestingly, L_*^- sees worse quality in MCL's clusterings for A_{dense}, as opposed by $L_{\text{perf}}^{\div}$.[19] The reason is MCL's rather coarse clustering for that region, something we will see $L_{\text{perf}}^{\div}$approve of repeatedly below. To briefly discuss the results on ICC note that L_*^-'s values largely agree with those on the generator. Exhibiting a rather exotiv behavior, $L_{\text{cov}}^{\div}$ seems to approve of the generally rather fine clustering of ICC; note how the number of clusters of ICC (see Figure 2.3.16d) correlates with $L_{\text{cov}}^{\div}$, especially for A_{sparse}. The general shape of the values of $L_{\text{perf}}^{\div}$ strongly resembles that for the generator, a result all other measures (except $L_{\text{cov}}^{\div}$) second. However, it does so at a lower absolute level; we shall see the reason for this in Figure 2.3.16g, where it becomes obvious that in terms of the number of clusters to be found, this measure disagrees with the behavior of ICC, i.e., that $L_{\text{perf}}^{\div}$ favors coarse clusterings.

[19] Keeping crossreferences rigorous and multiply referring to other figures in almost each sentence massively obfuscates the text. We therefore refrain from most further references to plots in this section and hope that the reader manages to find the relevant ones.



(a) L^{-}_{*} on MCL **(b)** $L^{\div}_{\text{cov}}$ on MCL **(c)** $L^{\div}_{\text{perf}}$ on MCL

(d) L^{-}_{*} on ICC **(e)** $L^{\div}_{\text{cov}}$ on ICC **(f)** $L^{\div}_{\text{perf}}$ on ICC

Figure 2.3.12. Plotted results for achieved *lucidity* on MCL's and ICC's clusterings (for a comparison to other measures on these clusterings review Figures 2.3.13-2.3.16)

summary for question 1.

Summary for Question 1. To summarize our findings, we can state that all three implementations of *lucidity* behave very reasonably on the controlled, pregenerated clusterings, with the small asset for $L^{\div}_{\text{perf}}$ which seems to react to p_{in} and p_{out} in a largely independent manner. Our experiments on the two benchmark algorithms MCL and ICC partially second these results, but already suggest that $L^{\div}_{\text{perf}}$ favors coarse clusterings in a mild manner, and that $L^{\div}_{\text{cov}}$ rather wildly favors fine clusterings. In turn, L^{-}_{*} appears not to depend too strong on this, but instead mildly disfavors both extremes.

$L^{\div}_{\text{perf}}$ coarse, $L^{\div}_{\text{cov}}$ fine

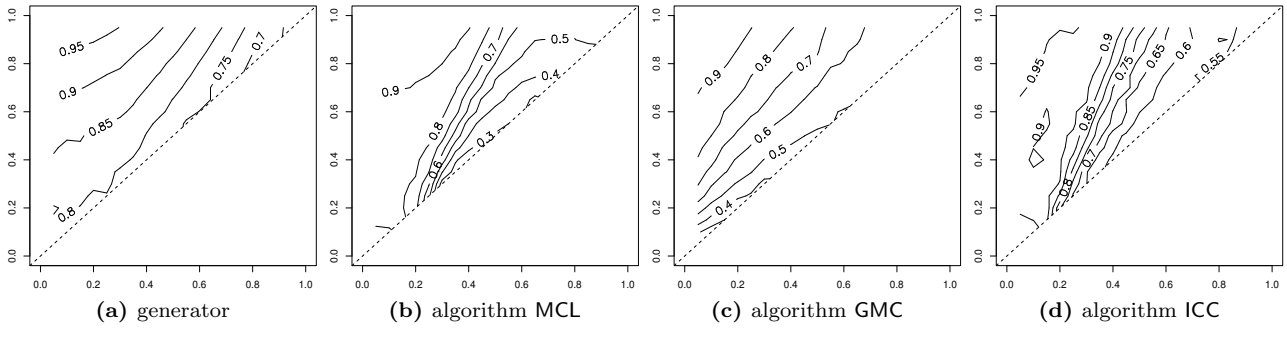
L^{-}_{*} decent

2. greedy lucidity

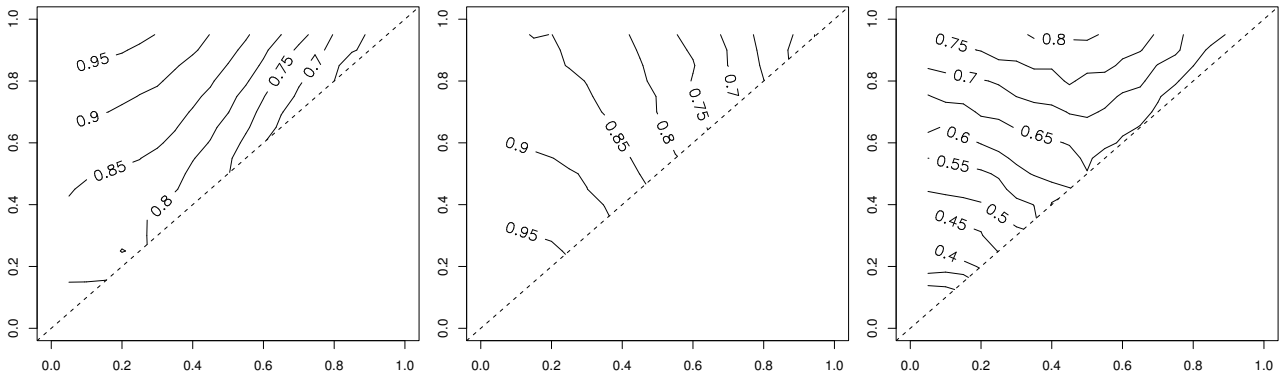
2. *Lucidity*-Based Algorithms In this section we measure the quality of clusterings identified with *lucidity*-based algorithms with three established indices and with *lucidity* itself. We thereby compare the results with those of three other algorithms which serve as benchmarks. Note that while a structural comparison with the generator's clustering is possible, it is not very meaningful, as this is not a "*ground-truth*" clustering in the traditional sense: we do not draw samples from an underlying distribution which is to be identified. Therefore it is possible (and for $A_{\text{rand.}}$ very probable), that the algorithms find *better* clusterings (in terms of quality) than the generator's. Moreover different clusterings on the same graph can yield the same objective quality in spite of heavily differing, structurally.

no ground truth for $A_{rand.}$

At a first glance, the statistical results for both relative variants ($L^{\div}_{\text{cov}}$ and $L^{\div}_{\text{perf}}$) and for L^{-}_{*} essentially differ for all three quality indices. Alongside the disagreement on the quality indices, $L^{\div}_{\text{cov}}$ tends to identify fine clusterings, i.e., 33 clusters on the average,while $L^{\div}_{\text{perf}}$ finds clusterings with a coarse granularity, i.e., 2.9 clusters on the average. The absolute variants exhibit a surprisingly similar behavior to the initial clustering with respect to all quality

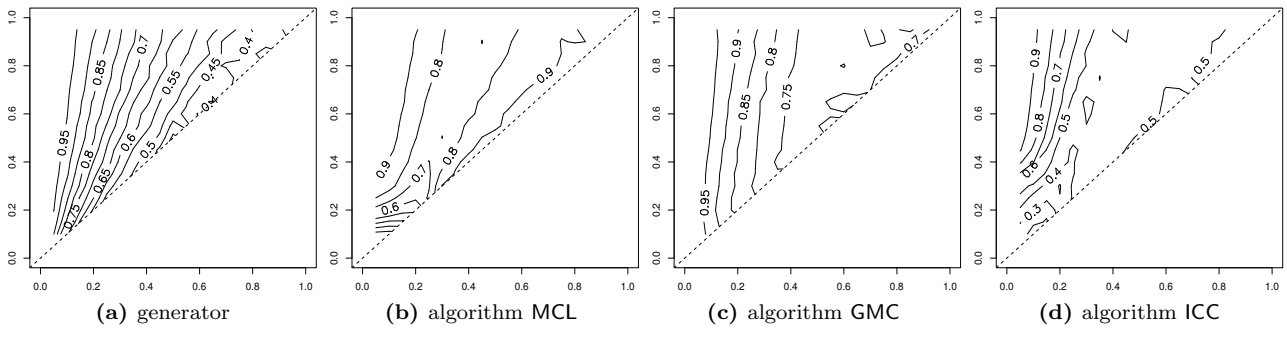


(a) generator **(b)** algorithm MCL **(c)** algorithm GMC **(d)** algorithm ICC

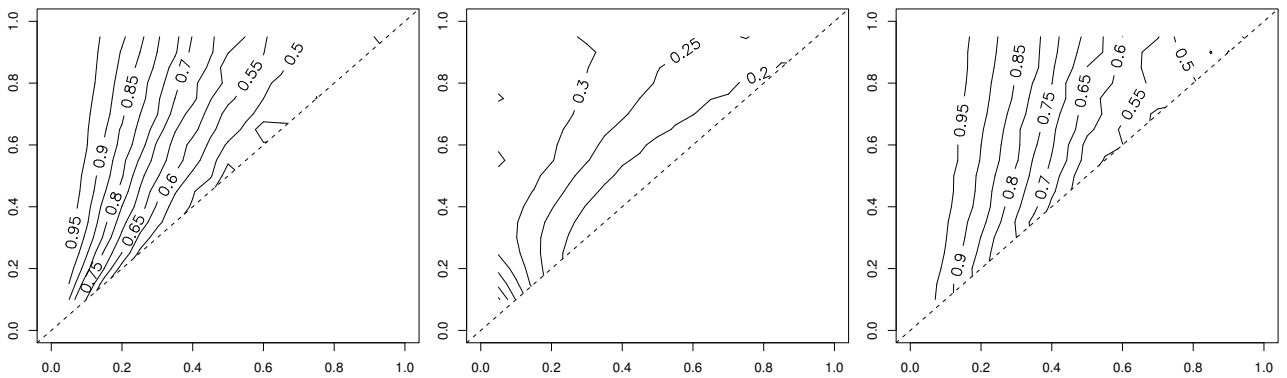


(e) greedy L_{*}^{-} **(f)** greedy $\mathrm{L}_{\mathrm{cov}}^{\div}$ **(g)** greedy $\mathrm{L}_{\mathrm{perf}}^{\div}$

Figure 2.3.13. Plotted results for values of *performance* achieved by the generator, the benchmark clustering algorithms and the *lucidity*-based algorithms

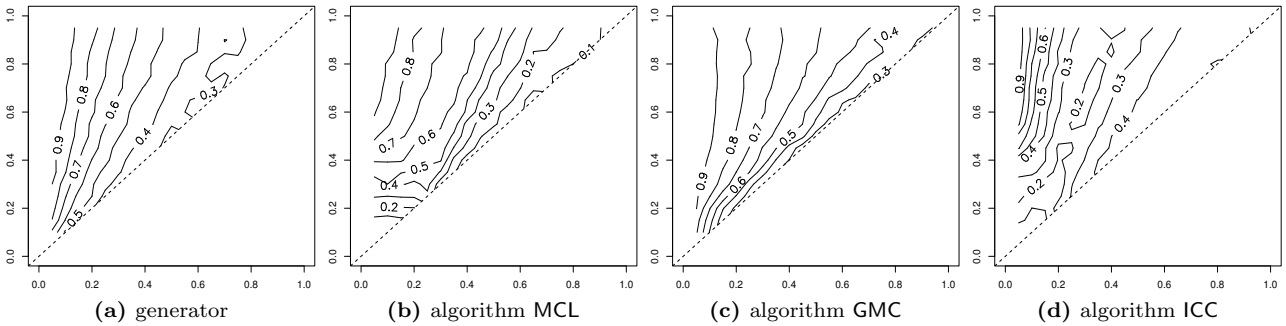


(a) generator **(b)** algorithm MCL **(c)** algorithm GMC **(d)** algorithm ICC

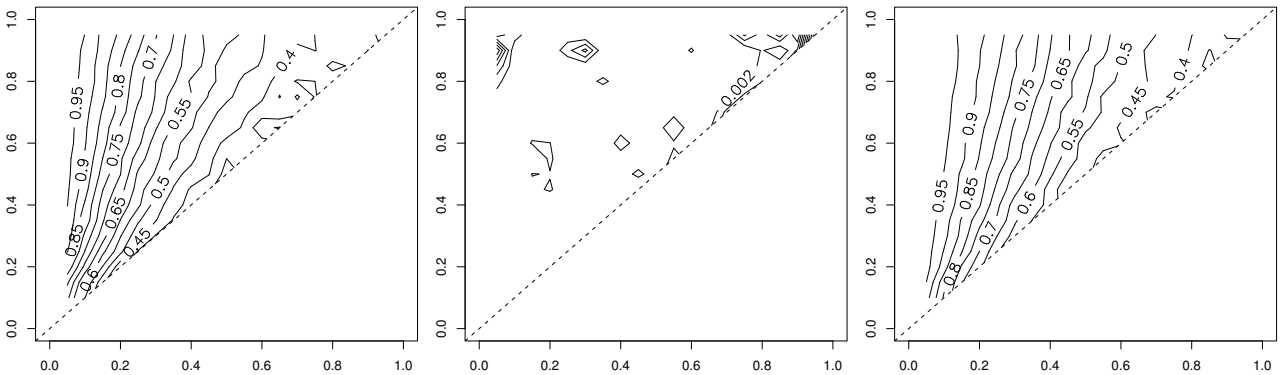


(e) greedy L_{*}^{-} **(f)** greedy $\mathrm{L}_{\mathrm{cov}}^{\div}$ **(g)** greedy $\mathrm{L}_{\mathrm{perf}}^{\div}$

Figure 2.3.14. Plotted results for values of *coverage* achieved by the generator, the benchmark clustering algorithms and the *lucidity*-based algorithms

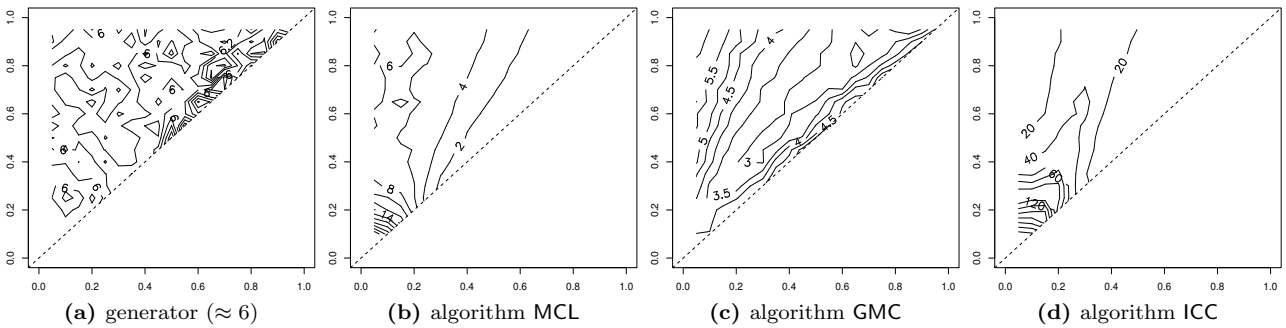


(a) generator **(b)** algorithm MCL **(c)** algorithm GMC **(d)** algorithm ICC

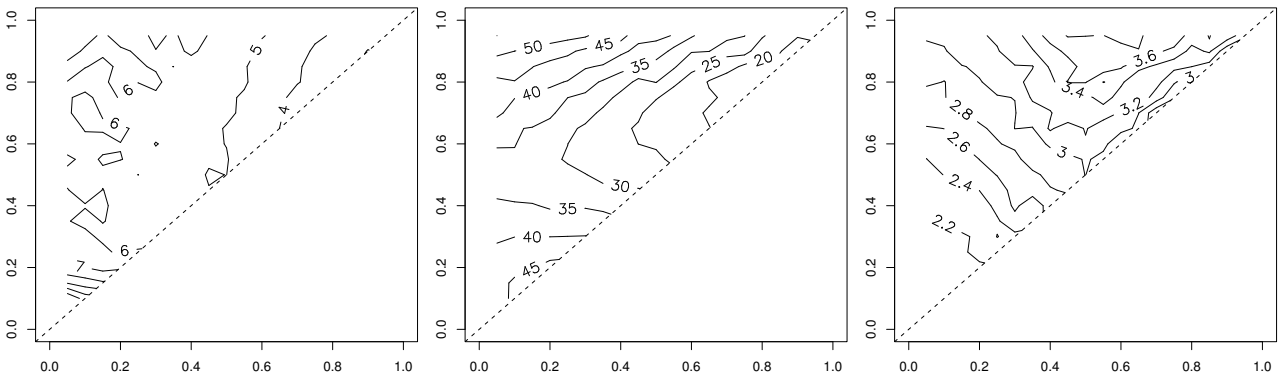


(e) greedy L_*^- **(f)** greedy $L_{\text{cov}}^{\div}$ **(g)** greedy $L_{\text{perf}}^{\div}$

Figure 2.3.15. Plotted results for values of *inter-cluster conductance* achieved by the generator, the benchmark clustering algorithms and the *lucidity*-based algorithms



(a) generator (≈ 6) **(b)** algorithm MCL **(c)** algorithm GMC **(d)** algorithm ICC



(e) greedy L_*^- **(f)** greedy $L_{\text{cov}}^{\div}$ **(g)** greedy $L_{\text{perf}}^{\div}$

Figure 2.3.16. Average number of clusters identified by the generator and the algorithms

indices. The same holds for $\mathrm{L}^{\div}_{\mathrm{perf}}$ with respect to *coverage* and *inter-cluster conductance*, however, the behavior is different for *performance*, but still acceptable scores are attained. In contrast, $\mathrm{L}^{\div}_{\mathrm{cov}}$ clearly fails to achieve high values of *coverage* and *inter-cluster conductance*, while its *performance* score is surprisingly good, a consequence of a very high number of clusters. The benchmark algorithms do not substantially surpass the initial clustering in general. Although the same holds for the *lucidity* algorithms, they shine for $A_{\mathrm{rand.}}$, finding higher quality clusterings than the generator (except $\mathrm{L}^{\div}_{\mathrm{perf}}$ for *coverage*).

lucidity-greedy shine for $A_{rand.}$

summary for question 2.

Summary for Question 2. In an overall assessment of the achieved clustering quality, the two absolute variants excel with respect to *performance* for almost all generated instances. This is particularly meaningful since both do not yield inappropriately high numbers of clusters, which would artificially increase *performance*. With respect to *coverage*, the absolute variants are only surpassed by the few algorithms that produced a substantially coarser clustering, among those $\mathrm{L}^{\div}_{\mathrm{perf}}$.

L^{-}_{*}-greedy yield high quality

An interesting observation is, that, using the *lucidity* measures as quality indices themselves, the greedy algorithms attain the maximum corresponding score for most testsets. However, in the case of A_{strong}, the obtained differences in the *lucidity* measures are small among most algorithms.

L^{y}_{x}-greedy best at L^{y}_{x}

comments on artifacts

Explaining Some Artifacts. The high values of *performance*, attained by $\mathrm{L}^{\div}_{\mathrm{cov}}$ for A_{sparse} are due to the fact that the large number of clusters identified by this algorithm yields a large fraction of non-connected pairs of nodes that are in different clusters. In turn, $\mathrm{L}^{\div}_{\mathrm{cov}}$ producing fine clusterings can be explained as follows. Each step of the algorithm increases *coverage* and $\mathbb{E}[\mathit{coverage}]$, which are both bounded by 1. These values increase faster, if an already large cluster is further enlarged. Thus, the fraction tends to 1 for coarse clusterings, causing the $\mathrm{L}^{\div}_{\mathrm{cov}}$-algorithm to terminate early, since, clearly, *coverage* is monotonic in $|\mathcal{C}|$.

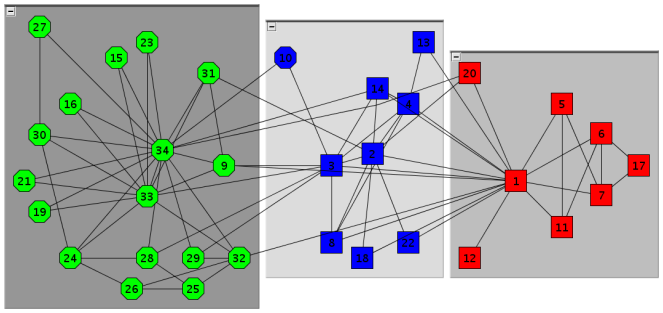
2.3.4.3 Real Data

Zachary's karate club

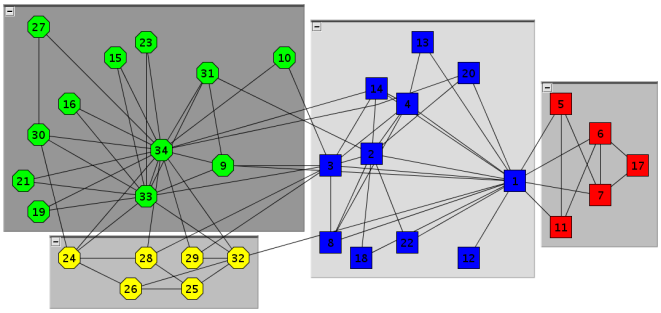
We have applied our algorithms to a number of real-world networks, due to limited space we only present two prominent ones. Figure 2.3.17 shows how the variants of *lucidity* perform on the *karate club* network, studied initially by [230]. The network represents friendship between the 34 members of a university club that, due to an internal dispute, split up into two groups (circular nodes on the left and square-shaped nodes on the right). Clearly, *relative performance lucidity* ($\mathrm{L}^{\div}_{\mathrm{perf}}$) excels here, exactly reproducing the original division and thereby surpassing even *modularity* in precision. Note that in cases the greedy algorithms have to break ties, different clusterings of the same input may occur. In particular this is the case for Figure 2.3.17a, which even yields the same value of $\mathrm{L}^{-}_{\mathrm{cov}}$ as the profoundly different clustering of the same network given in [173], but has been identified with the same (conceptually) algorithm.

Figure 2.3.18 shows an anonymized graph of the email contacts at our department over a period of three months (approx. 44300 emails). Nodes represent persons and weighted edges represent the number of email contacts between two coworkers. The grouping depicts the department's internal structure while the node colors (gray values) show the findings of community structure of the greedy algorithm based on L^{-}_{*}.[20] Since this example is based on the intuition that the graph structure reflects the grouping, we cleaned the network of artifact nodes with no links to their reference cluster (approx. 7.5% of the original nodes).

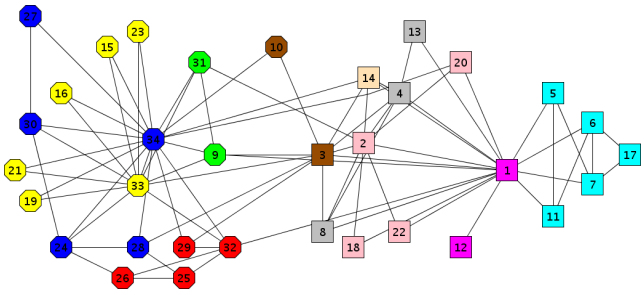
[20] See Section 5.1.1 for more details on this data set.



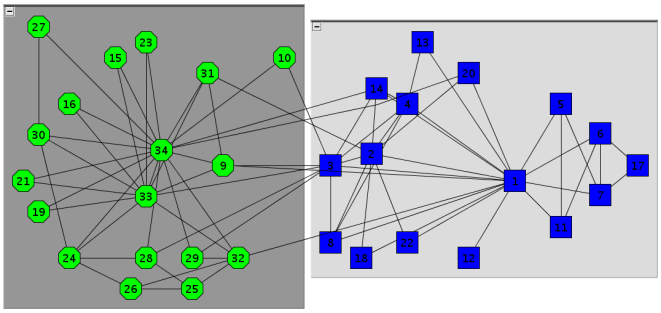
(a) L_*^- (*modularity*)



(b) ILP-optimized L_*^-



(c) $L_{\text{cov}}^{\div}$(boxes removed for readability)



(d) $L_{\text{perf}}^{\div}$

Figure 2.3.17. Results of the greedy *lucidity* algorithms on Zachary's karate club are in agreement with our experiments. The upper right figure additionally shows the clustering with optimum L_*^-, while in all figures node shapes denote the grouping in reality. While both the greedy and the ILP optimization of L_*^- are meaningful and close to the real grouping, *relative performance lucidity* (Subfigure 2.3.17d) yields a bisection which exactly reproduces the real grouping. The clustering $L_{\text{cov}}^{\div}$ identifies is not unreasonable, but too fine and insensitive for some applications.

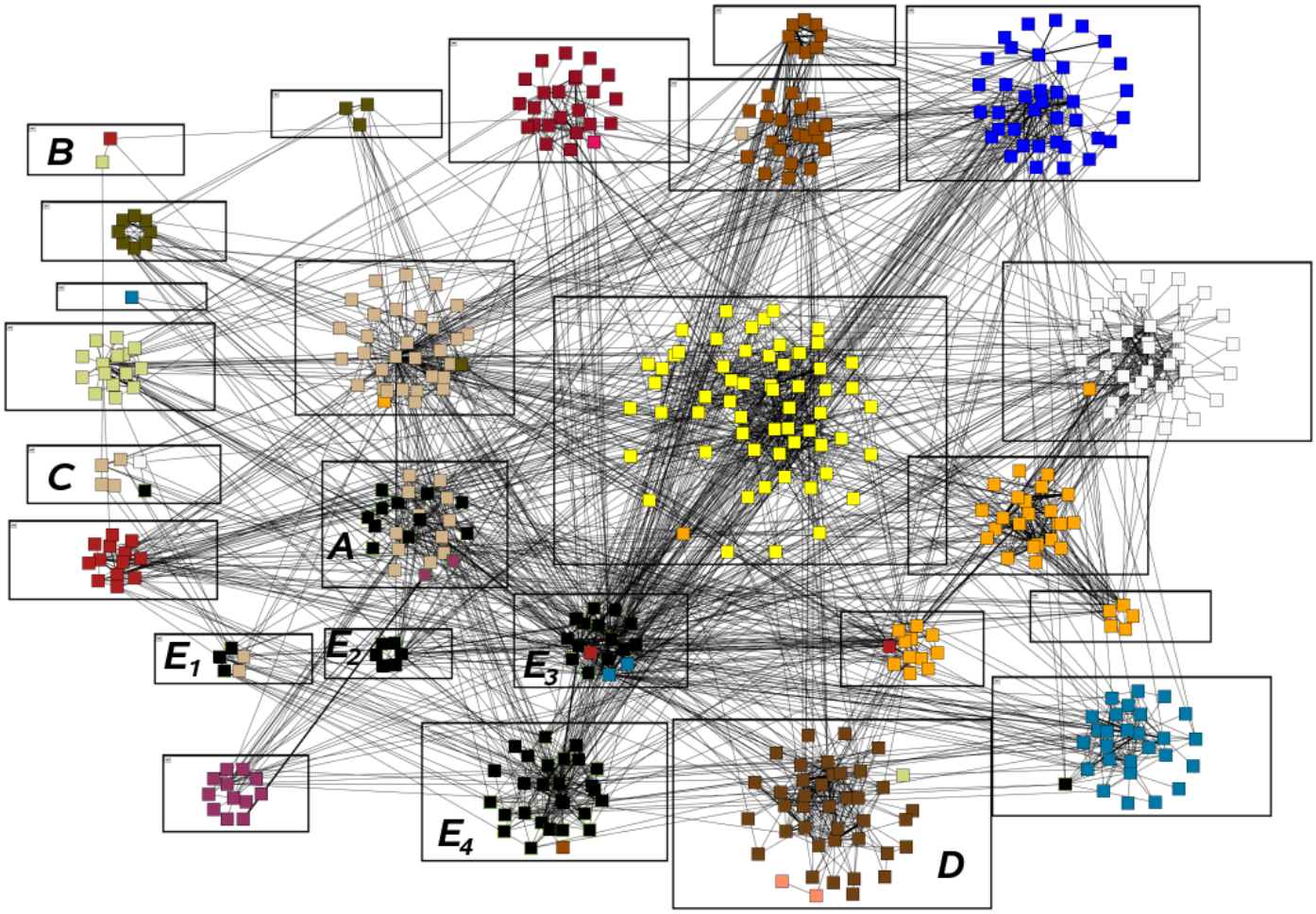


Figure 2.3.18. A network of email contacts at our department. The grouping depicts the department's internal structure as a reference, and the node colors (gray values) are the community detection result of L_*^-. Inside reference clusters, L_*^- misclassifies only 6.8% of nodes, most of which are due to the highly ambiguous reference cluster A, which is split in half by the algorithm. The clustering of L_*^- yields a noticeably higher ($\approx 6\%$) *coverage*, which is partly due to 9 clusters each being merged into other clusters they are strongly connected with. In terms of *inter-cluster conductance* and all four realizations of lucidity, L_*^- slightly surpasses the reference. However, the *performance* of the reference clustering is approx. 2.4% higher than that found by L_*^-. On the whole, a closer investigation explains most disagreements between the two clusterings, e.g., note the artifact nodes in clusters B, C, D and the strong connections between clusters $A, E_1, \ldots, E_4$, which account for the aggregation done by the algorithm. Please review Figure 2.6.6 for more details.

Section 2.4

ILPs for Graph Clustering

This is not a joke: They have moved the deadline to one full day earlier. You have less than two hours to submit.

(Daniel Delling, calling me at 10pm on the evening before the last day before the deadline for ISAAC'09. The organizers corrected their mistake at 11:17pm)

This small section tries to summarize a few insights into the formulation of ILPs for clustering tasks. We have already seen a feasible formulation in Section 2.2.2 alongside a objective function for *modularity*, which was later used to find the *modularity*-optimal clustering in example graphs; we briefly returned to that formulation in Section 2.3.2.3. However, there are a few obvious points that have not yet been covered. In particular, these are objective functions for other indices such as *performance* and *coverage* within the proposed framework of constraints, alternative sets of constraints and a few steps towards engineering an ILP for speed. In fact the results in this section were gathered in the context of *dynamic* graph clustering, but turned out to call for proper examination from a static point of view.

As a side note, a déjà vu that has become truly notorious among my colleagues, struck me during my work on possible ILP formulations for *modularity* optimization: Whatever clever ideas you put into an ILP formulation in order to reduce the set of variables, the set of constraints or the running time, the very first and most plain formulation will work best in the end. Of course this is partly due to sheer incompetence, but it is also a warning and a lesson I learned: Engineering an ILP for speed requires profound insights and superficial tweaks and even non-trivial ideas will probably not help immediately. The content of this section has not been published before.

Main Results

- We propose ILP formulations for clustering problems using *performance*, *modularity* and *coverage* as objective functions. The different setups model a node *equivalence relation*, a node *pseudometric* and a *node-cluster relation*, all allow for various side constraints. (Sections 2.4.1, 2.4.2 and 2.4.3, respectively)
- We report results on *modularity* optimization on well-known example networks, using basic tools for engineering ILPs. Despite our rather simple setup, to the best of my knowledge these results are the quickest running times for exact modularity optimization on these benchmark networks that have so far been reported in the literature. (Section 2.4.4)

Future Work. I strongly believe that there is potential in engineering an ILP for *modularity* optimization. On the one hand, I have seen many cases where variants of greedy maximization

yield very different results on the same graph instance, cases where naïve manual tuning improved *modularity*, and cases where a postprocessing driven by backtracking merges or by local optimization surprisingly fails or succeeds, which suggests that optimality can easily be more than just a stone's throw away. On the other hand, it is obvious that there are ideas the engineering presented in Section 2.4.4 leaves untouched, e.g., clever strategies for tagging constraints as *lazy*, *column generation* or *orbitopal fixing* [184]. Moreover a comparison in terms of quality to strategies like semi-definite programming as done, e.g., in [8] would be interesting.

2.4.1 *Equivalence Relation*

Viewing a clustering $\mathcal{C} = (C_1, \dots, C_k)$ as an *equivalence relation*, two nodes are equivalent if they belong to the same cluster. In turn every *equivalence relation* on a set of nodes induces a clustering, using the *equivalence classes* as clusters. Analogous to Section 2.2.2 we define for each pair $\{u, v\} \in \binom{V}{2}$ a binary decision variable X^{er}_{uv} [21] with the interpretation that $X^{\text{er}}_{uv} = 1$ if and only if u is equivalent to v and thus u and v belong to the same cluster. The set $\mathcal{X}^{\text{er}}$ of *node equivalence* variables thus are: $\mathcal{X}^{er}$

$$\mathcal{X}^{\text{er}}(V) := \{X^{\text{er}}_{uv} : \{u, v\} \in \binom{V}{2}\} \quad \text{with} \quad X^{\text{er}}_{uv} = \begin{cases} 1 & \text{if } \mathcal{C}(u) = \mathcal{C}(v) \\ 0 & \text{otherwise} \end{cases} . \tag{2.4.1}$$

We enforce consistency of these variables by modeling *transitivity* and *integrality*; to this end we add for each triple $\{u, v, w\}$ and for each pair $\{u, v\}$ the following linear constraints: *transitivity*

$$\underbrace{\forall \{u, v, w\} \in \binom{V}{3} : \begin{cases} X^{\text{er}}_{uv} + X^{\text{er}}_{vw} - X^{\text{er}}_{uw} \leq 1 \\ X^{\text{er}}_{uv} + X^{\text{er}}_{uw} - X^{\text{er}}_{vw} \leq 1 \\ X^{\text{er}}_{uw} + X^{\text{er}}_{vw} - X^{\text{er}}_{uv} \leq 1 \end{cases}}_{\textit{transitivity} \text{ constraints for } \mathcal{X}^{\text{er}}} , \quad \underbrace{\forall \{u, v\} \in \binom{V}{2} : X^{\text{er}}_{uv} \in \{0, 1\}}_{\textit{integrality} \text{ constraints for } \mathcal{X}^{\text{er}}} \tag{2.4.2}$$

This formulation (at least a very similar one) has already been used in [99] in order to calculate a clustering with maximum *performance*. It is easy to see that the other two properties of a sound *equivalence relation*, (i) *reflexivity* and (ii) *symmetry*, can be omitted in this context: (i) Since in a clustering a node is always inside a cluster, a reflexive variable X^{er}_{vv} always equals 1 and can thus be left out of any objective function (or set to 1). (ii) A objective function for clusterings must implicitly assume $X^{\text{er}}_{vu} = X^{\text{er}}_{uv}$, therefore only one such variable is needed. With the help of these these $\binom{n}{2}$ variables X^{er}_{vu} and $3\binom{n}{3}$ constraints (not counting integrality constraints) we can now list objective functions for *performance* (from [99]), *modularity* (see Equation 2.3.19), and *coverage*:

$$\begin{aligned} \textit{coverage}: \quad & \max \sum_{u<v \in V} \omega(u, v) \cdot X^{\text{er}}_{uv} \\ \textit{performance}: \quad & \max \sum_{u<v \in V} (2 \cdot A(u, v) - 1) \cdot X^{\text{er}}_{uv} \\ \textit{modularity}: \quad & \max \sum_{u<v \in V} \left(\omega(u, v) - \frac{\omega(u) \cdot \omega(v)}{2 \cdot W} \right) \cdot X^{\text{er}}_{uv} \end{aligned} \tag{2.4.3}$$

In these equations $A(u, v)$ is a binary constant with $A(u, v) = 1$ if and only if $\{u, v\} \in E$. Note that *coverage* (trivial) and *modularity* (see Sec. 2.3.2.3) can immediately be formulated in a weighted context, however we avoid the trouble for *performance*, which is, in its simple

[21] There is only one such variable for $\{u, v\}$, and $X^{\text{er}}_{uv} = X^{\text{er}}_{vu}$ is the same variable just written differently.

and unweighted form, derived as follows:

$$\begin{aligned} \mathit{performance}_{\mathrm{ILP}} &= \sum_{u<v} \left(A(u,v) \cdot X^{\mathrm{er}}_{uv} + (1 - A(u,v)) \cdot (1 - X^{\mathrm{er}}_{uv})\right) \\ &= \sum_{u<v} \left(A(u,v) \cdot X^{\mathrm{er}}_{uv} + 1 - X^{\mathrm{er}}_{uv} - A(u,v) + A(u,v) \cdot X^{\mathrm{er}}_{uv}\right) \\ &= \sum_{u<v} (2 \cdot A(u,v) - 1) \cdot X^{\mathrm{er}}_{uv} + \text{constant} \end{aligned}$$

2.4.2 *Pseudometric*

A similar idea has been used in [54] to solve the problem of *correlation clustering*: However, the authors used the same set of variables[22], yet reversed their interpretation, i. e., $X^{\mathrm{pm}}_{uv} = 0$ if and only if u is equivalent to v and thus u and v belong to the same cluster. Similar to the above formulation, and yielding the same size of the ILP, we now get *node distance variables* $\mathcal{X}^{\mathrm{pm}}$:

$\mathcal{X}^{pm}$

$$\mathcal{X}^{\mathrm{pm}}(V) := \{X^{\mathrm{pm}}_{uv} : \{u,v\} \in \binom{V}{2}\} \quad \text{with} \quad X^{\mathrm{pm}}_{uv} = \begin{cases} 0 & \text{if } \mathcal{C}(u) = \mathcal{C}(v) \\ 1 & \text{otherwise} \end{cases} \tag{2.4.4}$$

$$\underbrace{\forall \{u,v,w\} \in \binom{V}{3} : \begin{cases} X^{\mathrm{pm}}_{uv} + X^{\mathrm{pm}}_{vw} - X^{\mathrm{pm}}_{uw} \geq 0 \\ X^{\mathrm{pm}}_{uv} + X^{\mathrm{pm}}_{uw} - X^{\mathrm{pm}}_{vw} \geq 0 \\ X^{\mathrm{pm}}_{uw} + X^{\mathrm{pm}}_{vw} - X^{\mathrm{pm}}_{uv} \geq 0 \end{cases}}_{\textit{triangle inequality} \text{ constraints for } \mathcal{X}^{\mathrm{pm}}} , \quad \underbrace{\forall \{u,v\} \in \binom{V}{2} : X^{\mathrm{pm}}_{uv} \in \{0,1\}}_{\textit{integrality} \text{ constraints for } \mathcal{X}^{\mathrm{pm}}} \tag{2.4.5}$$

Note that X^{pm} can be interpreted as a *pseudometric* over $\{0,1\}$, where two nodes have distance 0 if and only if they belong to the same cluster and the constraints modeled by Equation 2.4.5 represent the *triangle inequality*. Every program using the above *equivalence relation* model can be transformed into the *pseudometric* system by replacing each variable X^{er}_{uv} by $(1 - X^{\mathrm{pm}}_{uv})$ in every constraint and in the objective function and vice versa. The objective functions that are induced by the quality indices are:

triangle inequality

$X^{er}_{uv} \doteq (1 - X^{pm}_{uv})$

$$\begin{aligned} \textit{coverage}\colon & \min \sum_{u<v\in V} \omega(u,v) \cdot X^{\mathrm{pm}}_{uv} \\ \textit{performance}\colon & \min \sum_{u<v\in V} (2 \cdot A(u,v) - 1) \cdot X^{\mathrm{pm}}_{uv} \\ \textit{modularity}\colon & \min \sum_{u<v\in V} \left(\omega(u,v) - \frac{\omega(u) \cdot \omega(v)}{2 \cdot W}\right) \cdot X^{\mathrm{pm}}_{uv} \end{aligned} \tag{2.4.6}$$

2.4.3 Node-Cluster Relations

The above programs for an *equivalence relation* and *pseudometric* can be extended in order to describe *node-cluster relations*. We defined an additional set $\mathcal{Y}^{\mathrm{er}}$ of n^2 binary variables Y^{er}_{uj} with $u \in V$ and $1 \leq j \leq n$. This can be thought of as reserving n empty clusters beforehand, represented by a dummy node j. With this in mind we can proceed exactly as above—but we shall stick to an *equivalence relation* for brevity. The interpretation thus is that $Y^{\mathrm{er}}_{uj} = 1$ if and only if node u belongs to cluster C_j. Clusters that ultimately end up empty are simply ignored. As with $\mathcal{X}^{\mathrm{er}}$ we need to couple this set $\mathcal{Y}^{\mathrm{er}}$ via transitivity constraints: if $X^{\mathrm{er}}_{uv} = 1$, then both u and v must belong to the same cluster, i.e. $Y^{\mathrm{er}}_{uj} = 1$ and $Y^{\mathrm{er}}_{vj} = 1$ for some j. We can avoid requiring full *transitivity* among nodes by enforcing that no node tries to belong to several clusters, i.e., the *uniqueness* constraints in Equation 2.4.9 replace the *transitivity*

$\mathcal{Y}^{er}$

uniqueness constraints

[22] We just renamed the variables for the sake of tractability.

constraints in Equation 2.4.2. This yields :

$$\mathcal{Y}^{\mathrm{er}}(V) := \{Y^{\mathrm{er}}_{uj} : \{u,j\} \in V \times [1,\ldots,n]\} \quad \text{with} \quad Y^{\mathrm{er}}_{uj} = \begin{cases} 0 & \text{if } \mathcal{C}(u) \neq C_j \\ 1 & \text{if } \mathcal{C}(u) = C_j \end{cases} \tag{2.4.7}$$

$$\underbrace{\forall \{u,v,j\} \in \binom{V}{2} \times [1,\ldots,n] : \begin{cases} Y^{\mathrm{er}}_{uj} + Y^{\mathrm{er}}_{vj} - X^{\mathrm{er}}_{uv} \leq 1 \\ X^{\mathrm{er}}_{uv} + Y^{\mathrm{er}}_{uj} - Y^{\mathrm{er}}_{vj} \leq 1 \\ X^{\mathrm{er}}_{uv} + Y^{\mathrm{er}}_{vj} - Y^{\mathrm{er}}_{uj} \leq 1 \end{cases}}_{\textit{transitivity} \text{ of } \mathcal{X}^{\mathrm{er}} \text{ via one element of } [1,\ldots,n]} \tag{2.4.8}$$

$$\underbrace{\forall \{u,j\} \in V \times [1,\ldots,n] : Y^{\mathrm{er}}_{uj} \in \{0,1\}}_{\textit{integrality} \text{ constraints for } \mathcal{Y}^{\mathrm{er}}} \qquad \underbrace{\sum_{i=1}^{n} Y^{\mathrm{er}}_{ui} = 1}_{\textit{uniqueness} \text{ constraints for } \mathcal{Y}^{\mathrm{er}}} \tag{2.4.9}$$

As the objective functions for the programs using the *equivalence* relation or the *pseudometric* formulation solely depend on X^{er} and X^{pm}, respectively, they directly carry over. These, slightly more, $\binom{n}{2}+nk$ boolean variables $\mathcal{X}^{\mathrm{er}}$ and $\mathcal{Y}^{\mathrm{er}}$ use a set of $3n\binom{n}{2}+n$ constraints (again neglecting integrality constraints). However, we can now set the maximum number of allowed clusters to some number k, such that $[1,\ldots,n]$ as used above now is some smaller interval $[1,\ldots,k]$ (see Section 2.4.6 below). This yields $3k\binom{n}{2}+k$ constraints which is potentially much less than those constraints required by Equation 2.4.2. Then, however $k \in o(n)$ must be a prerequisite, implying that a proper upper bound on the number of nonempty clusters must be provided.

2.4.4 Engineering the ILP

Although it is not a central point of this work, we briefly sketch our findings on engineering the above ILP formulations for the purpose of optimizing *modularity*. The instances used for these experiments are frequently used in the related literature, we refer the reader to [44] for further information and references. We used two different ILP solvers, the free solver package *lp_solve* [2] (version 5.5.0.10) and the commercial solver package *CPLEX* [1] (version 11.1). Starting out with the solver lp_solve and the node *equivalence* formulation (Section 2.4.1), we investigated what speedups can be achieved. Table 2.4.1 summarizes our findings. Variants 1-4 use node *equivalence* or *pseudometric* formulations on both solvers. As any constellation of clusters will yield a large number of redundant constraints, it might serve to declare one equation per triple to be *lazy*[23], yielding variant 5. Variant 6 uses *node-cluster* constraints and 7 again *lazyness*. As *node-cluster* constraints introduce symmetry, we tried, in variant 8, to break some of it by enforcing $v_i \in C_\ell \Rightarrow i \geq \ell$ ($n^2/2$ add. constraints), see [184, 144] for details. Finally, in variant 9, we added to the *node-cluster* constraints *user cuts*[24] given by node distance constraints. We omit further combinations of the above, since they did not yield significant insights.

lp_solve vs. CPLEX

lazy constraints

user cuts

Note that we did not perform the above tests to any degree of statistical significance, but with 5 averaged repetitions with manually excluded outliers only. Using more sophisticated ideas from [184, 144], further speedups might be possible using *column generation* or *orbitopal fixing*, but still, for our task, symmetry removal cannot avoid enforcing transitivity amongst all nodes. A deeper strategy for identifying suitable *lazy* constraints based on node distance might be another option. Summarizing, a simple node *pseudometric* formulation clearly seems best, with CPLEX generally having an advantage over lp_solve, except for the surprisingly quick Football run. Our tests were executed on one core of an AMD Opteron 2218 running

a simple form. worked best

[23] A *lazy* constraint is a necessary constraint but is only included into the problem description, if the solver finds a solution that otherwise conflicts with the *lazy* constraint.

[24] A user cut is a redundant constraint that the solver can choose to include in order to find a better LP relaxation.

Table 2.4.1. Running times of variant ILP formulations. Experiments were aborted (*dnf*) in case running times exceeded 24h. The top three lines give general information on the networks and the *modularity*-optimal clustering.

	Peterson	Zachary	Chesapeake	Dolphins	LesMis	Polbooks	Football	
$\|V\|/\|E\|$	10/15	34/78	36/122	62/159	77/254	105/441	115/616	
$\|\mathcal{C}\|$	2	4	4	5	8	6	10	
modularity	0.167	0.42	0.349	0.529	0.542	0.523	0.606	
Var. 1	0.48s	2h 21m	dnf	dnf	dnf	dnf	dnf	lp_solve, *node equivalence*
Var. 2	0.1s	1.74s	3.45s	1m 32s	3m.36s	54m 42s	1h 34m	lp_solve, *node distance*
Var. 3	0.35s	1.4s	0.52s	57.7s	22.6s	6m 15s	6h 44m	CPLEX, *node equivalence*
Var. 4	0.03s	1.17s	0.741s	1m 34s	20.45s	10m 09s	5h 54m	CPLEX, *node distance*
Var. 5	0.033s	3.18s	4.59s	4m 57s	7m 18s	2h 35m	1d 02h	CPLEX, *node distance*, *lazy*
Var. 6	1m 54s	dnf	dnf	dnf	dnf	dnf	dnf	CPLEX, *node-cluster*
Var. 7	7m 30s	dnf	dnf	dnf	dnf	dnf	dnf	CPLEX, *node-cluster*, *lazy*
Var. 8	2.4s	dnf	4h 32m	dnf	dnf	dnf	dnf	CPLEX, *n.-cl.*, break symm.
Var. 9	0.53s	44.3s	1m 32s	9h 54m	1d 4h	dnf	dnf	CPLEX, *n.-cl.*, user cuts

SUSE Linux 10.3. The machine is clocked at 2.6 GHz, has 32 GB of RAM and 2 x 1 MB of L2 cache. The programs were run on Java version 1.6.0_04.

2.4.5 Edge-Cluster and Node-Cluster Relations

edge-cluster relations

An alternative integer linear program using only variables which encode *node-cluster* and *edge-cluster relations* has been proposed by Xu et al. [229]. Although the reported exemplary running times are slower than ours, this approach is an interesting variant. The authors introduced variables Y_{uk} for nodes $u \in V$ and indices $1 \leq k \leq n$ and Z_{ek} for edges $e \in E$ and indices $1 \leq k \leq n$. The interpretation is that $Y_{uk} = 1$ if and only if node u belongs to cluster k; and $Z_{ek} = 1$ if and only if the edge e has both its endpoints inside cluster k. In order to ensure that the Y- and Z-variables are consistent, the following linear constraints are added to the *uniqueness* constraints as above:[25]

$$\forall\, 1 \leq k \leq n\ \forall\, \{u,v\} \in E\colon \begin{cases} 2 \cdot Z_{\{uv\},k} \leq Y_{uk} + Y_{vk} \\ Z_{\{uv\},k} \geq Y_{uk} + Y_{vk} - 1 \end{cases} \tag{2.4.10}$$

In order to express the above objective functions, we additionally need variables storing the numbers of nodes, edges or the sum of degrees inside the clusters. We define S_k^{node}, S_k^{edge} and S_k^{deg} as:

$$S_k^{\text{node}} := \sum_{u \in V} Y_{uk} \quad , \quad S_k^{\text{edge}} := \sum_{e \in E} \omega(e) \cdot Z_{ek} \quad \text{, and} \quad S_k^{\text{deg}} := \sum_{u \in V} \deg(u) \cdot Y_{uk} \quad .$$

Objective functions equivalent to the quality indices above can be written as follows:

$$\begin{aligned} \text{coverage:} \quad & \max \sum_{1 \leq k \leq n} S_k^{\text{edge}} \\ \text{performance:} \quad & \max \sum_{1 \leq k \leq n} \left(2 \cdot \sum_{e \in E} Z_{ek} - \left(S_k^{\text{node}}\right)^2 \right) \\ \text{modularity:} \quad & \max \sum_{1 \leq k \leq n} \left(2 \cdot \omega(E) \cdot S_k^{\text{edge}} - \left(S_k^{\text{deg}}\right)^2 \right) \quad . \end{aligned}$$

[25] In fact, the authors of [229] do not mention the second constraint, but it is necessary.

2.4.6 Side Constraints

From the myriad reasonable side constraints the above formulations allow, we just point out that lower or upper bounds on the number or sizes of clusters can easily be realized: A lower bound ℓ_n and an upper bound u_n on the number of clusters can be expressed by the constraints given in Equation 2.4.11; note that simply stating an upper bound is more easily done as mentioned at the end of Section 2.4.3.

$\ell_n \leq |\mathcal{C}| \leq u_n$

$$\forall\, 1 \leq j \leq \ell_n \colon \sum_{u \in V} Y^{\mathrm{er}}_{uj} \geq 1 \qquad \text{and} \qquad \forall\, u_n < j \leq n \colon \sum_{u \in V} Y^{\mathrm{er}}_{uj} = 0 \tag{2.4.11}$$

In order to force each of a collection of k clusters to contain at least ℓ_s and at most u_s elements, the linear constraint given in Equation 2.4.12 for *node-cluster relations*.

$\ell_s \leq |C| \leq u_s$

$$\forall 1 \leq j \leq k \colon \ell_s \leq \sum_{v \in V} Y^{\mathrm{er}}_{vj} \leq u_s \tag{2.4.12}$$

We can also enforce these size constraints for all (non-empty) clusters without the use of $\mathcal{Y}^{\mathrm{er}}$, just using either $\mathcal{X}^{\mathrm{er}}$ (Equation 2.4.13) or $\mathcal{X}^{\mathrm{pm}}$ (Equation 2.4.14), by simply bounding the maximum and the minimum number of other nodes a node u can sit together with in a cluster:

$$\forall\, u \in V \colon \ell_s \leq \sum_{v \neq u} X^{\mathrm{er}}_{uv} + 1 \leq u_s \tag{2.4.13}$$

$$\forall\, u \in V \colon \ell_s \leq (n-1) - \sum_{v \neq u} X^{\mathrm{pm}}_{uv} + 1 \leq u_s \tag{2.4.14}$$

Note that the annoying 1 in these Equations is due to the missing reflexive variable X^{er}_{uu}. In all the above cases, at most n additional constraints are necessary.

Section 2.5

Orca

In many ways I'm the burden
that divides us from the light
In many ways you're the halo
that keeps my spirit alive

(The Chosen Pessimist,
In Flames)

Races exist in many fields of computer science. Among these are races for *speed* such as the quickest average answer times to shortest path queries, races for storage space such as the best *compression* factor for a database of 1TB of random text, and of course, as seen in the preceeding sections, races for the best achieved *quality* in terms of some task. While sometimes such races seem to depart from any challenge faced in reality, they are more than often a decisive driving force behind progress. In this section we describe our partaking in the race for clustering huge graphs.

During the last years, a wide range of huge networks has been made available to researchers. In the exploration and the analysis of networks such as the World Wide Web, social and natural networks and recommendation systems or protein dependencies, *graph clustering* has become a valuable tool. Thus, clustering algorithms that can cope with huge graphs and yield a good clustering in a reasonable timeframe are desirable. However, we have to adjust our outlook onto graph clustering. In spite of technical advances, such as computational puissance and fast storage media, instances of the size of several millions of nodes still pose algorithmic challenges, and render techniques that are successfully applied on smaller problems infeasible. For the design of a clustering algorithm for huge networks, the emphasis must be on the feasibility of applying the algorithm on such problem instances in practice, i.e., both space and time consumption must be practicable. We generally advocate that modern clustering problems will hardly breach the limits of the main memory's size of modern server hardware, as the latter is subject to a race which seems to advance with a speed at least equal to that of the size of clustering instances. We thus aim at an algorithm which, given the instance fits into the main memory of a server, can also solve the instance in the main memory in reasonable time, i.e., within hours, if the worst comes to the worst. With these primary design goals at hand, otherwise crucial goals such as the quality of a clustering must become secondary, but certainly must not forego much importance.

In this section we present the Orca *reduction and contraction algorithm*, a locally operating, fast graph clustering algorithm, which is capable of handling huge instances that state-of-the-art methods cannot cope with. Orca is able to cluster inputs with hundreds of millions of edges in less than 2.5 hours, identifying clusterings with measurably high quality. Orca is designed to rely on simple structural observations that immediately translate to *intra-cluster density and inter-cluster sparsity*, while *avoiding* the direct maximization of some index. In fact, our approach explicitly avoids maximizing any single index value such as *modularity*, but instead relies on simple and sound structural operations (we explain the

reasons below). We evaluate the performance of Orca with respect to running time and several quality measures for clusterings on a number of publicly available networks and compare it to other graph clustering algorithms. Unlike most previous approaches, Orca works in a local sense: it iteratively contracts *dense regions* to *super-nodes* which become the clustering of the current iteration step.

After our ideas for a fast and locally operating clustering algorithm lay about for one year without anybody finding time for it, we found a good student, Christian Schulz, to get things started. Then, during the final weeks of our work on Orca, I attended a workshop on the "Detection and visualization of communities in large complex networks" at UCL in Louvain-la-Neuve, Belgium. Apart from the fact that this was probably the most worthwhile conference I attended so far,[26] I there learned about a recently devised serious competitor for Orca (see below). It even followed a similar approach! The lesson I learned was that you should not let good ideas lay about for one year, you might miss the chance to be the first to enter an untrodden field. Parts of this work have previously been published in [72], based on joint work with Daniel Delling, Christian Schulz and Dorothea Wagner.

Main Results

- We design and describe Orca, a state-of-the-art ultra-fast graph clustering algorithm, based on contraction operations that are driven by local density instead of the maximization of one particular quality index. (Section 2.5.1)
- We systematically determine feasible values for the two parameters Orca can be tuned by, and cluster two benchmark graphs. (Section 2.5.2)
- In an experimental evaluation on huge networks, Orca outperforms all but one competitors. Only a local variant of greedy agglomeration [38] competes with Orca in terms of feasibility. While it is faster on the whole, the scalability of our approach seems better. (Section 2.5.3)
- In terms of quality, Orca and its competitor [38] compete with other state-of-the-art algorithms (given the latter finish on an instance). Between them, no general assertion which one to prefer can be made. For huge instances the choice ultimately depends on the application and whether artifacts specific to *modularity* are acceptable or even desired. If the answer is yes, one can follow [38], otherwise Orca is the better choice. (Section 2.5.3)

Related Work. In order to facilitate a better positioning of this section into contemporary literature on the topic, we recall the important pieces of related work from Section 2.1.2. Provably good methods for graph clustering (e.g., [48, 87, 213]) by far cannot handle instances of huge size, let alone algorithms for solving NP-hard optimization problems such as *modularity* optimization. Even quite a few heuristic approaches do not suggest themselves for high speed-ups, e.g., the iterative removal of central edges [178], or the direct identification of dense subgraphs [76]. Immediate candidates, however, are variants of greedy agglomeration, since agglomeration criteria tend to behave far more stable than, e.g., centrality measures, and can thus be computed more "superficially". We will include two global such approaches in our evaluation, the *walktrap* [187], and the greedy maximization of *modularity* as discussed in 2.2.5 and 2.3.3, as these are the only ones with a fighting chance to keep up with Orca's low running times. However, we shall see that Orca's sole true competitor is [38] in terms of feasibility on huge networks. It is worth stressing again that this approach is similar to Orca, on a rough scale. However, while this technique is explicitly designed to maximize *modularity*—which it achieves quite well—and thus solely relies on one measure as the single criterion, Orca builds a clustering without this bias towards *modularity*. Although *modularity* has proven to be a rather reliable quality measure, it is known to behave artificially

whom to compare to?

[26] A thank-you to Marco Gaertler for pointing me at that workshop.

to some extent. On top of that, we shall see in Section 2.5.3 that in some cases, the greedy strategy terribly fails to achieve its aim.

potential bias from index-maximization

Why not be Content with Maximizing Good Indices? The pressing reason for this is that any known quality index can be tricked, i.e., exhibits pathological behavior in certain situations. While this may be neglected for most reasonably modeled instances, a subtle trend of index-maximized clusterings towards their specific behavior cannot be ignored, as has been shown in [90] and in Section 2.2.3.1 for *modularity*, and in Section 1.2.2 for a number of indices. Specifically, *modularity*'s "compulsion" to produce balanced clusters can be a severe weakness that must be kept in mind. In turn, this does not mean that established quality measures should not be used; it is certainly reasonable to largely rely on them when evaluating and even identifying clusterings; however, targeted maximization of one index harbors the risk that on graphs which do not comply with the bias of that index, strange results might occur. Thus, ORCA has been designed to work on simple structural observations that immediately translate to *intra-cluster density and inter-cluster sparsity*, avoiding direct index maximization. In particular (and in contrast to *modularity*) ORCA's routines do not enforce balanced clusters, and do not have local clustering decisions rely on overall graph properties such as the average degree.

heuristics can get very bad

As an interesting side note, insights from previous sections suggest that the risk for maximization heuristics to deviate from optimality on a grand scale increases on larger networks. We shall see this corroborated by the results of the algorithm from [38], which on some few instances delivers astonishingly bad *modularity* scores (Section 2.5.3, especially on the instance uk-2002).

advantages of local approaches

Making a Case for Local Methods. Many widespread clustering algorithms iterate some global mechanism a linear number of times, which is particularly typical for classic bottom-up agglomerative approaches (e.g., *greedy index maximization* Section 2.2.5 or the *walktrap* [187]), or they include some direct technique that is both time and space consuming (e.g., *global Markov chains* [213] or *iterative conductance cutting* [146]). Operating locally in graphs avoids these issues, if local operations are simple and bounded in number. Apart from this and their obvious eligibility for parallelization, more facts encourage local approaches. First, heuristics that maximize a clustering quality index are known to exhibit *scaling behavior*, an effect which local methods might be able avoid, if they do not strongly rely on global properties of the graph. Second, a limited set of local operations on a graph, e.g., iterating over incident edges, allows for fast data structures that grant further speed-ups and fit most graphs into the main memory of a server with 32GB of RAM. Third, local strategies are better suited for dynamization. They potentially miss some global structure but since it is natural to assume that local changes on graphs are of local semantics only, local decisions on the clustering *should* suffice instead of rippling through the entire network.

Future Work. ORCA would benefit from the adaptation of better rules for network hubs, maybe based on some meta-decisions which again rely on global graph properties such as its degree distribution. Furthermore a fast dynamic version is desirable, which, given the clustering of some snapshot and a graph update, recomputes only affected parts of the clusterings.

neighborhood $N_d(v)$

Preliminaries. We briefly recall a few items from Section 1.2.1. A node v's (standard) *neighborhood* is $N(v) := \{w \in V \mid \{v, w\} \in E\}$, and the set of nodes within distance d of v is denoted as the *d-neighborhood* $N_d(v) = \{w \in V \mid w \neq v, \text{dist}(v, w) \leq d\}$, where $\text{dist}(v, w)$ denotes the length of the shortest path between v and w. In this section, Δ is the maximum degree in a graph, and we assume G to be connected, (and thus $n \in O(m)$). Otherwise the input is split into connected components in linear time.

2.5.1 The Orca-Algorithm

The general approach of Orca is as follows: Preliminarily prune the graph of irrelevant nodes, then, iteratively identify dense neighborhoods and contract them into *super-nodes*; after contraction repeat the second step on the next hierarchy level or, if this fails, remove low-degree nodes and replace them by shortcuts. Do this until the whole graph is contracted. Due to the widely agreed on fact that no quality function can be elected *best* in general, an important design goal for Orca was to refrain from having any decision base on such an index. Instead we only rely on fundamental and indisputable structural properties such as the 2-*core*, the similarity of a subgraph to a clique and local sparsity. The following sections detail each step of Orca in the order of their execution, things are then put together in Section 2.5.1.6. We postpone technical details of our implementation and our data structures to Section 2.5.3.

Orca in brief

2.5.1.1 Core-2 Reduction

The initial preprocessing step of Orca is a simple reduction of the instance to its 2-*core*. Introduced in [201], the 2-*core* of a graph is the maximal node-induced subgraph in which each node has at least degree 2 (for more details, see Section 3.1.3). Note that the running time of this procedure Core-2 Reduction is linear in $m + n$. The rationale behind this pruning step is as follows. Nodes in the 1-*core shell* are tree-like appendices, which are highly ambiguous to cluster sensibly anyway (see Figure 2.5.2a). Since in a reasonably modeled real-world network such appendices should not be large, we make the straightforward assumption that any tree appendix is to be clustered together with its anchor node in the 2-*core*, which is done in a postprocessing step. Depending on the nature of the input, this step can significantly reduce the size of the actual problem instance.

Algorithm 5: Core-2 Reduction

```
   Input: Graph G = (V, E, ω)
1  Stack deleteMe
2  deleteMe.addAll({v ∈ V | deg(v) < 2)
3  while deleteMe.nonEmpty do
4      v ← deleteMe.pop
5      forall w ∈ N(v) do
6          if deg(w) ≤ 2 then
7              deleteMe.push(w)
8          G.deleteNode(v)
```

reduction to 2-core

2.5.1.2 Local Search for Dense Regions

We now describe an integral part of Orca, the elementary detection of *dense regions*. Roughly speaking, a *dense region* $R \subseteq V$ is a set of c nodes within distance d of some seed node v, such that each node $w \in R$ is within distance at most d of at least $|N_d(v)|/\gamma$ other nodes of $N_d(v)$. This step is employed repeatedly and iteratively as will be described in the next section. The pseudocode of this step is given in Algorithm 6, and its behavior is illustrated in Figure 2.5.1.

dense regions

Each call of the procedure Dense-Region-Local is parameterized by a seed node v and two positive reals γ and d which set the required degree of density and the size of the neighborhood to be explored, respectively. Low values of γ impose a stricter criterion on density, which leads to Dense-Region-Local returning smaller regions. First, the *dense region* is initialized with the seed node v (Line 1). Then each candidate node, i.e., each node w within distance d or less from v (Line 2), in turn has each node $x \in N_d(w)$ increment its *seen*-attribute (Lines 3-4). For each node this attribute thus stores how many nodes of $N_d(v)$ it considered a neighbor. The second part of this procedure now adds each node $w \in N_d(v)$ to the *dense region*, which has been seen by at least a γ-fraction of the nodes in $N_d(v)$ (Lines 5-7) and returns the assembled region as in Figure 2.5.2b. Finally, the assembled *dense region* $(D, E(D))$, being a node-induced subgraph of G, is returned. Note that also allowing nodes in any $N(w)$ into a region might produce undesirable "holes". Furthermore, identifying *dense regions* in a way analogous to the computation to *k-cores* would be significantly slower.

depth param. d
density crit. γ
gather γ-dense nodes

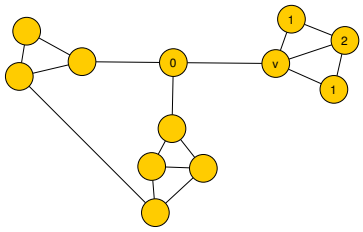


Figure 2.5.1. A run of DENSE-REGION-LOCAL starting at node v with $\gamma = 4$ and $d = 1$ assigns to each neighbor of v the "seen by neighbors" attribute.

Algorithm 6: DENSE-REGION-LOCAL

Input: $G = (V, E, \omega)$, $\gamma \in \mathbb{R}^+$, depth d, seed v
Output: Dense region

```
1 denseRegion ← {v}
2 forall w ∈ N_d(v) do
3     forall u ∈ N_d(w) do
4         u.seen++
5 forall w ∈ N_d(v) do
6     if w.seen ≥ |N_d(v)| / γ then
7         denseRegion.add(w)
8 return denseRegion
```

The time complexity of this procedure is highly dependent on d. Setting $d = 1$ at most Δ nodes each have their at most Δ neighbors increment their attribute, yielding $O(\Delta^2)$. In the worst case (G being a clique), this can amount to a running time of $\Omega(n^2)$. However, as we set our focus on practical instances, we ignore such pathological cases.

2.5.1.3 Contraction of Dense Regions

contract a dense region

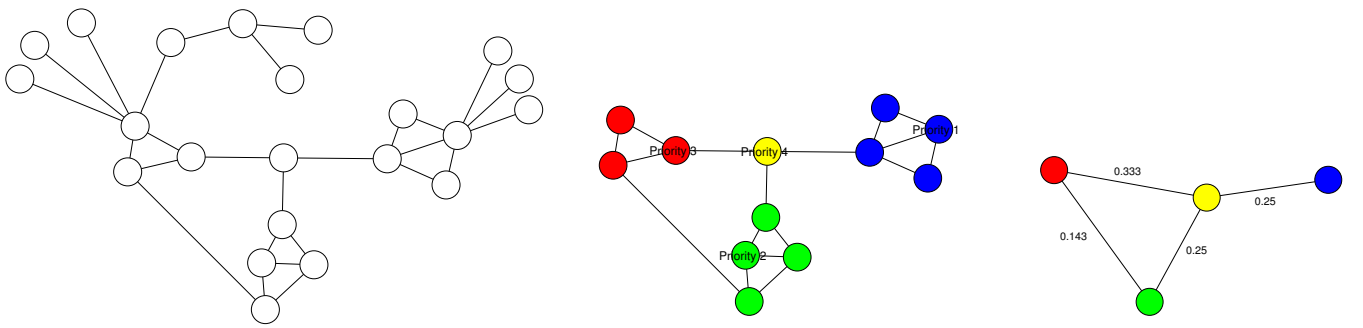
The second elementary operation on the graph is the contraction of a subgraph into a single *super-node*. The main goal of CONTRACTION is to reduce the size of the problem instance by summarizing parts that have already been solved; Figure 2.5.2 illustrates its effect and Algorithm 7 gives its pseudocode. Naturally, contracted subgraphs can participate in later *dense regions* and thus grow even further. A useful byproduct of iterative contraction is the construction of a *hierarchy* of clusterings. the general methodology of an instance

Algorithm 7: CONTRACTION

Input: $G = (V, E, \omega)$, Nodes to contract $\mathcal{D}$

```
1 create a super-node s in G
2 forall edges e = {v, w} with v ∈ D, w ∈ V \ D do
3     if {s, w} ∉ E then
4         insert edge {s, w}, ω({s, w}) = 0
5     ω({s, w}) ← ω({s, w}) + ω({v, w}) / |D|
6 remove nodes D
```

The contraction of a node-induced subgraph of G is straightforward. A new node replaces the subgraph, and is receives former adjacencies to other nodes are replaced by new edges, weighted by their average adjacency to the region. A rough upper bound on the running time of such a CONTRACTION clearly is $O(m)$, since each edge is touched only once. An amortized analysis of the time complexity of a series of calls of Algorithm DENSE-REGION-LOCAL and CONTRACTION will be given in the next section.



(a) The input instance, first it will be reduced to its 2-*core* by the removal of appendices.

(b) Now, a set of *local dense regions* are identified (colors).

(c) Each *dense region* is contracted by priority.

Figure 2.5.2. ORCA starts: On the 2-*core*, *dense regions* (by colors) are contracted in the order of contraction priority.

2.5.1.4 Global Dense Region Detection

Dense-Region-Global orchestrates

While procedure Dense-Region-Local identifies a *dense region*, and procedure Contraction reduces it to a *super-node*, the following algorithm, called Dense-Region-Global, orchestrates the calls to these local operations. Roughly speaking, a single run of Dense-Region-Global assigns each node to a prioritized *dense region* (Figure 2.5.2b), and then abstracts the graph to the next hierarchy level by replacing each *dense region* by a *super-node* (Figure 2.5.2c). The crucial observation is that Dense-Region-Global reduces the size of the instance very quickly and in a meaningful way, paving the way for further and more far-reaching clustering steps.

Algorithm 8: Dense-Region-Global

```
Input: G = (V, E, ω), γ ∈ ℝ⁺, search depth d
PriorityQueue pq
forall v ∈ V do
    denseRegion ← Dense-Region-Local(G, γ, d, v)
    pq.insert(v, ψ(denseRegion))
List contractionList
while !pq.isEmpty() do
    v ← pq.popMax()
    denseRegion ← Dense-Region-Local(G, γ, d, v)
    ExludeFromSearch(denseRegion)
    contractionList.add(denseRegion)
forall denseRegion ∈ contractionList do
    Contraction(denseRegion)
```

Given parameters γ and d as above, Dense-Region-Global first calls for each node v in the graph Dense-Region-Local using v as the seed node. Each *dense region* returned is then inserted into a priority queue with a priority key that expresses how significant the region actually is, as indicated in Figure 2.5.2b. This key is computed by evaluating the following simple function ψ that measures the average edge weight mass incident with a node in the region:

priority key ψ

$$\psi : \mathcal{P}(V) \to [0,1] \qquad D \mapsto \frac{\sum_{e \in E(D)} \omega(e)}{|D|} \quad , \quad D \subseteq V$$

As mentioned in Section 2.1.3, a very recent alternative approach to accomplish this ranking of anomalously dense local subgraphs, which we have yet to compare, is given in [220] and uses *spacial scan statistics* from the field of data mining. However, we shall not detail it here. After determining and queuing for each node $v \in V$ its *dense region*, regions are popped from the queue and contracted. Since we seek a proper partition of nodes, we first have to reassemble *dense regions* excluding all nodes that are assigned to *dense regions* with a higher priority by tagging them as invalid. Experiments showed that reordering the queue after such exclusions is costly and yields a minimal gain in quality, thus initial priorities are kept.

avoid overlaps

In total, n calls of Dense-Region-Local account for $O(n\Delta^2)$ and n priority queue operations require $O(n \log n)$ time. During the course of all Contraction operations each edge is touched at most twice, which yields an amortized time of $O(m)$. Summing up, Dense-Region-Global is in $O(m + n(\Delta^2 + \log n))$.

2.5.1.5 Densification via Shortcuts

densification with shortcuts

While initially, low degree nodes or appendices are pruned and assigned to clusters in a canonical way (see Core-2 Reduction), this might not be desirable for *super-nodes* incorporating thousands of elementary nodes. However, such low degree elements are potentially incompatible with a given choice of the threshold parameter γ Thus, we *densify* a graph, by

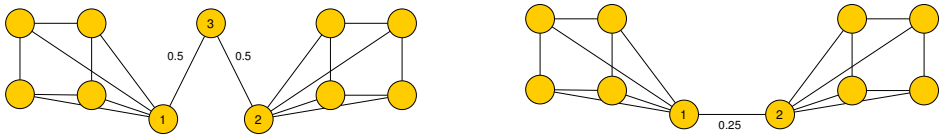


Figure 2.5.3. Shortcuts using $\delta = 2$, a shortcut between nodes 1 and 2, replaces node 3.

replacing a low-degree node v with a clique construction of *shortcuts* among its neighbors as in Figure 2.5.3. Similar to nodes removed during the CORE-2 REDUCTION, such a node is then potentially affiliated with the node it is most strongly connected to.

Algorithm 9: SHORTCUTS

Input: $G = (V, E, \omega)$
1 $\delta \leftarrow \min_{v \in V} \deg(v)$
2 **forall** $v \in V$ **do**
3 **if** $\deg(v) = \delta$ **then**
4 **forall** $p = \{v_1, v_2\} \mid v_1, v_2 \in N(v), v_1 \neq v_2$ **do**
5 **if** !∃ edge between v_1 and v_2 **then**
6 create edge $\{v_1, v_2\}$, $\omega(\{v_1, v_2\}) \leftarrow 0$
7 $\omega_1 \leftarrow \omega(\{v, v_1\})$
8 $\omega_2 \leftarrow \omega(\{v, v_2\})$
9 $\omega(\{v_1, v_2\}) \leftarrow \omega(\{v_1, v_2\}) + \frac{1}{\frac{1}{\omega_1} + \frac{1}{\omega_2}}$
10 remove v

Algorithm SHORTCUTS loops through all nodes v with the minimum degree δ, ensure that all pairs $\{v_1, v_2\}$ of nodes adjacent to v become connected and removes v. The weight on the edge between $\{v_1, v_2\}$ is then set to its new *conductivity*, a concept borrowed from electrical circuits: To the old weight, which is 0 if the edge was not present, the term $1/(\frac{1}{\omega(\{v_1,v\})} + \frac{1}{\omega(\{v_2,v\})})$ is added that expresses the *conductivity* of the path $\pi = v_1, v, v_2$. The rationale is that this adjustment maintains *conductivities* between all neighbors while densifying the graph structurally, again enabling the detection of *dense regions*. Analyzing the time complexity very roughly yields a worst case complexity of $O(n \cdot \Delta^2)$.

2.5.1.6 Putting Things Together

This section details the overall approach of ORCA, i.e., Algorithm 10 which repeatedly calls all necessary procedures. After the CORE-2 REDUCTION, for as long as there are more than two nodes left in the graph, DENSE-REGION-GLOBAL and SHORTCUTS iteratively reduce and contract the graph. If at any time no significant contraction is possible (Line 4), SHORTCUTS removes low degree nodes and compactifies the graph (Line 5). After each successful global contraction stage we store the current clustering (Line 6). ORCA returns the whole clustering hierarchy alongside evaluated quality indices for manual choice of the preferred clustering. Additionally a recommendation is given, based on quality indices. In practice, procedure SHORTCUTS is hardly ever called, and no value of $\delta > 2$ was ever used, leaving SHORTCUTS with a marginal impact on running times. Only with ill-modeled graphs, pathological examples or unreasonable choices of γ does this procedure ever operate on a graph with size comparable to the input, usually it is only called after a series of contraction steps. We discuss good choices for the two parameters γ and d in the following section.

ORCA *'s output*

SHORTCUTS *rare*

Algorithm 10: ORCA

Input: $G = (V, E, \omega)$, $d, \gamma \in \mathbb{R}^+$
1 CORE-2 REDUCTION(G)
2 **while** $|V| > 2$ **do**
3 DENSE-REGION-GLOBAL(G, γ, d)
4 **if** $|V_{old}| > 0.25|V|$ **then**
5 SHORTCUTS(G,δ)
6 **else** Store current clustering

The total running time of ORCA derives from its subroutines, and factor h, the number of iterations of the main loop or, in other words, the depth of the clustering hierarchy, which is n in the worst case, but always below $\log n$ in practice. However, since this work is on practical performance, we refrain from a detailed analysis and close with our observation that empirically the total running time of ORCA sums up to $O(\log n(m + n(\log n + \Delta^2)))$.

2.5.1.7 Engineering ORCA

We here shortly discuss two small optimizations for ORCA. It turns out that for particular graphs with a few high-degree nodes the running time of ORCA is dominated by the Δ^2 term. This particular observation has been made with other algorithms as well and seems to call for researching some proper preprocessing of such nodes. Hence, we use a little tweak to reduce

running times: After the CORE-2 REDUCTION, we remove all nodes with a degree greater than $4 \cdot \sqrt{n}$ from the graph, as these global hubs hardly indicate local density. Later we assign such a node to the cluster which contains most of its neighbors.

super-hubs are factored out

At later iteration steps, it is possible that the current clustering still contains many singleton elementary nodes, which have not found their way into any *dense region*. In order to reduce these undesirable clusters, we assign each singleton node to the cluster it is connected to most strongly.

assigning leftover nodes

2.5.2 Parameters and Feasibility

This brief section yields insights on reasonable choices for the parameters γ and d and corroborates the feasibility of ORCA on two toy examples. Parameter testing was conducted with the aid of two random generators that served as a source for graphs with an implanted clustering structure.

Parameter Estimation. We employed two generators for random test instances: first, an *Attractor Generator*, which is based on assigning nodes, randomly placed in the plane, to clusters in a Voronoi fashion and connecting them with probability based on distance and cluster affiliation; and second, a *Significant Gaussian Generator* which partitions the node set into clusters and then interconnects nodes similar to the Erdős-Rényi model, using intra- and inter-cluster edge-probabilities. We refer the reader to [71] for details on these generators, where they are evaluated and shown to produce reasonable and variable pre-clustered graphs with a tunable clarity. The latter generator is a slight variant of the generator described in Section 2.3.4.1. In a broad study on smaller graphs with 50 to 1000 nodes (step size 50), we varied the density parameter of the Attractor Generator from 0.5 (mostly disconnected stars) to 2.5 (almost a clique) in steps of 0.1, and we varied the intra-edge probabilities of the Significant Gaussian Generator between 0.1 (very sparse) and 0.7 (almost cliques) in steps of 0.1, having the ratio of inter-cluster edges range between 0.1 and 0.5 (0.05 step size). For each such setup we performed 30 experiments and evaluated the results of ORCA with respect to *performance*, *coverage* and *modularity*.

employed random graph generators

The results of this parameter exploration revealed that setting depth d to 1 for unweighted graphs is the best choice in general. The main reason for this is that a broader candidate neighborhood encourages "holes" inside clusters which at a later stage cannot be repaired. Parameter γ, proved to be feasible for values between 2 and 10 for sparse graphs, with low values working best in general.

$d = 1$ works best

values for γ

Two Toy Examples. In the following we show clustering results for two graphs, one of which is well known in the clustering community (and this thesis), and one that very fundamentally incorporates a clustering hierarchy. The latter graph is clearly organized into 16 small groups which themselves are organized into four groups, it was proposed in [153], as a basic benchmark for hierarchy detection. Figure 2.5.4 shows ORCA's results, a clear success. The second example was compiled by Wayne Zachary [230] while doing a study on the social structure of friendship between the members of a university sports club. The two real-world factions are indicated by color in Figure 2.5.5. Using $\gamma = 2$ and $d = 1$, ORCA clusters this graph as illustrated in Figure 2.5.6, where hierarchy levels 1 to 3 are shown. Level 3 misclassifies only a single node (as usual, the notorious node number 10, in the original numbering).

ORCA passes benchmark test

ORCA passes the Zachary test

2.5.3 Experiments

Implementation Details. Another field with huge datasets in algorithm engineering is the development of fast shortest path algorithms (see [74]). There we made the experience that in most cases, the loss with respect to running times stemming from external libraries is rather

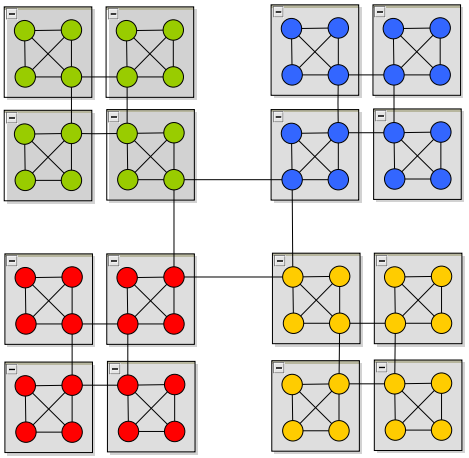


Figure 2.5.4. Hierarchy levels 1 (grouping) and 3 (colors) found by ORCA.

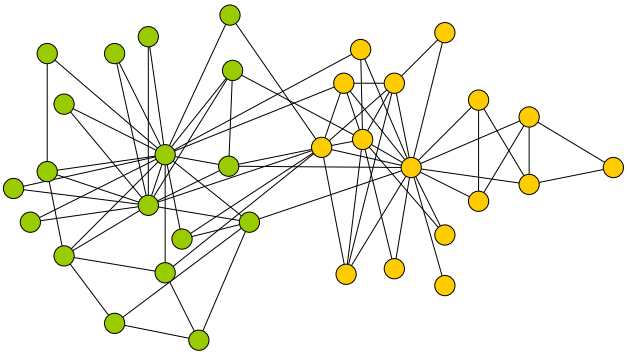
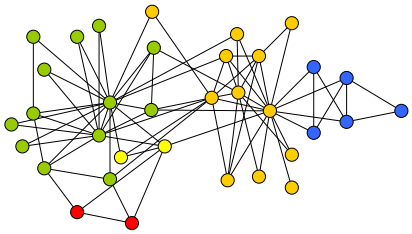
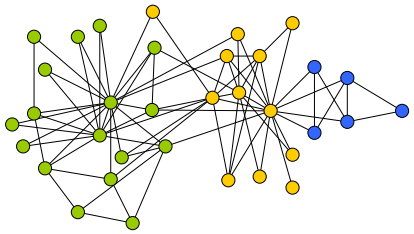


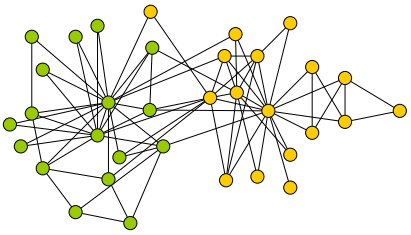
Figure 2.5.5. Zachary in reality (cov = 0.87, perf = 0.62, icc = 0.87, mod = 0.37).



(a) cov = 0.73, perf = 0.79, icc = 0.22, mod = 0.39



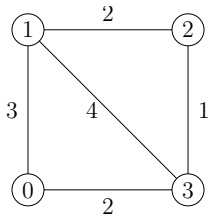
(b) cov = 0.82, perf = 0.71, icc = 0.75, mod = 0.40



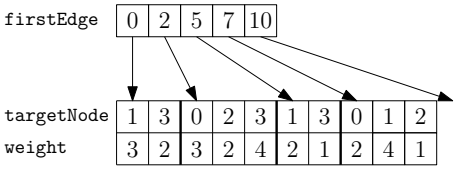
(c) cov = 0.87, perf = 0.62, icc = 0.87, mod = 0.37

Figure 2.5.6. Hierarchy levels 1 to 3 (left to right), using $\gamma = 2$ and search depth $d = 1$.

high. As the goal of ORCA was the development of a fast clustering algorithm, our implementation is written in C++, using only the STL at some points. As priority queue we use a binary heap, and we represent the graph as an adjacency array. In the following we report running times and quality achieved by ORCA, using fixed parameters $\gamma = 2$ and $d = 1$. For measuring Our tests were executed on one core of an AMD Opteron 2218 running SUSE Linux 10.3. It is clocked at 2.6 GHz, has 32 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with GCC 4.2, using optimization level 3. The data structure we used for the adjacency array representation of a graph is best looked up in [67]. In fact, we observed that the operations ORCA performs most—the iteration over incident edges—is very well supported by this data structure borrowed from the field of route planning. Figure 2.5.7b shows how graph G in Figure 2.5.7a is stored. Ids of nodes are implicitly stored by the position in the upper array, and the value at that entry is a pointer to the first of a node's incident edges. However, this edge is again only stored implicitly by the id of its second incident node. The end of the list of a node's incident edges is easily determined by looking up the start of the next node's first edge. In a second, corresponding array we store edge weights. It is easy to see that our data structure uses linear space, and supports finding neighbors in linear time. Furthermore, after a contraction, we build a new graph from scratch in linear time.



(a) an example graph G



(b) the data structure for G

Figure 2.5.7. A weighted graph G is represented by three arrays.

Inputs. We use three different types of inputs. Small world graphs, webgraphs and road networks. The first group contains three graphs. The first dataset represents the Internet on the router level, it is taken from the CAIDA webpage [52]. The second graph is a citation network, obtained from crawling citeseer [5]. The final dataset is a co-authorship [20] network, which is obtained from DBLP [4]. The second group of inputs are webgraphs taken from [6]. Nodes represent webpages, edges represent hyperlinks. We use four graphs, namely cnr-2000, eu-2005, in-2004 and uk-2002. The final group of inputs are road networks taken from the DIMACS homepage [75]. We use three graphs, the first one represents Florida, the second one central USA while the last one is the full road network of the US. Sizes are given in Tables 2.5.1-2.5.3.

2.5.3.1 Hierarchy of Clusterings

We first evaluate the clustering hierarchy computed by Orca. Figure 2.5.8 shows the score of all quality indices and the number of clusters for all levels of the hierarchy. For brevity, we restrict ourselves to one representative of each group of our inputs. As on higher hierarchy levels, the number of clusters decreases, coverage increases. It turns out that inputs are too large (contain degeneracies) for the worst-case index *inter-cluster conductance* to yield reasonable insights. Interestingly, *modularity* first increases and later decreases, yielding a clear preference. For sparse graphs *performance* is known to favor fine clusterings, but the point where *performance* severely decreases agrees with what *modularity* favors. Summarizing, Orca produces a reasonable clustering hierarchy from which a user can choose his favorite. A good choice seems to be a level, where *performance*, *coverage*, and *modularity* score best. To keep things brief here, the corresponding plots for the other graphs instances we clustered are summarized at the end of this section in Figure 2.5.9.

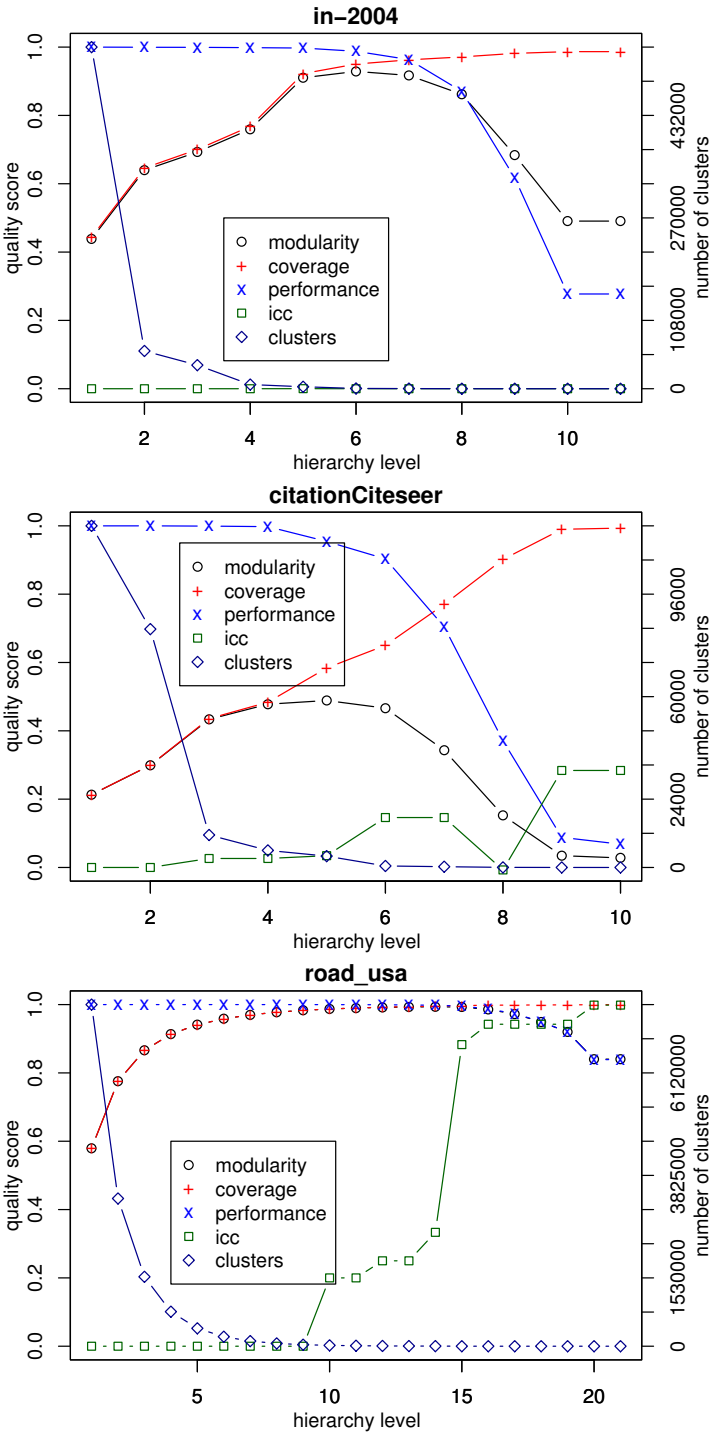


Figure 2.5.8. Quality of the clustering hierarchy computed by Orca. The inputs are the webgraph in-2004, the small world citation network, and the road network of the US.

2.5.3.2 Comparison

Next, we compare our results with competing graph clustering algorithms. We evaluate the *global* greedy *modularity* algorithm [57], the new *local* variant [38], and the *walktrap* [187]. We omit a number of other promising approaches, e.g., via *simulated annealing* [81], which

are computationally too demanding for these instance sizes. The implementations of *global greedy* and *walktrap* are taken from the *igraph* library [60], the code for *local greedy* is taken from [39]. Note that in the following we only report the clustering with maximum *modularity* for Orca, quality scores of other levels can be found in Figure 2.5.8 and 2.5.9.

local algorithms much faster

Small World Graphs. Tab. 2.5.1 reports running times and quality scores achieved by Orca and competing algorithms applying our three small world inputs. We observe excellent running times for Orca with feasible quality scores. Moreover, we observe that in terms of running time, *global greedy* and *walktrap* cannot compete with the local algorithms. While this is to be expected, note that they do not achieve better quality scores either. Comparing Orca with *local greedy* we observe that Orca tends to produce finer clusterings. Roughly speaking, quality scores are worse than for *local greedy* but still feasible. For the instance citation, Orca fails to find a very good clustering, this is mainly due to many high degree hubs—milestone papers or major surveys, where Orca seems to take too many simplification steps (see Engineering Orca above). Please refer to Figure 2.5.9 for quality indices on different hierarchy levels of Orca.

Orca competes well

Webgraphs. Next, we focus on the scalability of our approach. The webgraphs we have taken from [6] have similar properties but different sizes. It turns out that the *global greedy* algorithm needs too much memory to be executed while *walktrap* takes too much time. Hence, we compare Orca with the *local greedy* algorithm only, Tab. 2.5.2 reports running times and quality scores. At a glance we observe that Orca provides good clusterings within reasonable computation times. All graphs are clustered in less than 2.5 hours. Only for eu-2005, we achieve a *modularity* score of less than 0.85, and do not agglomerate long enough to find a better clustering. Interestingly, *inter-cluster conductance* is always almost zero for Orca. This stems from the fact that, *inter-cluster conductance*, being a worst-case quality index, always considers a clustering with at least one singleton a very poor clustering. While this may make sense for small inputs, such a worst-case index is not reliable for large inputs. As observed in Fig. 2.5.8, in most cases clusterings on a higher level score higher values. Comparing Orca with *local greedy*, we observe that Orca has longer running times but achieves comparable quality scores on these large inputs, neglecting *inter-cluster conductance*. On cnr-2000 and eu-2005 *local greedy* has a slight advantage in terms of quality indices while on in-2004 and uk-2002 Orca yields higher values. On these two instances, Orca outperforms the *local greedy* method in terms of *modularity*—especially on uk-2002 by a surprisingly large margin. Although the latter technique merges groups of nodes until no more improvement in *modularity* can be attained, it seems to fundamentally run past the innate clustering structure

Table 2.5.1. Running times and quality of the algorithms on small world graphs.

Instance	n/m	Algorithm	time [h:mm]	clusters	icc	perf.	cov.	mod.
caida Router	190 914 / 607 610	*global greedy*	0:20	1672	0.5667	0.9397	0.9052	0.7639
		Walktrap	0:23	24952	0.0000	0.9858	0.7540	0.6693
		local greedy	< 0:01	442	0.6410	0.9720	0.8944	0.8440
		Orca	< 0:01	492	0.2105	0.9578	0.7113	0.6500
co-Authors	299 067 / 977 676	*global greedy*	1:15	2930	0.5000	0.9187	0.8638	0.7413
		Walktrap	0:55	37669	0.0000	0.9790	0.7089	0.6432
		local greedy	< 0:01	269	0.6196	0.9813	0.8486	0.8269
		Orca	< 0:01	2038	0.1733	0.9954	0.7274	0.7212
citations	268 495 / 1 156 647	*global greedy*	2:08	1927	0.2857	0.8253	0.9106	0.6650
		Walktrap	0:51	16822	0.0000	0.9690	0.7449	0.6824
		local greedy	< 0:01	147	0.5983	0.9544	0.8593	0.8037
		Orca	< 0:01	4201	0.0000	0.9973	0.5649	0.5100

Table 2.5.2. Running times and quality of the algorithms on webgraphs.

Instance	n/m	Algorithm	time [s]	clusters	icc	perf.	cov.	mod.
cnr-2000	325 556	*local greedy*	8	242	0.8571	0.9799	0.9971	0.9130
	5 565 376	Orca	28	110	0.0002	0.9632	0.9427	0.8567
eu-2005	862 664	*local greedy*	23	326	0.7668	0.9643	0.9708	0.9376
	32 778 307	Orca	307	217	0.0002	0.9458	0.7965	0.7014
in-2004	1 382 908	*local greedy*	36	1004	0.0000	0.9931	0.9234	0.9094
	27 560 318	Orca	313	740	0.0002	0.9877	0.9503	0.9288
uk-2002	18 520 486	*local greedy*	432	6280	0.0000	0.9981	0.5693	0.5671
	529 444 599	Orca	8807	66595	0.0000	0.9995	0.8758	0.8749

Table 2.5.3. Running times and quality of the algorithms on road networks.

Instance	n/m	Algorithm	time [s]	clusters	icc	perf.	cov.	mod.
florida	1 070 376	*local greedy*	15	541	0.9845	0.9978	0.9971	0.9948
	2 687 902	Orca	37	48	0.9609	0.9954	0.9913	0.9866
central	14 081 816	*local greedy*	–	–	–	–	–	–
	33 866 826	Orca	1116	343	0.9319	0.9943	0.9966	0.9909
usa	23 900 746	*local greedy*	–	–	–	–	–	–
	58 389 712	Orca	1317	209	0.9424	0.9980	0.9954	0.9933

of this network, since Orca identifies ten times as many clusters, with both a significantly higher *coverage* and *modularity*. At this point it is worth noting that the size of the *local greedy* clustering monotonously scales with the number of nodes (except for the smallest instance). This is paralleled by the predictable and artificial behavior of the *modularity* index, favoring a balance of (small) degree sums within clusters over *coverage*. This might be the reason for the algorithm's behavior on uk-2002, which seems better clustered much finer. Again, please refer to Figure 2.5.9 for quality indices on different hierarchy levels of Orca.

Orca's modularity higher than mod.-based greedy

Road Networks. Unfortunately, *walktrap* and *global greedy* are way too slow for this input and the implementation of the *local greedy* algorithm crashes with a segmentation fault, for reasons unknown to us. Hence, we conclude that Orca is currently the only graph clustering algorithm working on large instances of such kinds of inputs. As observable in Tab. 2.5.3, both running times and quality scores are excellent. All quality indices score a value higher than 0.94 and most scores are on a high standard for many levels of the hierarchy, see Figure 2.5.9. We need less than 22 minutes to construct all levels of the hierarchy. Note that although usa is almost double the size of central, and Orca clusters the former even coarser; running times are very similar. Together with the very high quality values, usa seems to be an easy instance.

only Orca finishes

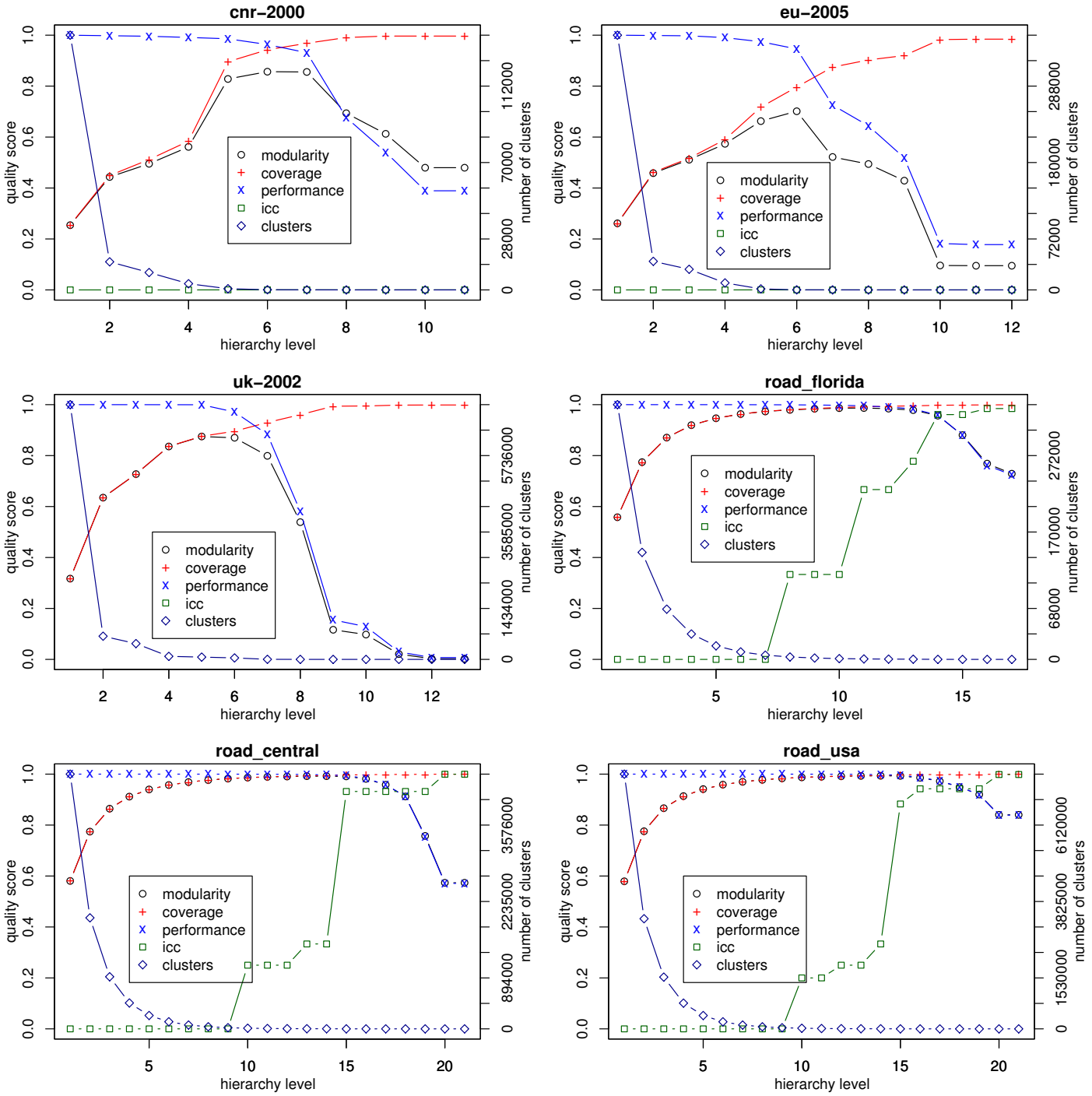


Figure 2.5.9. Quality of the clustering hierarchy computed by ORCA.

2.5.4 Example: a Complete Orca-Run

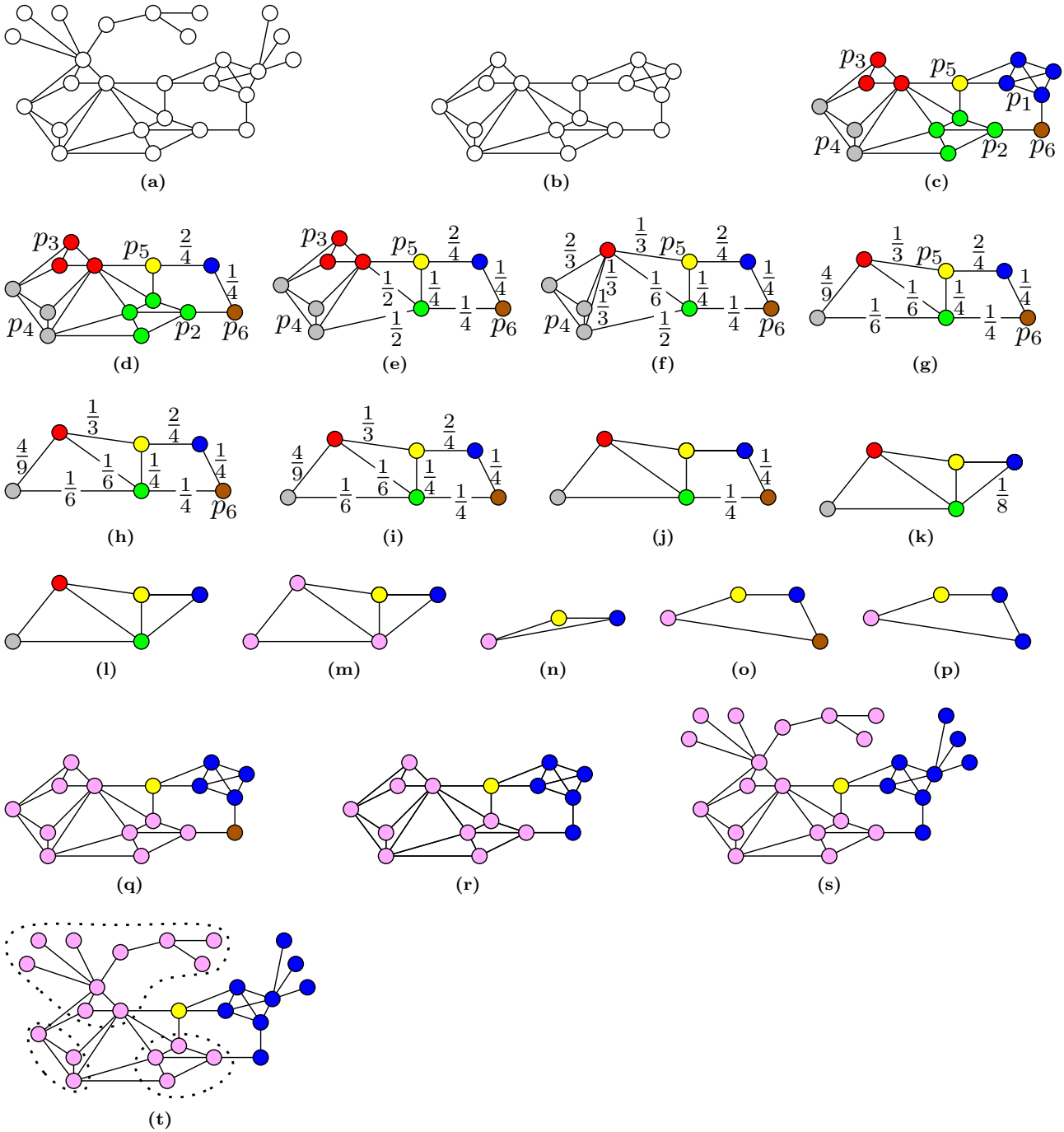


Figure 2.5.10. An illustrative complete run of Orca on a tiny example. Note the shortcutting of the brown node in Figures (j)-(k). In Figure (n) Orca is actually finished, then the contracted subgraphs are unfurled. In the end, the yellow node is left as a singleton cluster, as it did never actually take part in a *dense region*. Figure (u) illustrates that a user is also presented clusterings from intermediate hierarchy levels, such as given by the dashed subclustering.

Section 2.6

Comparing Clusterings

This chief, one Nosnra, is a grossly fat and thoroughly despicable creature, sly and vicious, loving ambush and backstabbing.

(Against the Giants, Gary Gygax)

Suppose we are given a few clusterings, each one stemming from some individual opinion on how to group the nodes of some graph. Can we quantify how much two opinions differ? Consider another scenario: We require a good clustering algorithm for a specific application, and we have access to a number of controlled preliminary experiments, where we know the *ground-truth* clusterings of typical instances of the application. Can we use this information to find a suitable clustering algorithm for the application? We could now simply rely on some quality indices, which might work well. However, we could also choose the algorithm which on the average is as *close* as possible to the *ground truth* on the test instances. And as a final question, which anticipates Chapter 4, suppose a network changes dynamically, and we are to cluster it periodically. Can we quantify how much the clustering changes between two consecutive steps?

For each of these tasks we require a means to measure the *distance* between two clusterings. After we have spent quite a few pages on quality measures, it is immediate that there exists a mutual relation between the two concepts of quality and distance: On the one hand, one could use the difference of the qualities of two clusterings as a distance measure, and on the other hand, measuring the distance of a given clustering to some "optimal" clustering could yield a measure of the quality of the clustering. However, there are pressing reasons against this approach: The subjective dependence on the used quality measure and, more importantly, the fact that completely different clusterings may yield the same quality value and are thus judged to be equal. Current techniques for the comparison of clusterings mainly use existing measures from the field of data mining [218], which have a crucial drawback: they only consider the partition of nodes and ignore the structure of graphs.

In this section we address these drawbacks and introduce new approaches combining structural properties and qualitative aspects. In order to achieve this, we extend data mining measures by adding qualitative features and introduce a new promising measure having its origin in quality measurement. Since comparing clusterings requires keeping many dimensions and aspects in consideration, we focus on the case of static comparison, i.e., the graph does not chang, but give an outlook on the case where the underlying graphs of two clusterings are allowed to differ. In an experimental evaluation we postulate certain traits of intuitive behavior of distance measures in controlled experiments, and show that the drawbacks of data mining measures are not only theoretical in nature but manifest often, and that our proposed approaches comply with our postulations. Summarizing, extensions of established set-based measures to graph-based measures are not trivial and need not lead to intuitive results, however, some do so, as we show. The new measure we propose, the *editing set difference*, can be

recommended as a reasonable measure and can naturally cope with a dynamic setting, where the edge set is allowed to change over time.

The vision of fully dynamic graph clustering is—again, at least in my personal reckoning—long-standing. However, there are numerous points of uncertainty and unresolved questions even in the static context, which holds back a theoretician. One of these concerns the comparison of clusterings, and thus motivated our work on it. Most of the material in this section has been published in [68, 69, 70], based on joint work with Daniel Delling, Marco Gaertler and Dorothea Wagner. In my eyes it is a very reasonable approach towards measures that are specific to *graph* clustering and not provisionarily borrowed from data mining. However, I suspect that in first works to come on dynamic graph clustering, measures described in this section will initially have a hard time and will probably have to yield to traditional and well known measures from data mining. The reason for this is simple: it is much harder to convince people of two new and freshly designed concepts at once. We shall take that risk in Section 4.3.

Main Results

- We conceive a systematic approach for *extending* whole classes of *set-based distance* measures to *graph-based* measures. These classes are measures that use *pair-counting*, *overlaps* or *entropy*. (Section 2.6.3.1)
- We design and advocate a new *graph-structural* distance measure $\mathcal{ESD}$ based on the notion of the *cluster editing-set*.
- A controlled experimental evaluation exposes how traditional measures violate intuitive postulations for distance measures and shows that our new measure $\mathcal{ESD}$ and some of the *extended* measures comply, the *extensions* of the adjusted Rand index and Fred & Jain's index in particular. We thus arrive at sound recommendations on which measures to use and what behavior to be aware of.

2.6.1 Preliminaries

Since we have not used these terms for a while now, recall from Section 1.2.2 that we call a graph with disjoint cliques a *clustergraph*. Moreover, the *cluster editing set* $F_{\mathcal{C}}$ is the set of edges to be added to or deleted from a given graph in order to transform the graph and a given clustering $\mathcal{C}$ into an according clustergraph, i.e., such that the clusters constitute the cliques. When comparing two clusterings we use $\mathcal{C}$ and $\mathcal{C}'$, with $k := |\mathcal{C}|$, $l := |\mathcal{C}'|$. Furthermore, it should be noticed, that all presented measures are given in a *distance* version, normalized to the interval $[0, 1]$, from equal/very close (0) to very distant/dissimilar (1). Although most of the results in this Section can immediately be transferred to weighted graphs, we keep things restricted to simple graphs here, since in this conceptual approach, weights will only disrupt notation and introduce special cases.

clustergraph

cluster editing set

distances (not similarities)

2.6.2 Existing Distance Measure

In the following, we give a short overview of existing comparison techniques. Among them are both measures based on quality and on comparing the partitions of node sets. The latter are also called *node-structural*.

node-structural

Quality-Based Distance. Quality-based measurements can be constructed by comparing the scores of the two clusterings with respect to an arbitrary quality index such as *coverage*, *performance* or *modularity* [46, 57]. Note, that a distance measured in such a way is highly dependent on the used index. Furthermore, completely different clusterings can yield the same value. Thus, we neglect purely quality-based distances in the following and focus on measuring the distance based on the structure of the clusterings.

quality-based

counting pairs

Counting Pairs. In [218] some techniques based on *counting pairs* are presented. Summarizing, every pair of nodes is categorized based on whether they are in the same (or different) cluster with respect to both clusterings. Four sets are defined: S_{11} (S_{00}) is the set of unordered pairs that are in the same (different) clusters under both clusterings, whereas S_{01} (S_{10}) contains all pairs that are in the same cluster under $\mathcal{C}$ ($\mathcal{C}'$) and in different under $\mathcal{C}'$ ($\mathcal{C}$). In the following we present two representatives for this class: *Rand* and *adjusted Rand* measure. Rand introduced the distance function $\mathcal{R}$ given in Equation 2.6.1 in [189], it suffers from several drawbacks. For example, it is highly dependent on the number of clusters. One attempt to remedy some of these drawbacks, which is known as *adjusted Rand* $\mathcal{AR}$ and given in Equation 2.6.1, is to subtract the expected value for clusterings with a hypergeometric distribution of nodes, see [167].

$S_{11}, S_{00}, S_{01}, S_{10}$

Rand $\mathcal{R}$

adjusted Rand $\mathcal{AR}$

$$\mathcal{R}(\mathcal{C},\mathcal{C}') := 1 - \frac{2(n_{11}+n_{00})}{n(n-1)} \quad , \quad \mathcal{AR}(\mathcal{C},\mathcal{C}') := 1 - \frac{n_{11}-t_3}{\frac{1}{2}(t_1+t_2)-t_3} \quad , \tag{2.6.1}$$

where $t_1 := n_{11}+n_{10}$, $t_2 := n_{11}+n_{01}$, and $t_3 := (2t_1t_2)/(n(n-1))$ and t_1 (t_2) is the cardinality of all pairs of nodes that are in the same cluster under $\mathcal{C}$ ($\mathcal{C}'$).

overlaps

confusion matrix

van Dongen $\mathcal{NVD}$

Overlaps. Another counting approach is based on the $k \times l$ *confusion matrix* $CM := (m_{ij})$ whose ij-entry indicates how many elements are in cluster C_i and C'_j, i.e., how large is the *overlap*, formally $m_{ij} := |C_i \cap C'_j|$, for $1 \leq i \leq k$ and $1 \leq j \leq l$. Several measures are based on the *confusion matrix*. We restrict ourselves to the measure $\mathcal{NVD}$, introduced by van Dongen in [213], given in Equation 2.6.2. Other measures suffer from the obvious disadvantage of asymmetries, thus we exclude them. We use a normalized version to keep the measure to the interval $[0,1]$.

$$\mathcal{NVD}(\mathcal{C},\mathcal{C}') := 1 - \frac{1}{2n}\sum_{i=1}^{k}\max_j m_{ij} - \frac{1}{2n}\sum_{j=1}^{l}\max_i m_{ij} \tag{2.6.2}$$

One major drawback of $\mathcal{NVD}$ is that the distance between the two trivial clusterings, i. e., $k = 1, l = n$, only yields a value of about 0.5. In addition, this measure suffers from the drawback that only the maximum *overlaps* contribute, resulting counter-intuitive examples are given in [163].

entropy-based

entropy $\mathcal{H}$

mutual information $\mathcal{I}$

Information Theory. More promising approaches are based on information theory [59]. Informally, the *entropy* $\mathcal{H}(\mathcal{C})$ of a clustering is the uncertainty of a randomly picked node belonging to a certain cluster. The *entropy* of a clustering is always positive and is bounded by $\log_2(n)$, see [206]. An *extension* of *entropy* is the *mutual information* $\mathcal{I}(\mathcal{C},\mathcal{C}')$. The *mutual information* of two clusterings is the loss of uncertainty of one clustering if the other is given. With $P(i) := |C_i|/n$ and $P(i,j) := (|C_i \cap C'_j|)/n$, *entropy* and *mutual information* are defined as follows.

$$\mathcal{H}(\mathcal{C}) := -\sum_{i=1}^{k} P(i)\log_2 P(i) \quad , \quad \mathcal{I}(\mathcal{C},\mathcal{C}') := \sum_{i=1}^{k}\sum_{j=1}^{l} P(i,j)\log_2\frac{P(i,j)}{P(i)P(j)} \tag{2.6.3}$$

Note that *mutual information* is positive and bounded by $\min\{\mathcal{H}(\mathcal{C}),\mathcal{H}(\mathcal{C}')\} \leq \log_2(n)$. In the following we present two representatives in this class, namely one introduced by Fred & Jain in [140] and *variation of information*, introduced by Meila in [162].

Fred & Jain $\mathcal{FJ}$

variation of information $\mathcal{VI}$

$$\mathcal{FJ}(\mathcal{C},\mathcal{C}') := \begin{cases} 1 - \frac{2\mathcal{I}(\mathcal{C},\mathcal{C}')}{\mathcal{H}(\mathcal{C})+\mathcal{H}(\mathcal{C}')} & \text{, if } \mathcal{H}(\mathcal{C})+\mathcal{H}(\mathcal{C}') \neq 0 \\ 0 & \text{, otherwise} \end{cases} \tag{2.6.4}$$

$$\mathcal{VI}(\mathcal{C},\mathcal{C}') := \mathcal{H}(\mathcal{C}) + \mathcal{H}(\mathcal{C}') - 2\mathcal{I}(\mathcal{C},\mathcal{C}') \tag{2.6.5}$$

The first measure $\mathcal{FJ}$, given in Equation 2.6.4, is a normalized version of the *mutual information* and stated as a distance function. The case differentiation is used to deal with the degenerated case of two trivial clusterings, i. e., $k = l = 1$.

The second measure $\mathcal{VI}$ is motivated by an axiomatic approach and given in Equation 2.6.5. In [163], it is shown that $\mathcal{VI}$ is the only measure fulfilling several axioms. However, these axioms seem to be inadequate in the special case of graph clustering. According to these axioms, the movement of a node v from one cluster C_i to another cluster C_j must be equivalent to first splitting v off from C_i and then merging it with C_j. Figure 2.6.1 shows an example regarding this axiom: intuitively $d(\mathcal{C}, \mathcal{C}'')$ should be greater than $d(\mathcal{C}, \mathcal{C}') + d(\mathcal{C}', \mathcal{C}'')$ of which both terms represent minor changes, but according to the axiom $d(\mathcal{C}, \mathcal{C}'') = d(\mathcal{C}, \mathcal{C}') + d(\mathcal{C}', \mathcal{C}'')$ must hold. This measure is not normalized and the two possible normalization factors, which are $1/\log_2(n)$ and $1/\log_2(\max\{k, l\})$, mapping to the intervals $[0, x], x \leq 1$ and $[0, 1]$ respectively, have significant drawbacks. Nevertheless, we use the $\log_2(n)$ normalized version for comparability with the other measures.

axiomatic approach

Drawbacks of the Data Mining Approach. All *node-structural* measures suffer from the same drawback: They neglect the structure of the graph. The examples in Figure 2.6.2 clarify this circumstance. The figure shows four clusterings $\mathcal{C}_1, \mathcal{C}_1', \mathcal{C}_2$ and $\mathcal{C}_2'$ on two graphs G_1 and G_2. A measure d not considering the structure of the graphs fulfills $d(\mathcal{C}_1, \mathcal{C}_1') = d(\mathcal{C}_2, \mathcal{C}_2')$. Intuitively, the distance $d(\mathcal{C}_1, \mathcal{C}_1')$ has to be greater than $d(\mathcal{C}_2, \mathcal{C}_2')$ since the quality of $\mathcal{C}_1$ is almost equal to that of $\mathcal{C}_2$, but $\mathcal{C}_1'$ has far lower quality than $\mathcal{C}_2'$. This drawback can become arbitrarily grave when the edge set of the graph is allowed to change.

graph structure is ignored

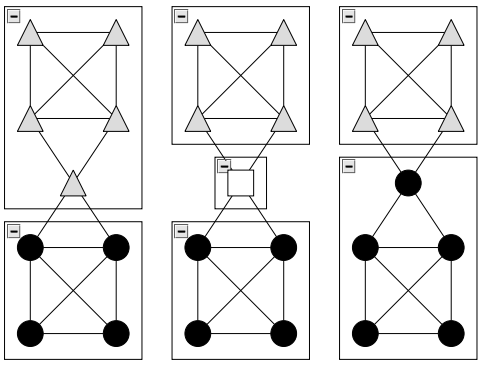
2.6.3 Engineering Graph-Structural Comparison Measures

In order to remedy some of the disadvantages of *node-structural* measures, we introduce the concept of *graph-structural* measures. Since they are also based on the underlying graph structure, they can include qualitative aspects for measuring the distance of two clusterings. In the first part, Section 2.6.3.1, we *extend node-structural* measures, while a novel measure is introduced in the second part, Section 2.6.3.2.

graph-structural measures

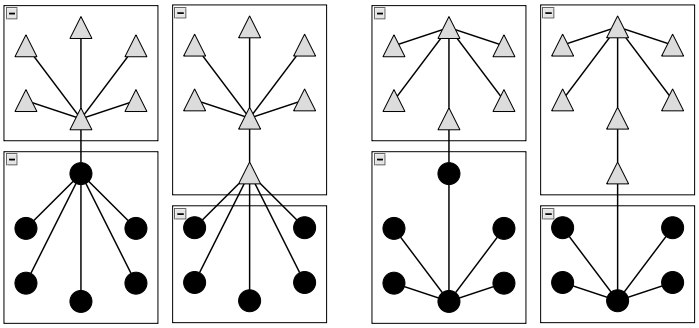
2.6.3.1 Extension of Node-Structural Measures

For consistency, all *extended* measures should meet the following requirement: If the underlying graph is complete, then both the graph- and *node-structural* version should yield the



(a) $\mathcal{C}$ (b) $\mathcal{C}'$ (c) $\mathcal{C}''$

Figure 2.6.1. Two minor changes sum up to a major one.



(a) $\mathcal{C}_1$ (b) $\mathcal{C}_1'$ (c) $\mathcal{C}_2$ (d) $\mathcal{C}_2'$

Figure 2.6.2. Two static comparisons of graph clusterings; if we ignore edges, both yield the same distance.

same value, since then the graph structure does not provide additional information. A second objective is to adjust the three founding principles—*counting pairs*, *overlaps* and *information theory*—of the existing measures themselves, instead of adjusting each implementation separately.

graph-based pair counting $\mathcal{R}_g, \mathcal{AR}_g$

Counting Local Pairs. Instead of categorizing every pair, we only consider those pairs, that are connected by an edge. For $a, b \in \{0,1\}$ we define $E_{ab} := S_{ab} \cap E$ and $e_{ab} := |E_{ab}|$. It is obvious that $S_{ab} = E_{ab}$ holds for complete graphs. Thus, we obtain the graph-based versions $\mathcal{R}_g$ and $\mathcal{AR}_g$ of the Rand and adjusted Rand measure given in Equation 2.6.6:

$$\mathcal{R}_g(\mathcal{C},\mathcal{C}') := 1 - \frac{e_{11}+e_{00}}{m} \quad , \quad \mathcal{AR}_g(\mathcal{C},\mathcal{C}') := 1 - \frac{e_{11}-t_3}{\frac{1}{2}(m(\mathcal{C})+m(\mathcal{C}'))-t_3} \quad , \tag{2.6.6}$$

where $t_3 := (m(\mathcal{C})m(\mathcal{C}'))/m$. Note, that $m(\mathcal{C}) = e_{11}+e_{10}$ and $m(\mathcal{C}') = e_{11}+e_{01}$, respectively, hold.

graph-based overlaps

Degree-Based Overlaps. Measures based on *overlaps* can be transformed into *graph-structural* measures by a slight modification in the definition of the *confusion matrix* as follows. The ij-th entry of the *degree-based confusion matrix* $CM^d := (m^d_{ij})$ indicates the sum of the degrees of the nodes that are both in C_i and C'_j, formally $m^d_{ij} := \deg(C_i \cap C'_j)$. Note, that if G is d-regular graph, then the equality $CM = CM^d/d$ holds. In certain cases, this may lead to different normalization factors. The *extension* of $\mathcal{NVD}$ is given in Equation 2.6.7.

$\mathcal{NVD}_g$

$$\mathcal{NVD}_g(\mathcal{C},\mathcal{C}') := 1 - \frac{1}{4m}\sum_{i=1}^{k}\max_j m^d_{ij} - \frac{1}{4m}\sum_{j=1}^{l}\max_i m^d_{ij} \tag{2.6.7}$$

The equivalence of the *node-* and the *graph-structural* variant of the normalized van Dongen measure for regular graphs follows from $m = dn/2$ and $m_{ij} = m^d_{ij}/d$.

graph-based entropy

Edge Entropy. The *entropy* defined in Section 2.6.1 solely depends on the node set, thus we *extend* it to the edge-set using the following paradigm: Instead of randomly picking a node from the graph for measuring the uncertainty, we pick the end of an edge randomly. As a consequence, a node with high degree has a greater impact on the distance. The formal definition of *edge entropy* $\mathcal{H}_E$ and *edge mutual information* $\mathcal{I}_E$ is given in Equation 2.6.8 and 2.6.9.

$$\mathcal{H}_E(\mathcal{C}) := -\sum_{i=1}^{k} P_E(i)\log_2 P_E(i) \quad , \tag{2.6.8}$$

$$\mathcal{I}_E(\mathcal{C},\mathcal{C}') := \sum_{i=1}^{k}\sum_{j=1}^{l} P_E(i,j)\log_2\frac{P_E(i,j)}{P_E(i)P_E(j)} \quad , \tag{2.6.9}$$

where $P_E(i) := \deg(C_i)/2m$ and $P_E(i,j) := \deg(C_i \cap C'_j)/2m$. Note that for regular graphs, the *entropy* and the *edge entropy* coincide. The *extensions* of $\mathcal{FJ}$ and $\mathcal{VI}$ are given in Equation 2.6.10 and 2.6.11.

$\mathcal{FJ}_g, \mathcal{VI}_g$

$$\mathcal{FJ}_g(\mathcal{C},\mathcal{C}') := \begin{cases} 1 - \frac{2\mathcal{I}_E(\mathcal{C},\mathcal{C}')}{\mathcal{H}_E(\mathcal{C})+\mathcal{H}_E(\mathcal{C}')} & \text{, if } \mathcal{H}_E(\mathcal{C})+\mathcal{H}_E(\mathcal{C}') \neq 0 \\ 0 & \text{, otherwise} \end{cases} \tag{2.6.10}$$

$$\mathcal{VI}_g(\mathcal{C},\mathcal{C}') := \mathcal{H}_E(\mathcal{C}) + \mathcal{H}_E(\mathcal{C}') - 2\mathcal{I}_E(\mathcal{C},\mathcal{C}') \tag{2.6.11}$$

The equivalence of the *node-* and the *graph-structural* variant for regular graphs results from the equality of *entropy* and *edge entropy* for complete graphs. Meila showed in [163] that $\mathcal{VI} \leq \log_2(n)$ also holds for weighted clusterings. Since the degree of a node can be interpreted as node weight our $\log_2(n)$-normalization maps to the interval of $[0,1]$.

2.6.3.2 A Novel Approach for Measuring Graph-Structural Distance

Although the *extensions* introduced in the previous section incorporate the underlying graph structure, they are not directly suitable for comparing clusterings on different graphs. In that case elements not existent in both graphs will have to be excluded from consideration. As a first step to solve this task, we consider the restriction to graphs with the same node set, but potentially different edge-sets. Our approach is motivated by the *cluster editing set* problem (see, e.g., [65]) which can be phrased as follows: What is the minimum number of edge-deletion and edge-insertion operations that suffice to change a given graph into a *clustergraph*? This problem has been shown to be NP-hard and *fixed parameter tractable* with the (size of the) solution as the parameter; please refer to the above reference and to further pointers therein. In our setting we do not require an optimal clustering (i.e., the "closest" *clustergraph*) to compare to, as we already have two reference clusterings at hand, which we can compare to. Based on this notion, we introduce the *editing set difference* defined in Equation 2.6.12.

cluster editing set

editing set difference $\mathcal{ESD}$

$$\mathcal{ESD}(\mathcal{C},\mathcal{C}') = \frac{|F_{\mathcal{C}} \cup F_{\mathcal{C}'}| - |F_{\mathcal{C}} \cap F_{\mathcal{C}'}|}{|F_{\mathcal{C}} \cup F_{\mathcal{C}'}|} = 1 - \frac{|F_{\mathcal{C}} \cap F_{\mathcal{C}'}|}{|F_{\mathcal{C}} \cup F_{\mathcal{C}'}|} \tag{2.6.12}$$

The *editing set difference* takes the *cluster editing set* of each of the graphs wrt. their given clusterings, and computes the geometric difference of these two sets. Edges which either both sets delete or add or which both sets leave untouched do not contribute to the distance. Small cluster editing sets correspond to significant clusterings. By comparing the two clusterings with a geometric difference, we obtain an indicator for the structural difference of the two clusterings. It easy to see, that in the case of static comparison, $\mathcal{ESD}$ is a metric. The example in Figure 2.6.3 illustrates how $\mathcal{ESD}$ operates. A noticeable property of $\mathcal{ESD}$ is the fact that two bad clusterings, which both need to edit many edges to reach their clustergraphs, will have a large normalizing term in the denominator. They will thus only have a large distance from each other if their *editing sets* largely differ. Clearer clusterings, which already are very close to their clustergraphs, are more sensitive to different *editing sets*. This is a property which we conjecture to be in agreement with intuition, and we shall come back to it in our postulations below. As a side note, in fully dynamic graph clustering, if large *batches* of

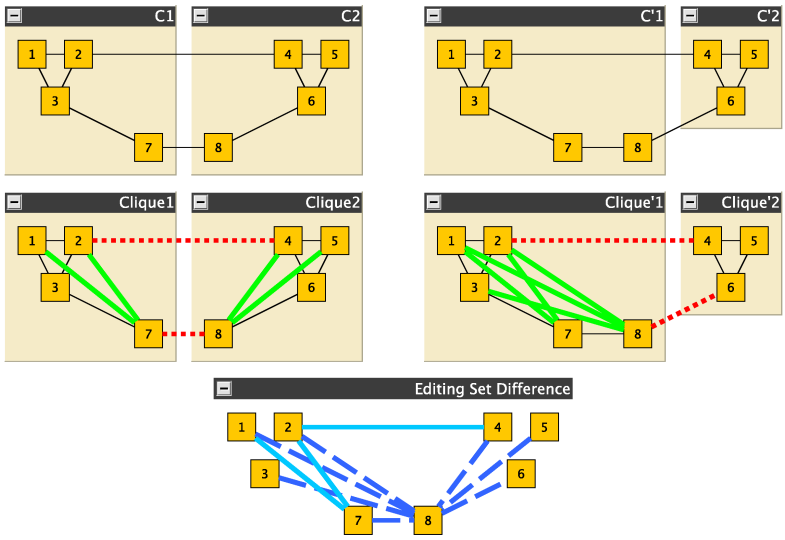


Figure 2.6.3. In the upper row, two clusterings $\mathcal{C}$ and $\mathcal{C}'$ (on the same graph) are given. The second line highlights their individual *editing sets*, deletions are red and insertions are green. The final figure then shows the edges that contribute to the distance (dashed) and those that only partake in the normalization (light, solid), as they are in both *editing sets*.

updates are to be expected, $\mathcal{ESD}$ might be improved if the sets of edges to be inserted and those to be deleted are handled separately. However, this point is not pursued any further in this work.

2.6.4 Experiments and Evaluation

experimental setup

We evaluate the introduced measures on two setups. The first focuses on structural properties of clusterings, the second concentrates on qualitative aspects:

Initial and Random Clusterings. The tests consist of two comparisons, each including clusterings with the same expected intrinsic structure of the partitions, i. e., the expected number of clusters and the size of clusters. The first comparison uses one significant clustering and one uniformly random clustering, while the second one uses two uniformly random clusterings.

Local Minimization. The setup consists of two parts, each comparing a reference clustering with a clustering of less significance. The two parts differ in the significance of the reference clustering.

The intuition of the first test is to clarify the drawbacks of the *node-structural* measures, while the second setup verifies the obtained results. We use the *attractor generator* introduced in [71] which uses geometric properties based on Voronoi Diagrams to generate initial clusterings. The Voronoi cells represent clusters and the maximum Euclidean distance of two nodes being connected is determined by a perturbation parameter. All tests use $n = 1000$ nodes and are repeated until the maximal length of the 0.95-confidence intervals is not larger than 0.1.

2.6.4.1 Initial- and Random Clusterings

first test: GvR vs. RvR

The generated clustering is used as a significant clustering. For the random clustering we first pick k uniformly at random between 2 and $\sqrt[3]{n}$ for the number of clusters and assign each node uniformly at random to the k clusters. Figures 2.6.4a and 2.6.4b show the measured quality by the indices *coverage*, *performance* and *modularity* (see Section 1.2.2). The tests consists of two cases. On the one hand, the comparison of the generated and a random clustering (GvR) and on the other hand, the comparison of two random clusterings (RvR).

Postulations

GvR: distance ∼ clarity

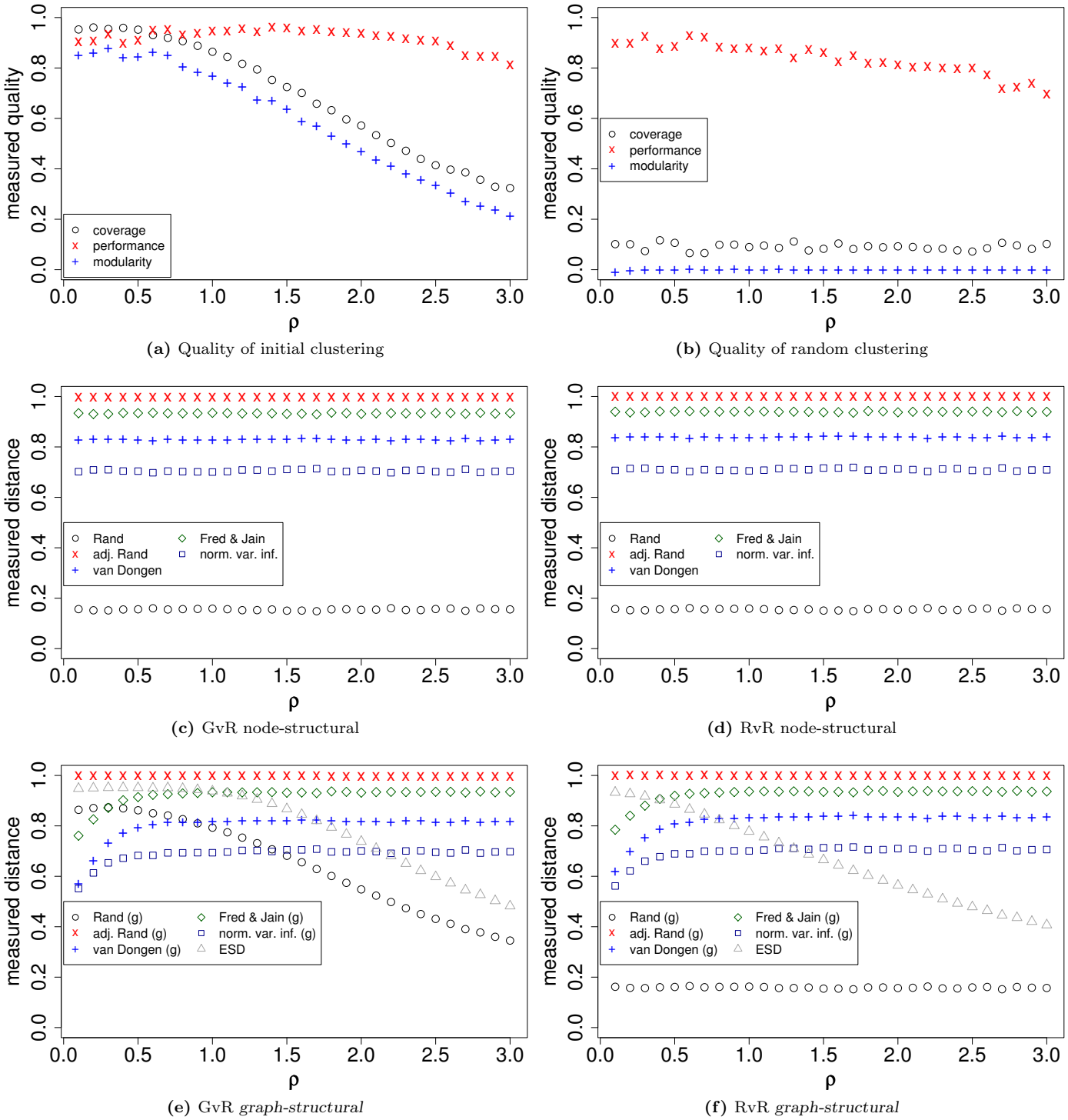
1. A measure for comparing graph clusterings should differ in the two cases. For GvR, a suitable measure should indicate a decreasing distance with the loss of significance of the reference. Still, the distance of any clustering to a random clustering should always be high.

RvR: distance ∼ clarity

2. For RvR two acceptable outcomes are possible: (i) On the one hand, one could claim that the distance between two random clusterings should be independent of the underlying graphs. (ii) On the other hand, the distance should decrease with the loss of significance because two random clusterings on an almost complete graph are closer to each other than on a graph with an existing significant clustering.

node-structural fails test

Results. Figure 2.6.4 shows the results for the *node-* and *graph-structural* measures. By comparing Figure 2.6.4c and 2.6.4d it is evident that *node-structural* measures do not distinguish the two cases. Only Fred & Jain and adjusted Rand reflect the interpretation that the distance to a random clustering is always maximal. However, the situation changes for the *graph-structural* distance (Figures 2.6.4e and 2.6.4f). Only Rand and $\mathcal{ESD}$ capture the difference, while the remaining measures show nearly the same behavior as their *node-structural*



(a) Quality of initial clustering

(b) Quality of random clustering

(c) GvR node-structural

(d) RvR node-structural

(e) GvR *graph-structural*

(f) RvR *graph-structural*

Figure 2.6.4. Results of the initial- and random clustering setup

counterparts. For GvR, the distance measured by Rand is decreasing with increasing density while for RvR the distance is invariant under the density. Furthermore, the measured distance equals the *node-structural* measurement for RvR. $\mathcal{ESD}$ has the same behavior for GvR as Rand, whereas RvR reflects the intuition that two random clusterings become more similar

$\mathcal{ESD}$ and $\mathcal{AR}_g$ pass test

Fred & Jain OK

with loss of significance. Under the assumption that a comparison to a random clustering should always be interpreted as maximal, adjusted Rand and Fred & Jain can be accepted. Nevertheless, the equivalence of the *node-* and the *graph-structural* versions of van Dongen and the normalized variation of information is counterintuitive. This partly originates from the fact, that attractors produce graphs that are close to regular for $\varrho > 0.5$. Furthermore, the clusters are equal in size. The strange behavior of Fred & Jain, van Dongen and the variation of information for very small ϱ stems from the fact that for small ϱ attractors are nearly stargraphs with k centers.

2.6.4.2 Local Minimization

second test: ruining a $\mathcal{C}$

Since there are several possible interpretations of *graph-structural* distance and the structural similarity of the clusterings in Section 2.6.4.1, a second test is executed having a precise intuition for *graph-structural* distance. Again, as a reference clustering we use the generated clustering of an attractor graph. The second clustering of less significance is obtained from the reference clustering by locally moving nodes from one cluster to another. Such a shift is executed, if it maximally decreases a given index among all possible shifts. This is done until no decrease of quality can be achieved or the number of moved nodes has reached a maximum value of $M_{\max}$; leaving a considerably bad clustering. We now measure the distance of each intermediate clustering to the initial clustering with our indices. In this setup, we use *modularity* as the index to be decreased, the density is set to the values $\varrho = 0.5$ (type 1) and $\varrho = 2.5$ (type 2), and $M_{\max}$ increases from 0 to 500 using steps of 5. As a control, Figures 2.6.5a and 2.6.5b show the measured quality of the locally decreased clusterings on increasing number of moved nodes.

Postulations

type 1 steeper than type 2

1. A suitable distance measure should first of all distinguish the two cases. In type 1 a very clear clustering is iteratively deteriorated, which should immediately results in high distances. In type 2 rather unclear clustering is further made worse, this should not yield distances as high as in type 1.

distance curves should flatten

2. In addition, with increasing $M_{\max}$ the distance curves of both types should flatten, since in the beginning clustering structure is lost, but after a while only mere random partitions are further estranged from the original.

type 1 $\geq$ type 2

3. The total distance of type 2 must only very late reach the level of the distance in type 2, if at all, since type 1 always compares to clearer initial structure than type 2. Only when mere randomness is reached, can a distance close to 1 be accepted for type 2.

$\mathcal{ESD}$ and $\mathcal{AR}_g$ pass test

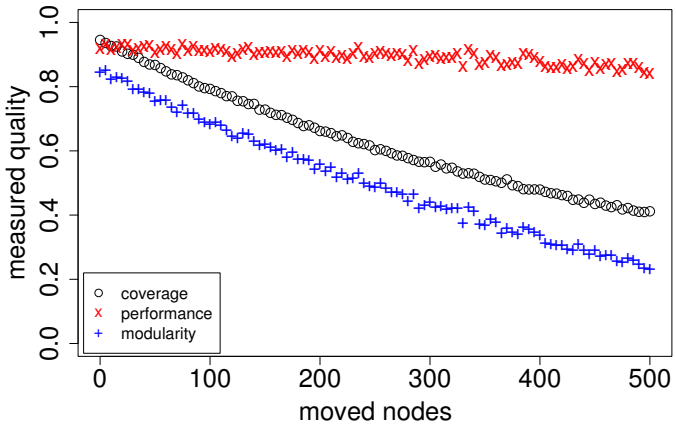
others mostly fail

Results. Figure 2.6.5 shows the result for all measures on this specific setup. As shown in Figures 2.6.5c and 2.6.5d, all *node-structural* measures hardly distinguish the two cases. Evaluating the *graph-structural* measures (Figures 2.6.5e and 2.6.5f), the intuitive behavior of Rand is verified. Furthermore, adjusted Rand and $\mathcal{ESD}$ distinguish both cases very well. The remaining *graph-structural* measures show the same behavior as their *node-structural* counterparts. Thus, the failure of van Dongen and the variation of information is confirmed. Unlike in Section 2.6.4.1, Fred & Jain fails on this setup. The unexpected behavior of the *overlap* and *entropy* based measures may be due to—as mentioned in Section 2.6.4.1—the fact that for $\varrho = 0.5$ and $\varrho = 2.5$ attractor graphs have a fairly regular structure. As shown in Section 2.6.1, the *graph-structural* versions of *overlap-* and *entropy*-based measures equal the *node-structural* variants for regular graphs.

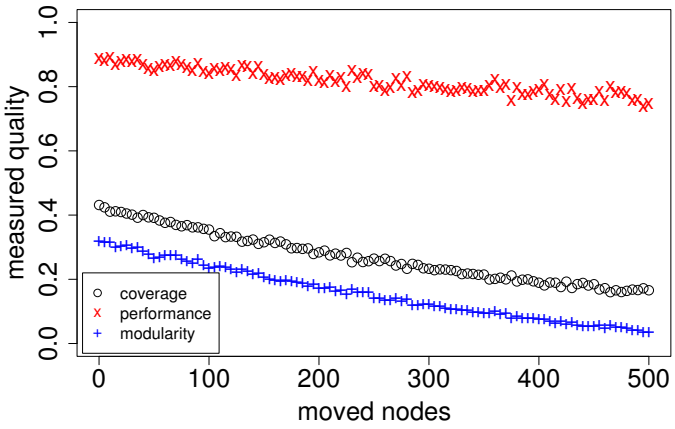
2.6.4.3 Real-World Scenario

example: email graph

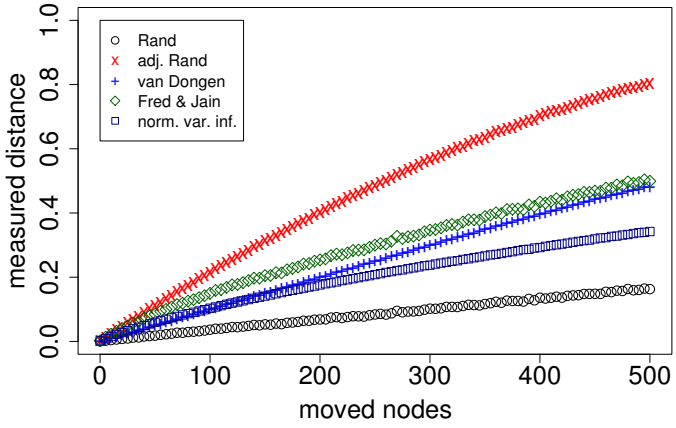
In this section, we discuss a real-world instance in order to illustrate the advantages of *graph-structural* measures over *node-structural* ones. As the input, we use an email graph (Figure 2.6.6) of Karlsruhe's Fakultät für Informatik, similar to that in Section 2.3.4.3, stemming



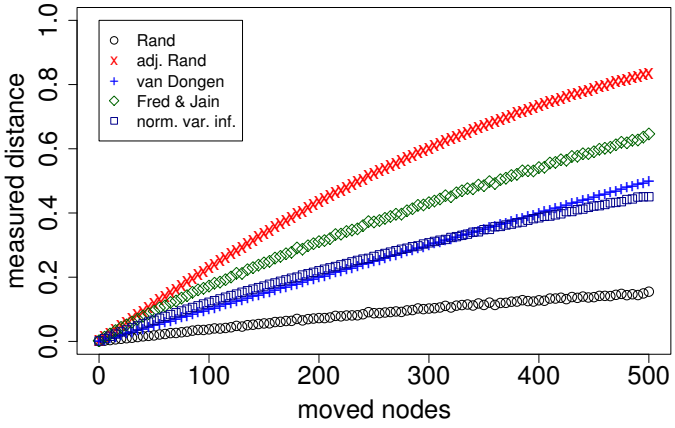
(a) type 1 quality of the clusterings



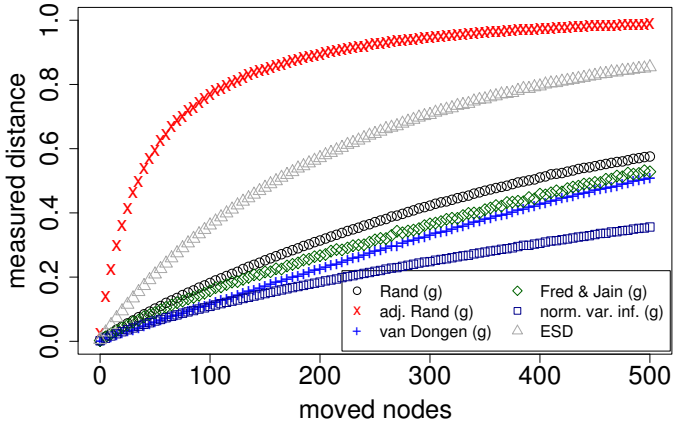
(b) type 2 quality of the clusterings



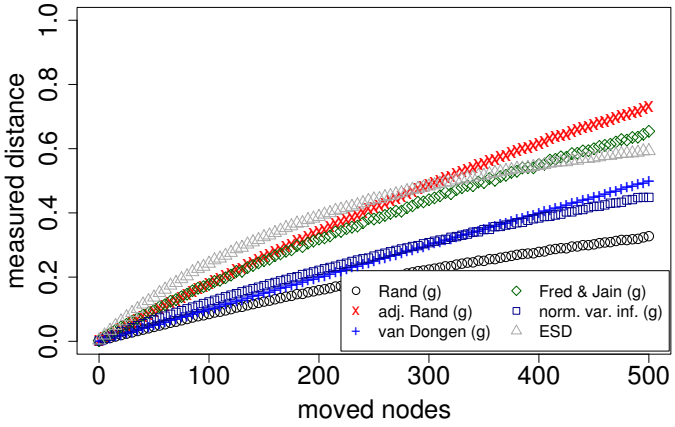
(c) type 1 *node-structural* distances



(d) type 2 *node-structural* distances



(e) type 1 *graph-structural* distances



(f) type 2 *graph-structural* distances

Figure 2.6.5. Results of the local minimization setup

from a different timespan. As a reference clustering, we group by departments. We additionally compute two clusterings by using the greedy *modularity* approach (see Section 2.2.5) and the MCL algorithm [213]. Table 2.6.1 depicts the scores achieved by the quality measures *coverage*, *performance* and *modularity*. Table 2.6.2 gives an overview of the measured distances between the abovementioned clusterings. We observe that the MCL-clustering is

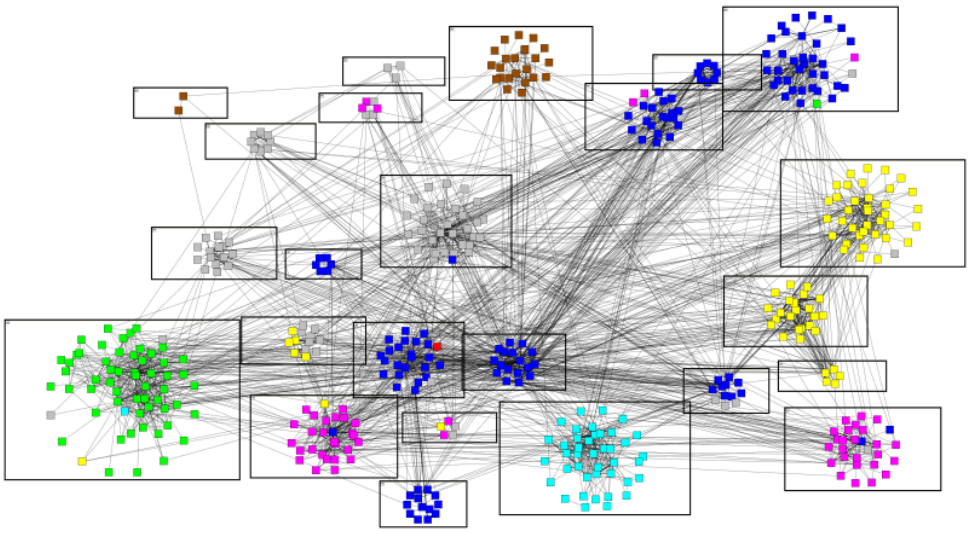


Figure 2.6.6. Karlsruhe email graph. Groups refer to the reference clustering, colors to the clustering obtained by the greedy *modularity* algorithm.

Table 2.6.1. Quality scores achieved by the reference clustering and those computed by the greedy approach and by MCL. The input is the Karlsruhe email graph.

	reference	greedy	MCL
coverage	0.8173	0.8634	0.8182
performance	0.9387	0.8286	0.9238
modularity	0.7423	0.6725	0.7282

graph-structural ⇒ *closer*

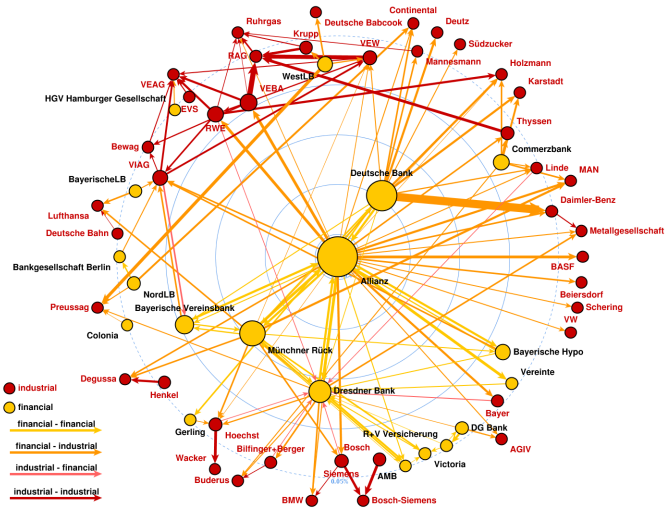
not as close to the reference than one could expect from the figures in Table 2.6.1. All *graph-structural* distance measures indicate a difference of more than 0.1. More interestingly, $\mathcal{ESD}$ yields a lower score than *graph-structural* adjusted Rand. For artificial data, the contrary is true (cf. Section 2.6.4). Most of our *graph-structural* measures indicate a lower distance between all clusterings than their *node-structural* versions. As all clusterings score similar quality values, and thus have quite a low distance with respect to quality, the *graph-structural* measures really incorporate qualitative aspects. Hence, they harmonize better with intuition than the purely *node-structural* versions. As discussed in Section 2.6.1, the *node-structural* Rand measure yields a very small value, due to the high number of small cluster. However, this drawback appears to be remedied by the *graph-structural* version.

Table 2.6.2. Measured distances between the reference and two computed clusterings. One clustering is obtained by MCL, the other one by the greedy *modularity* algorithm. The input is the Karlsruhe email graph (cf. Figure 2.6.6).

measure type	measure	reference vs. greedy	reference vs. MCL	greedy vs. MCL
quality	*modularity* difference	0.0697	0.0140	0.0557
node-structural	Rand	0.1233	0.0463	0.1466
	adj. Rand	0.5765	0.3555	0.6549
	van Dongen	0.2676	0.1834	0.3465
	Fred & Jain	0.3137	0.1794	0.3876
	variation of information	0.2425	0.1658	0.2904
graph-structural	Rand	0.1963	0.1305	0.2452
	adj. Rand	0.4689	0.2820	0.5730
	van Dongen	0.2435	0.1714	0.3215
	Fred & Jain	0.2828	0.1623	0.3581
	variation of information	0.2107	0.1427	0.2549
	$\mathcal{ESD}$	0.7325	0.5382	0.7796

Chapter 3

A Foray into Network Analysis



In several cases, network analysis geared towards the media has brought to the public an idea of what this field can accomplish. Notorious examples include the 9/11 terrorist network, the Enron breakdown email network and the network of shareholding among large German companies [133]. This visualization of the latter network, taken from [30], conveys information obtained by network analysis, including the degree of involvement (node size and edge width), importance (centrality of placement) and the affiliation to an economic sector (color).

Contents

Section 3.1

Preface to Network Analysis

Oi oi oi, me gotta hurt in 'ere
Oi oi oi, me small a ting is near,
Gonna bosh 'n gonna nosh
'n da hurt'll disappear.

(Traditional,
Uthden Troll, Revised Edition,
Magic: The Gathering, WotC)

The title of Chapter 3 is a misnomer. By any means, graph clustering is an integral part of network analysis and not a separate field. Thus, this brief chapter is actually about *other* techniques of network analysis and their connection to graph clustering. The rise of networks and of network analysis roots particularly strong in social network analysis. This area spawned many concepts and methods that today experience a renaissance, empowered by the ubiquity and availability of social and technical networks, especially on top of the internet (another, technical network) and by a surge of interest in large networks—*complex systems*—by a new breed of physicists [16]. Simply put, in the view of a graph clusterer the two driving incentives for taking a broader look onto network analysis are: (i) Can other techniques help to better understand, percept and interpret a graph clustering? (ii) Using other techniques as a preprocessing step, can we find more meaningful clusterings? The latter issue has already been pointed out in Section 2.1.5 and we will come back to it in Section 5.1.5 in the context of finding the "right" clustering algorithm for a real-world instance. In this chapter we will focus on the first issue, and thereby even dare to walk away from graph clustering for a moment.

3.1.1 Introductory Remarks

graph drawing conventions

The starting point of this foray is the very central question about how to represent a clustering in an informative and well-perceivable format. The fascinating field of graph drawing has established many reasonable criteria for such representations. Guidelines like *crossing minimization*, *small total edge length* or *angular resolution* in combination with constraints such as *orthogonal edge routing* or *grid placement* have led to a collection of very good drawing techniques for a multitude of applications; a good introduction is [77]. Even for clustered graphs, there are rigorous results about if and how drawings that comply with certain esthetic requirements are possible. As a reference and a source of further pointers, in [92] the authors consider *clustered graphs*[1] for which it is known that and how they can be drawn without edge crossings, cluster intersections and without edges passing through unrelated clusters. It is then shown that every *clustered graph* for which such a drawing is possible can actually be drawn in a way such that clusters are rectangles and edges are straight lines. Such graphs

[1] In graph drawing, a *clustered graph* is a graph and several nested and possibly incomplete clusterings, such that any node can be in a whole hierarchy of clusters. Thus, cluster overlaps cannot occur but clusters can be contained in other clusters and the clustering is *no* partition of V in general.

are called *c-planar*, and the complexity of determing whether a *clustered graph* is *c-planar* is open. *c-planar*

While such fundamental work is crucial, the networks this thesis aims at are much larger than those to which sophisticated layout methods can be applied to and range from hundreds to hundreds of thousands of nodes. Graph drawing conventions need to step back and become secondary criteria when dealing with thousands of elements. However, the result of a clustering algorithm still requires a good presentation, and while a table with simple listings of the members of clusters might suffice for some purposes, it does not foster any further understanding of the clustering and its background, let alone a single quality index. *large graphs ⇒ different focus*

Motivating Questions. Being more specific, we ask the following questions: Can we draw a large graph and a partition of the set of nodes—either stemming from a clustering or any other decomposition—in a way such that the partition itself and its properties, e.g., the sizes of the subsets, are well-readable? This means that each element of the graph should still be visible and that additional element- and group-level properties, such as the importance of a node or the connectivity of a subset to some other, should also be included, in order to deepen the understanding of the partition. Can such an approach be applied in practice, where a long term goal requires network analysis to guide the way towards ideas for a solution? Which properties of the network turn out to reveal the most useful information? We then turn to the *k-core decomposition*, a network analysis tool which we describe below. The *k-core decomposition* is regarded as an increasingly important structural property of a graph [80]—and yielded crucial insights regarding the last question. Can we actually construct a graph with a predefined *k-core decomposition*, and how well does this already describe a graph? *questions*

Answers in this Thesis. We start answering the above questions by describing *LunarVis*, a tool for analytic visualizations of large graphs. *LunarVis* focuses on properties of a partition of the set of nodes, e.g., a clustering. In a collaboration with the field of telematics we apply *LunarVis* as a means of visual network analysis and show how this tool can actually reveal those properties of a network which are a priori unknown but crucial for further analyses. Motivated by the observed importance of the *k-core decomposition* we prove bounds for the properties of this decomposition. We then design a random graph generator that complies with a predefined *k-core decomposition* and allows to additionally accommodate the hyped concept of *preferential attachment* [16]. *answers*

Parts of this chapter have previously been published in [19, 102, 116, 11, 12, 13, 14, 33, 34]. (We will point out the respective publications in the corresponding sections.)

3.1.2 Outlook

Recall that point (ii) in the introductory paragraph above is not dealt with, in this Chapter. Although we shall mention the issue in Section 5.1.5, I would like to stress what has already been touched in the outlook of Section 2.1: Network analysis can not only help in engineering clustering algorithms for speed and accuracy, it also has grand potential in serving as a meta-heuristic which recommends a good clustering algorithm for an instance at hand, and maybe even a good setting of parameters for it. The crucial point is to keep things simple, otherwise such an approach shall never see the light of practical application. *network analysis as a meta-heuristic*

3.1.3 Preliminaries

In contrast to the previous two chapters, we will not give an introduction to this vast field but only introduce two central tools. A very good book on social network analysis and its development is [94]. As a more technical reference and for further pointers we again recommend [46].

i-core

core number
core of a graph
coreness
i-shell
intra-, inter-shell

k-Cores. The concept of k-cores was originally introduced by Seidman [201] and generalized by Batagelj and Zaversnik [29]. We shall only use k-cores in an unweighted and simple context. Constructively speaking, the *i-core* of an undirected graph is defined as the unique subgraph obtained by iteratively removing all nodes of degree less than i. This is equivalent to the closed definition of the i-core as the set of all nodes with at least i adjacencies to other nodes in the i-core. The *core number* of a graph is the smallest integer i such that the $(i+1)$-core is empty, and the corresponding i-core is called the *core* of a graph. A node has *coreness* i, if it belongs to the i-core but not to the $(i+1)$-core. We call the collection of all nodes having coreness i the *i-shell*. An edge $\{u, v\}$ is an *intra-shell edge* if both u and v have the same coreness, otherwise it is an *inter-shell edge*. An example *core decomposition* is shown in Figure 3.1.1. Note that generally not all i-cores induce a connected subgraph. The *core decomposition* of a graph can be constructed in time $O(\max\{m, n\})$ with a simple algorithm [28].

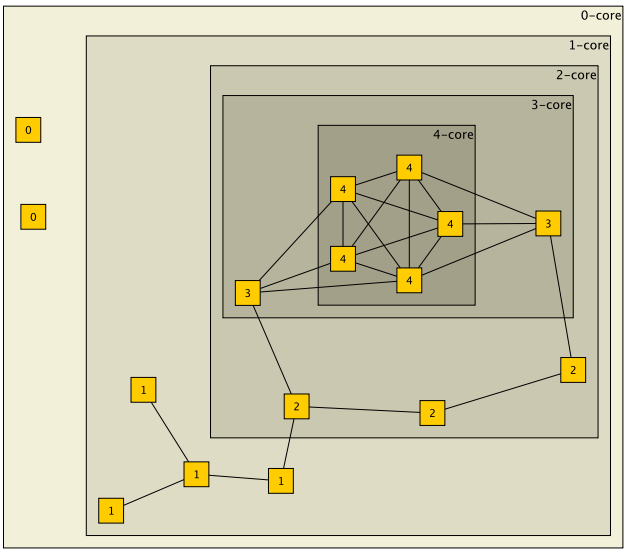


Figure 3.1.1. A k-core decomposition with 5 core shells. Note that the 3-shell is not connected, but the 4-shell again is connected.

centrality measure

Centrality measures. A *centrality measure* quantifies the structural importance of an element of a graph. The simplest such measure is the degree of a node. The literature has seen quite a few reasonable *centrality measures* for both nodes and for edges—often a measure can be applied to both, but we shall only measure nodes. For an overview we again recommend [46].

betweenness centrality
shortest-path betweenness

The *betweenness centrality* measures for a node v or an edge e of a graph G its importance for the set of all s-t-connections in G. Roughly speaking, two main variants of *betweenness* exist. The *shortest-path betweenness* [93, 21] is used when edge weights represent distances, it is defined as follows: For a pair of nodes $s, t \in V$, let σ_{st} be the number of different *shortest paths* (see Section 1.2.1) between s and t. The degree of involvement of a third node v in σ_{st} is measured by $\sigma_{st}(v)$ which is the number of *shortest paths* between s and t which pass through v, such that $\delta_{st}(v) := \frac{\sigma_{st}(v)}{\sigma_{st}}$ is the ratio of involvement of v. The *shortest-path betweenness* of node v is then defined as the sum of its ratios of involvement for all pairs of nodes:

$$c_{\text{betw}}^{\text{sp}}(v) := \sum_{s \neq v} \sum_{t \neq v} \delta_{st}(v) \tag{3.1.1}$$

Suppose now edge weights represent strengths or similarity, then the above notion has to be changed. Instead of considering *shortest paths* the *current-flow betweenness* [174] measures a node's involvement in an electrical network where current flows from s to t. An intuitive view without a precise unterstanding of electrical networks and *maximum-flows* in networks suffices to grasp the idea behind this definition. Suppose we let one unit of electrical current pass through a graph G by having it enter G by a wire attached to node s and let it escape G by a wire attached to node t. Then the balance $b_{st}(v)$ of a node v is $b_{st}(s) = 1$, $b_{st}(t) = -1$ and $b_{st}(v) = 0$ for all $v \neq s, t$. Analogous to c_{betw} we now measure the involvement of a node in terms of how much current passes through it, more precisely, for a node v we measure how much current passes through the edges incident to v—and thus also once through v for each pair of edges carrying incoming current and outgoing current. The *throughput* of v with respect to b_{st} thus amounts to

current-flow betweenness

throughput

$$\tau_{st}(v) := \frac{1}{2}\left(-|b_{st}(v)| + \sum_{e \sim v} |\text{current}(e)|\right) \quad . \tag{3.1.2}$$

Using this *throughput* we can now simply proceed as with c_{betw}, and define

$$c^{\text{cf}}_{\text{betw}}(v) := \frac{1}{(n-1)(n-2)} \sum_{s,t \in V} \tau_{st}(v) \quad . \tag{3.1.3}$$

For someone with Electrical Engineering as a minor it is astounding to see that such basic formulae, which for many decades students have been learning in their first week in the context of electronic networks, have such a late and pronounced impact on network analysis. In the following we will always use the appropriate version of *betweenness* without further discussion. We refrain from introducing other interesting measures, such as the *reach centrality* [128] of a node, which—roughly speaking—is a node's importance for *shortest paths*, and refer the reader to [46].

reach centrality

Section 3.2

LunarVis—Analytic Visualizations of Large Graphs

Before you criticize someone,
walk a mile in their shoes.
That way, you'll be a mile from them,
and you'll have their shoes.

(Jack Handey)

Tasks of network analysis with a very strict focus can often be done using tables, numbers or plots alone. However, the faintest hope for an exploratory nature of an analysis quite quickly calls for a visualization of the network. The observable trend to apply the concept of *network* to anything consisting of more than one entity—this ranges from perfectly reasonable to fairly absurd contexts—adds to this, as without visual exploration unknown networks can hardly be unterstood. Current research activities in computer science and physics aim at understanding the structural characteristics of large and complex networks such as the Internet [183, 55], networks of protein interactions [228, 141], social networks [82] and many others [176, 20]. A multitude of laws of evolution and scaling phenomena have been investigated [154, 26], alongside studies on community structure, e.g. [57], and traditional network analyses [46]. Heavily relying on mathematical models and abstract characteristics, many of these techniques highly benefit from, or even depend on feasible advance information about structural properties of a network, in order to properly guide or find starting points for an analysis. The design of adequate visualization methods for complex networks is a crucial step towards such advance information. Furthermore, due to the diversity of such analyses, customized visualizations concentrating on user defined structural characteristics are required. Along the lines of the more general issue in the field of information visualization, see e.g. [221], visualizations of large networks naturally suffer a trade-off between the level of detail and the visible amount of information. In other words, a detailed representation of a graph often antagonizes the immediate perceptibility of abstract analytic information.

In this section we propose *LunarVis*, a layout paradigm that tackles the task of detailed analytic visualizations for large graphs and their decomposition. Our approach incorporates the strengths of abstract layouts, while individually placing all nodes and edges, i.e. without hiding away potentially crucial details. Through sophisticated utilization of force directed drawing techniques and the neat design of an apt global shape—a (partial) annulus—our technique creates visualizations of networks that reveal analytic properties of decompositions alongside properties of the *shell* connectivity at a glance, on the one hand, and offer insights into the interior characteristics of *shells* on the other hand. An emphasis on either inter- or intra-adjacencies can easily be adjusted. The technique works in three phases. In the first, abstract phase, a network decomposition—e.g., a clustering—determines the general shape of the layout, defining and arranging the drawing bounds of each annular segment. The second phase initializes the drawing of individual nodes and estimates parameters and

the third phase determines the final layout by means of sophisticated force-directed methods. Our paradigm offers many degrees of freedom that can incorporate any desired analytic property, allowing for well readable simultaneous visualizations of complementary properties. Simple user parameters tune the focus of our visualizations to either inter- or intra-segment characteristics, and furthermore permit a scalable trade-off between the overall quality and the required computational effort. The idea of nicely and perceptibly drawing graphs is not new, however, research on large networks, with many conjectured mechanisms behind network growth, evolution and functional structure, inspired a new family of visualizations. One might call them analytic visualizations with an emphasis on abstract features and measurements, or simply *fingerprints*. Traditional paradigms of graph drawing are certainly still valid for such tasks, but have to find a compromise with new requirements. As an example, crossing minimization becomes secondary at best, when visualizing thousands of nodes with a layout that emphasizes network centrality. Thus, *LunarVis* is not a tool for investigating small-scale substructures or for purely esthetic, energy-minimal drawings.

Our work on *LunarVis* was motivated by a *fingerprint* layout made with LaNet-vi (see below). In a series of productive meetings with José Ignacio Alvarez-Hamelin, one of its authors, we conceived a new method with many ideas for improvement, which were then engineered until the technique reliably yielded informative and useful layouts. A less comprehensive version of this section was published in [116], based on joint work with Marco Gaertler, José Ignacio Alvarez-Hamelin and Dorothea Wagner. The name of our paradigm *LunarVis* has been inspired by the semblance of our visualizations to the shape of the moon, sometimes waxing, sometimes full, but always a nice sight.

Main Results

- We propose *LunarVis*, a new layout paradigm for drawing large networks, with a focus on decompositional properties. Numerous abstract features of the decomposition can immediately be recognized in the visualizations produced by *LunarVis*, while all elements are drawn. Our layouts offer good readability of the decompositional connectivity and at the same time are capable of revealing subtle structural characteristics. (Section 3.2.2)
- We employ an approach consisting of several concurrent and annealing force-directed algorithms for determining a node's position. (Section 3.2.2.2 and)
- The application of *LunarVis* in a number of domains produces informative layouts, sometimes even suggesting yet unknown properties for a taxonomy. (Section 3.2.3)

Future Work. A simplification of the definition of forces might speed up *LunarVis*, in particular, in very large networks, forces could be summarized. For such networks, an interactive zooming technique which is able to abstract certain visual entities even further would also be helpful.

Related Work. In the past, several layout techniques have been developed driven by the ambitious goal to properly visualize complex networks such as the Autonomous Systems (AS) network. Two important approaches are the *landscape metaphor* [31] and network *fingerprinting* [18], examples of which are shown in Figure 3.2.1 and Figure 3.2.2, respectively. Introduced by Baur et al., the former modifies a conventional layout technique by a framework of underlying constraints that are based on analytic properties. The global shape of the network is induced by the position of structurally important elements, which automatically conceal inferior parts. Thus, it reflects the "landscape" of importance, either in two or three dimensions. The latter approach, LaNet-vi [18] uses analytic properties to define a suitable global shape, which in this case consists of concentric rings of varying thickness, one for each level of the *core decomposition* (see Sect. 3.2.1.1). Then, the elements of the network are

landscape metaphor

LaNet-vi

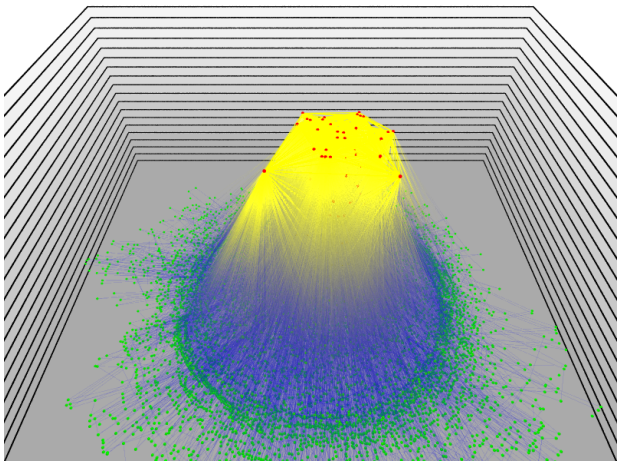


Figure 3.2.1. A 2.5-dimensional layout of the AS network, utilizing the *landscape metaphor* [31].

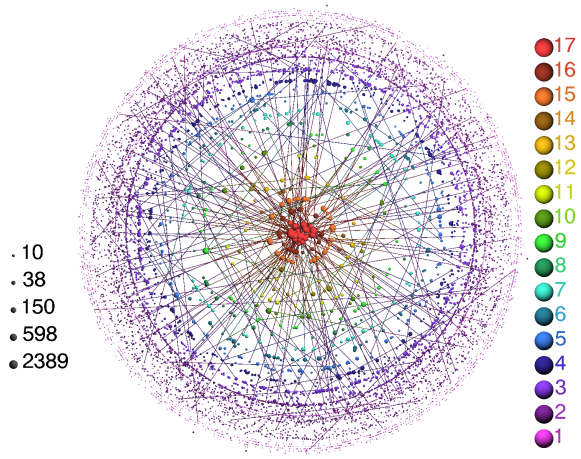


Figure 3.2.2. A *fingerprint* of the AS network made with the visualization tool LaNet-vi [18].

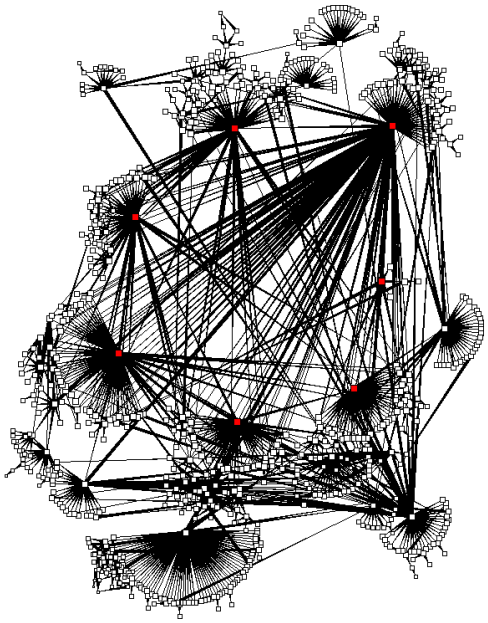


Figure 3.2.3. Visualization of the growth and topology of the NLANR caching hierarchy [137] with Plankton [3]

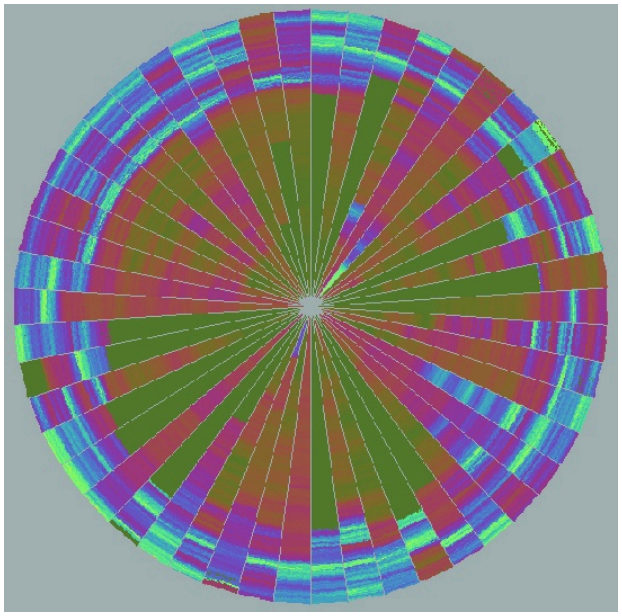


Figure 3.2.4. Circle Segments [147] are used for visualizing multidimensional data sets. Here, about 265000 data values are drawn.

placed within these bounds, while the overall readability is achieved by showing only a small sample of the edge set.

Plankton

Figure 3.2.3 is a visualization of the NLANR web caching hierarchy, created with the aged tool Plankton [3], which displays all nodes and edges of the NSF-sponsored web caching network. Although it has the look and feel of classic force-directed methods (for an overview see e.g. [77]), it exploits the strongly hierarchical nature of the network, and its relatively small size to directly determine a node's position. The low asymptotic complexity of the algorithm allows for an interactive emphasis of geographical or topological properties, and for the visualization of temporal evolutions. Figure 3.2.4 displays 50 stock prices from the

Frankfurt stock index over a period of 10 years. Thus, no actual graph is depicted, however, the drawing technique [147] from the field of information visualization is somewhat related to our approach, since it segments a circular drawing area. This pixel-per-value technique fills each segment with one dimension of the data (i.e., one stock item), starting from the inside and coloring pixels according to the stock value.

Circle Segments

The above techniques have been applied in numerous tasks, serving as an aide in network analyses. The method we present in the following synergizes assets of the above approaches and remedies a number of shortcomings in order to provide a layout technique that *fingerprints* a network (as LaNet-vi), but adds to this a much clearer visual realization of a number of analytic properties, thus offering a high informative potential. Before describing our visualization technique, we state a few definitions and introduce some preliminary conventions and concepts.

3.2.1 Preliminaries

3.2.1.1 Network Decompositions

Let $G = (V, E)$ be an undirected graph. We call a partition $P = \{V_0, \ldots, V_k\}$ of the set V of nodes a *decomposition* with *shells* V_i. Recall from Section 3.1.3 the definition of the *core decomposition*, as we borrow some of its nomenclature. Edges between or within *shells* are canonically called *inter-* or *intra-shell* edges, respectively. The set of *intra-shell* edges of *shell* V_i is called E_i.

shells

inter- and intra-shell

The choice of suitable network decompositions primarily depends on the field of application. In this section we focus on four different exemplary decompositions, *k-cores*, *clusterings*, by *reach centrality* and by *betweenness centrality*. The *betweenness centrality* of a node states, roughly speaking, how important it is for the set of all shortest paths through a network [93], *reach* is a similar concept used in transportation networking [128], see Section 3.1.3 for details. These decompositions are highly relevant as fundamental techniques for the analysis of large networks, such as protein network analyses [228], recommendation networks [57] and social sciences.

betweenness

reach

3.2.1.2 Reduction versus Abstraction

Visualizations of large networks usually suffer a trade-off between the details of shown elements and the amount of represented information. Widely known concepts resolving this are *abstraction*, as can be seen in Figure 3.2.5, and the *reduction* of data to specific *shells* or parts of interest, illustrated in Figure 3.2.6. While abstracted visualizations offer the best readability of these properties, much detail is lost, as in Figure 3.2.5. In contrast, zoomed visualizations as in Figure 3.2.6 allow for the exploration of small scale subgraphs and structural subtleties. We overcome this compromise by using the layout of an abstracted graph as a blueprint but still draw all elements. Our goal is the visualization of all nodes and edges in a manner both pleasing and informative on intra *shell* characteristics, in addition to revealing the characteristics of the given hierarchical decomposition. More precisely, we focus on properties like the size of *shells* and the connectivity within and between *shells*.

detail vs. overview

blueprint plus all elements

3.2.2 The Layout Technique

In the following we detail our construction technique for *LunarVis*. The general underlying shape of the layout is a (partial) annulus. Subgraphs, defined by some decomposition, are then individually molded into annular segments. The annulus has been chosen for three primary reasons, first, it offers immediate readability of hierarchies and decompositional characteristics. Second, it allows for an insightful segment-internal layout, and third, it provides a large area for the drawing of edges, permitting the perception of segment connectivity at a glance, which is a major focus of many applications. Roughly speaking, our approach divides up into three distinct phases, the first of which sets out the abstract layout attributes of the annular

annulus

annular segments

1. abstract layout

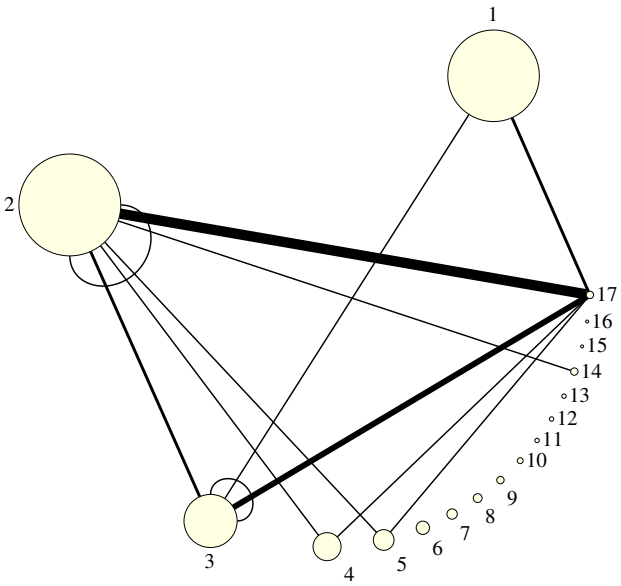


Figure 3.2.5. *Core*-abstracted version of the AS graph (May 1st, 2001). Each *core-shell* is represented by a node of size proportional to its number of AS nodes. Edges are induced by the number of inter-*shell* edges (light edges are omitted).

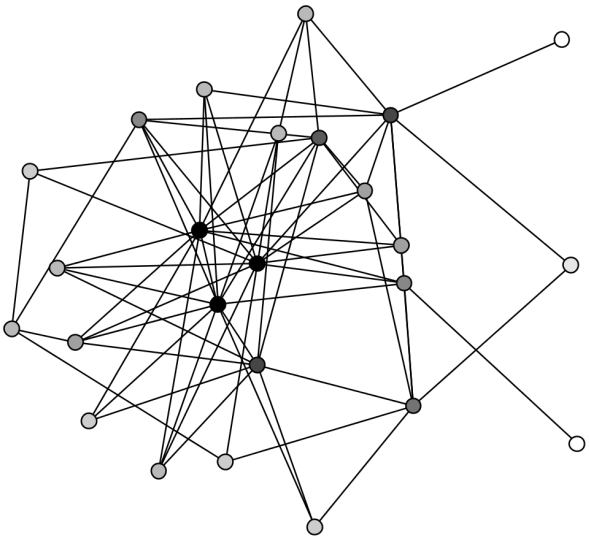


Figure 3.2.6. A reduction of the 16-*shell* of the left hand network, layouted with force-directed methods. Darker nodes have a higher degree within in this *shell*.

2. parameter estimation

3. force-directed layout

layout, such as the number of segments, their dimension and their placement. Based on these, a heuristic computation of suitable parameters follows, which will then be employed in the third and last step. This last, and by far the most intricate and computationally demanding step can be regarded as an iterative, segment-wise application of spring forces. These forces determine the final placement of each single node based on neighborhood attraction and repulsion both inside and between segments. In the end, we scale the annulus to the desired angular range and radial spreading and finally draw edges as straight lines with a high degree of transparency. Optionally, the size of a node and its color may serve as additional dimensions of information, yet ample use of these potentially overburdens a visualization. Algorithm 11 gives an overview of these three phases, which we describe in detail in the following sections.

Algorithm 11: LunarVis

Input: Graph $G = (V, E)$
Output: *LunarVis* Layout

```
1  Initialize abstract layout
2  Compute appropriate parameters
3  Initialize random node placement within segments
4  for i = 1, ..., ℓ_out do
5      forall shells s do
6          Project layout of s to middle square s̄
7          for k = 1, ..., ℓ_inter do  Apply inter-shell forces to s̄
8          for j = 1, ..., ℓ_intra do  Apply intra-shell forces to s̄
9          Project new layout of s̄ to annular segment s
10 Finalize and scale annulus to desired format
11 Draw transparent edges, color and resize nodes
```

3.2.2.1 Abstract Attributes

By any means, the informative potential of the our technique heavily relies on a suitable rough, abstract layout. We propose as the general underlying shape of the visualization an annulus, as shown in Figure 3.2.7. The *shells* s_i are lined up along a predefined angular range (here a full circle), placing the bottom (s_1) and the top *shell* (s_8) at the extremes. Thus, *shells* correspond to annular segments. User-defined properties then determine the individual dimensions of these segments, namely the *angular width* α_i and the *radial extent* r_i. In order to increase readability, small gaps β_i that separate neighboring segments can be included. The underlying annulus has an inner radius r_{in} and an outer radius r_{out}, which, together with the angular range, define the total drawing area. In our experiments, setting the annular segments to touch the inner rim and sizing them such that the largest *shell* also touches the outer rim, offered the best readability. For consistency, we let the number of nodes V_i per *shell* define the angular width and the number of intra-*shell* edges E_i define the radial extent throughout this paper, since these properties are generally of immediate interest. Molded into the underlying shape of annular segments, the *shells* can now be layouted individually.

s_i

α_i *and* r_i

β_i

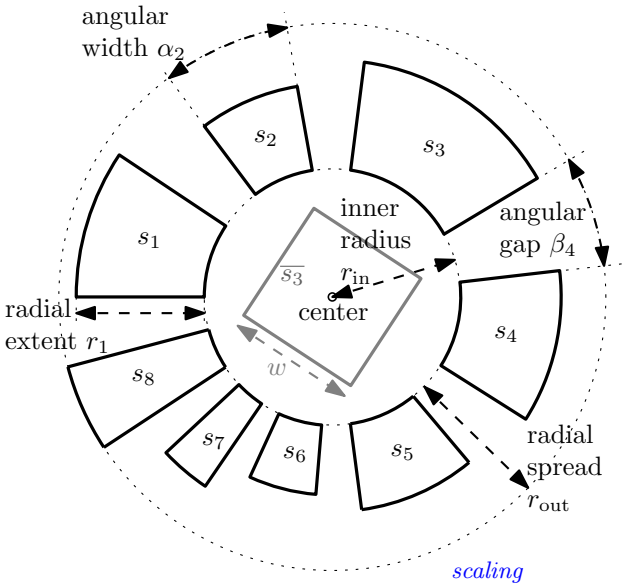


Figure 3.2.7. Overview of the annular blueprint of *LunarVis*. In this current iteration of line 5 in Algorithm 11, *shell* s_3 is layouted.

To give an impression of this step, and to point out the utility of an additional scaling function for the abstract layout, Tables 3.2.1 and 3.2.2 each show nine layouts of the same network, using different scaling functions for the radial extent and the angular width of a *shell*. As canonic scaling functions, we used the strictly monotonically growing functions *square root* and *logarithm*. The network is a snapshot of the AS network, decomposed into its *core* hierarchy. Individual nodes are left with a random placement, and the total angle is π. Linear scaling enables the immediate comparison of sizes, however, large values overshadow more subtle variations that do not become obvious without a logarithmic scaling of the radial extent. The inter-*shell* edge distribution is revealed by logarithmically scaling angular widths. Next, we describe how individual nodes are placed. For the sake of a better understanding we describe our parameter settings afterwards in Sect. 3.2.2.3.

3.2.2.2 Force-directed Node Placement

Placing the individual nodes is by far the most computationally demanding task. Simple strategies – random placement in the extreme – offer an easy recognition of the *shells*' shapes, however, more sophisticated techniques can additionally reveal the internal structure of the *shells* while requiring more time and storage. Based on the forces proposed by Fruchterman and Reingold [97] we use spring- and repulsion forces to iteratively have the nodes of each *shell* adjust their position as suggested by their adjacencies and their geometric neighbors. In the following we describe this procedure in detail.

As sketched out in Algorithm 11, our layout algorithm cycles through all *shells* a set number (ℓ_{out}) of times by lines 4 and 5. The nodes of a *shell* are then first subjected to inter-*shell* spring forces (repeated ℓ_{inter} times), thus moving towards their inter-*shell* adjacencies, and then, as a relaxational step, to intra-*shell* forces (repeated ℓ_{intra} times), see lines 7 to 8. To this end, we maintain a mapping of each *shell*, i.e. annular segment s_i, to a square $\overline{s_i}$ of size $w = 2/3 \cdot r_{\text{in}}$, centered at the origin and rotated such that it faces its original annular segment (see lines 6 and 9). This is illustrated in Figure 3.2.7 for the segment s_3. Forces are applied to

ℓ_{out}

ℓ_{inter}

ℓ_{intra}

$\overline{s_i}$

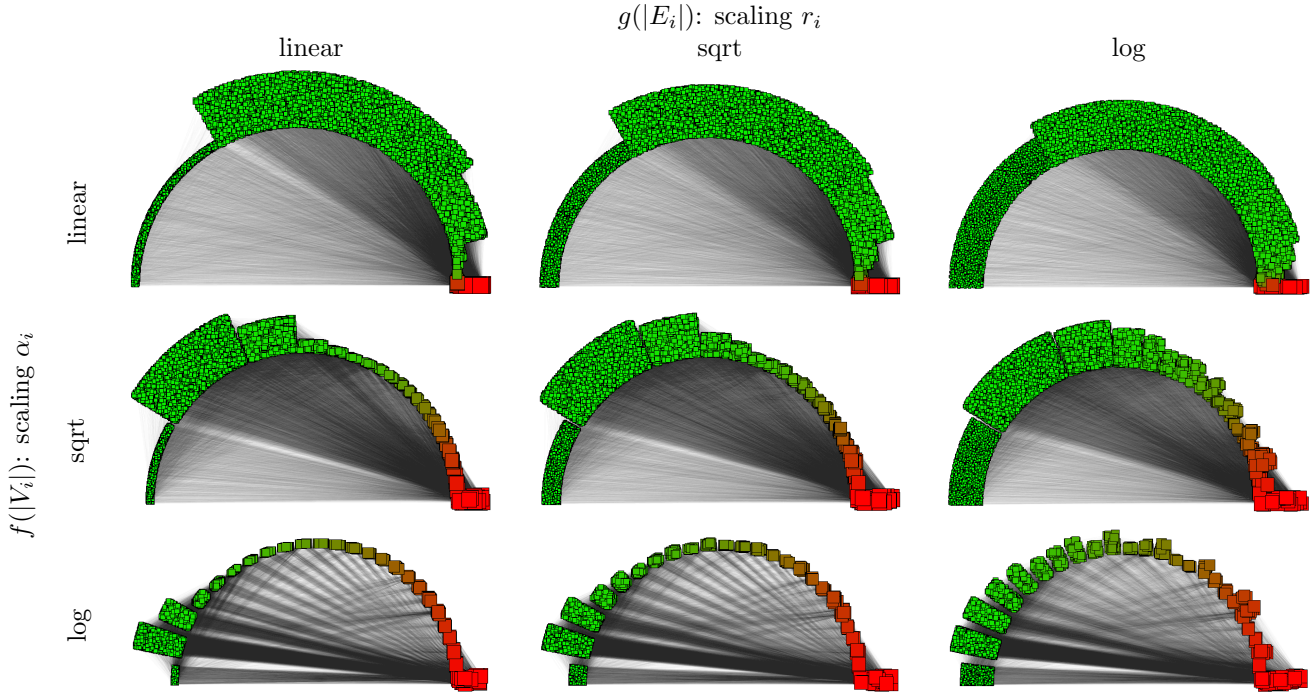


Table 3.2.1. Visualizations of the AS (1st March, 2005) using different scaling options for the abstract shape, i.e., the sizes of the annular segments. Color and node size both emphasize the *shell* index.

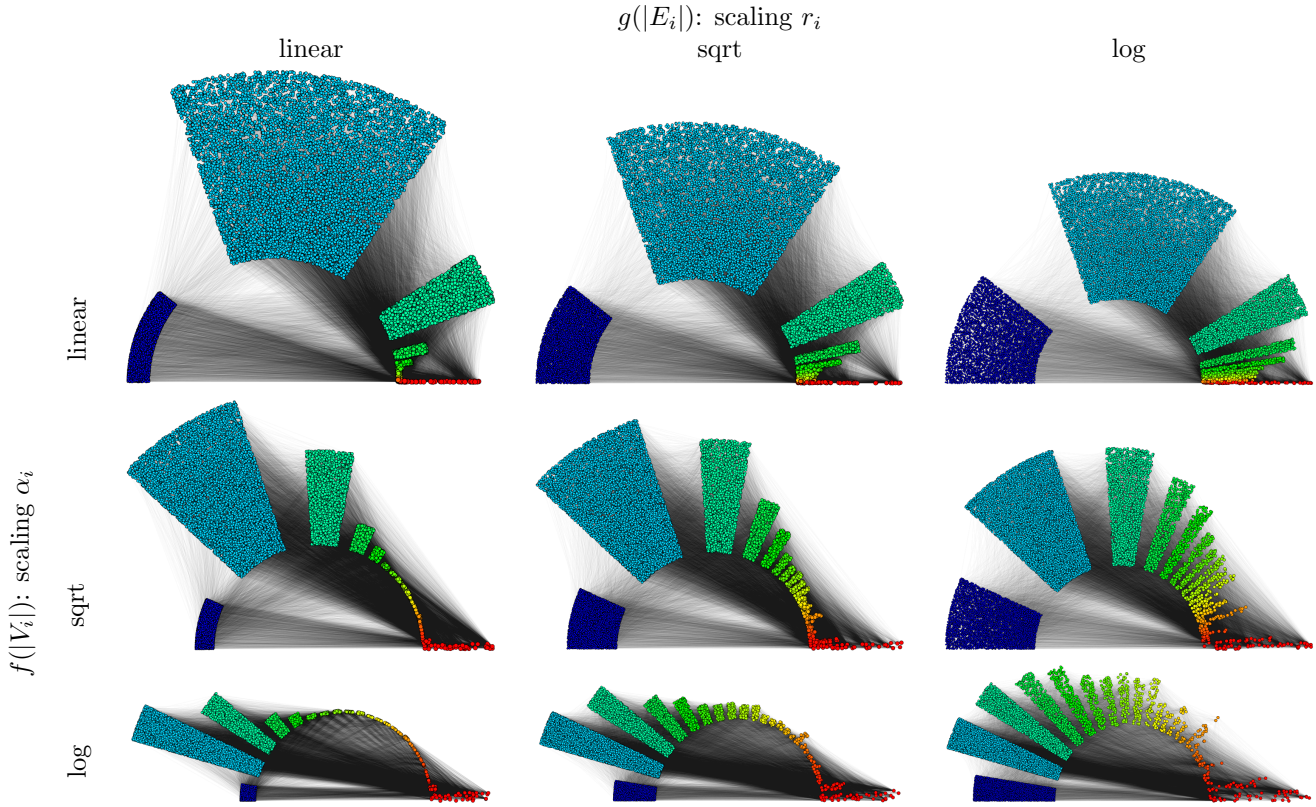


Table 3.2.2. Visualizations of the AS (1st March, 2005) using different scaling options. Color and node size emphasizes the *shell* index. Compared to Table 3.2.1 an alternative set of parameters for scaling nodes, colors and the outer radius has been chosen.

the copies of nodes in the square $\overline{s_i}$, and then, the new coordinates of nodes in $\overline{s_i}$ are mapped back to the annular segment s_i and its nodes are moved accordingly in line 9. Note that nodes in s_i themselves exert inter-*shell* forces on their copies in $\overline{s_i}$. Figures 3.2.8 and 3.2.9 illustrate the intention of this approach. First, note that a node coordinate $(x_{\overline{v}}, y_{\overline{v}})$ in a square shaped working copy $\overline{s_i}$ is obtained by transforming the circle coordinates (ρ_v, ϕ_v) in the annular segment s_i in a canonical way, such that the angular position ϕ_v of v within s_i is linearly mapped to the the x-coordinate $x_{\overline{v}}$ within $\overline{s_i}$, and the radial position ρ_v to $y_{\overline{v}}$. The rotation of $\overline{s_i}$ then aligns the y-axis of $\overline{s_i}$ with the middle axis (ϕ_{mid}) of s_i.

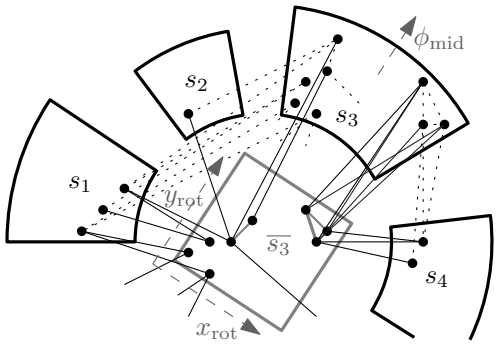


Figure 3.2.8. Forces for $\overline{s_3}$ (excerpt). Inter-*shell* forces are caused by edges that link $\overline{s_3}$ with annular segments (solid, black). Intra-*shell* forces are standard attraction and repulsion of nodes within $\overline{s_3}$. Dotted edges are irrelevant at this stage.

The crucial idea behind this setup is that inter-*shell* forces pull nodes towards a specific side of the square, thus indicating their linkage tendency, while intra-*shell* forces relax the resulting crowding and unmask community structure and disconnected components. In Figure 3.2.8, inter-*shell* forces draw the triangle of nodes in the right of $\overline{s_3}$ towards s_3 and s_4, while the nodes on the left, primarily being linked to other *shells* are pulled towards s_1, s_2 and other adjacencies. The subsequent application of intra-*shell* forces will keep the triangle grouped and separated from the rest, and will disperse and relax the disconnected nodes on the left.

concurrent forces

preferred placements

The areas of s_3 in Figure 3.2.9 roughly sketch out where nodes, with a majority of adjacencies in *shells* as indicated, are drawn by inter-*shell* forces, before intra-*shell* forces relax the layout. The size and placement of these areas are induced by the abstract layout of the annular segments, see Figure 3.2.7 for comparison. This fuzzy segmentation of each *shell* allows for a sophisticated interpretation of a node's position. Needless to say, we augmented our force-based algorithms with several well known techniques, such as soft clipping [97] to guarantee containment within *shells*, sentinel nodes that uncrowd segment borders [97] and an increased sluggishness of nodes with high degree [96]. However, (anti-)gravitational forces as well as simulated annealing [64], a randomized node ordering or an impulse history [96] yielded no substantial increase in quality, since that our technique does not aim at a highly optimized local layout. Although we observed acceptable convergence behavior and independence from the initial placement, we apply a simple exponential cooling, such that the movement of nodes is increasingly slowed. This proved necessary since certain constellations of adjacency can result in stubborn oscillations, especially if intra-*shell* forces are used purely relaxational.

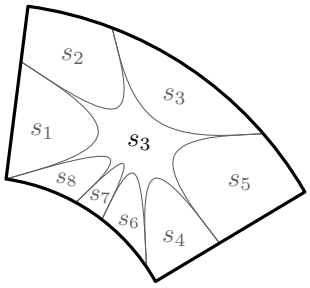


Figure 3.2.9. Preferred node locations

An important observation is, that applying inter- and intra-*shell* forces at the same time naturally encourages force equilibria, but does not allow for a structurally targeted analysis. On the contrary, the separate application of inter- and intra-*shell* forces allows for a user-defined emphasis on either *shell*-internal properties or global connectivity.

no equilibria but emphasis

3.2.2.3 Parameters

Heuristic or experimental assessment of parameters is inevitable when using customized force-directed methods. We base our forces on those proposed by Fruchterman and Reingold [97]. Alternative force models as proposed e.g. by Eades [77] or Frick et al. [96] did not prove more suitable but increased the running time, partly due to the fact that equilibria are not enforced.

For intra-*shell* forces we set the base spring length to $C_i \cdot \sqrt{(\text{area}/\#\text{ vertices})}$. The factor C_i turned out to yield best results, when set to a function negatively linear in the ratio of vertices to the number of edges in *shell* i, i.e. dense *shells* require boosting the intra-*shell*

spring length

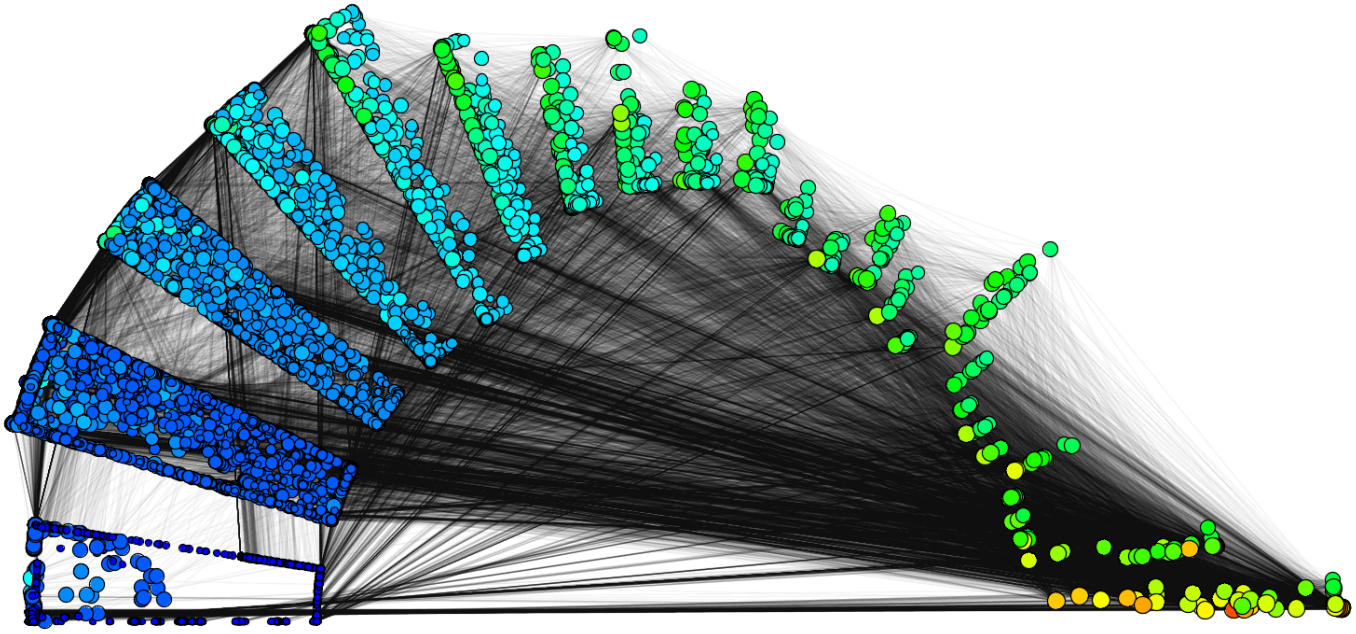


Figure 3.2.10. A snapshot of the AS network taken at the 01.01.2006, decomposed by *k-cores*. Nodes with a high (low) degree are colored blue (red) and the area of a node is proportional to its *betweenness centrality* (all on a logarithmic scale). We chose a half circle for the total angular range and set the maximum *shell* at the right end.

sentinel nodes

global repulsion

w

ℓ_*

spring length. For the artificially introduced sentinel nodes, which enforce a repulsion from *shell* edges, we found that placing them 5% beyond the boundary and setting them to a multiplicity of 10 works well. Depending on the decomposition, global factors for repulsion forces and spring lengths between 1 and 1.5 and 1.2 and 1.5, respectively, worked best. In fact, these two parameters were the only ones that required adjustment. Our inter-*shell* forces work with a base spring length of half the inner radius. Both the spring length and the spring force hardly needed additional tuning. Moreover, setting the edge length w of the squares $\overline{s_i}$ to significantly smaller values than $2/3 \cdot r_{\text{in}}$ blurred inter-*shell* forces, while much larger values exaggerated their range of effect.

As mentioned above, the iteration counters ℓ_{out}, ℓ_{inter} and ℓ_{intra} are pure user parameters, since these govern the interaction and the emphasis of intra-*shell* and inter-*shell* aspects. In fact, surprisingly low iteration numbers often proved better results than high numbers. As a rule of thumb, the following settings are a good starting point: $\ell_{\text{out}} = 10$, $\ell_{\text{inter}} = 10$, $\ell_{\text{intra}} = 5$. In the majority of drawings we used the logarithm for most scalings, as it copes best with power-law distributions and generally dampens overshadowing maxima.

3.2.3 Results

In the following, we present a selection of visualizations drawn with the *LunarVis* technique. All visualizations offer many immediate insights. Nevertheless, knowledge about the drawing process, i.e. how nodes are placed, allows for a more structurally oriented interpretation. For computing our drawings, we used one core of an AMD Opteron 2218 processor clocked at 2.6 GHz, with 1 MB of L2 cache, running SUSE Linux 10.1. Our non-optimized development implementations in Java required drawing times between a few seconds and several hours, depending on the chosen number of iterations and the size of the network.

AS network

Figure 3.2.10 reveals numerous characteristics of the *core decomposition* of the AS network at a glance. The well investigated fact that all *shells* primarily link to the *core* is immediately obvious, alongside the observation that the internal communities of the first 5 *shells* are well

interconnected (strong connectivity near the outer rim), but not those of other *shells*. To name just a few subtle facts visible in this drawing, note that mid-degree nodes can already be found in the 3-*shell*, that nodes with low *betweenness centrality* are exclusively found in low *shells* while the opposite is not true, and that in low- to mid-*shells* nodes with higher degrees primarily link to lower *shells*, as they sit on the upper left.

low shells interconnected
degree ∼ shell
low betw. ⇏ low shell

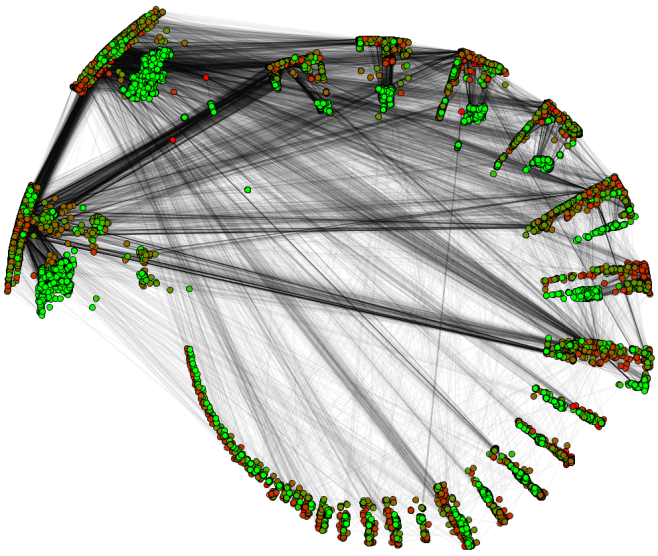


Figure 3.2.11. The AS network, decomposed by a clustering. Nodes with a high (low) *betweenness* are colored red (green).

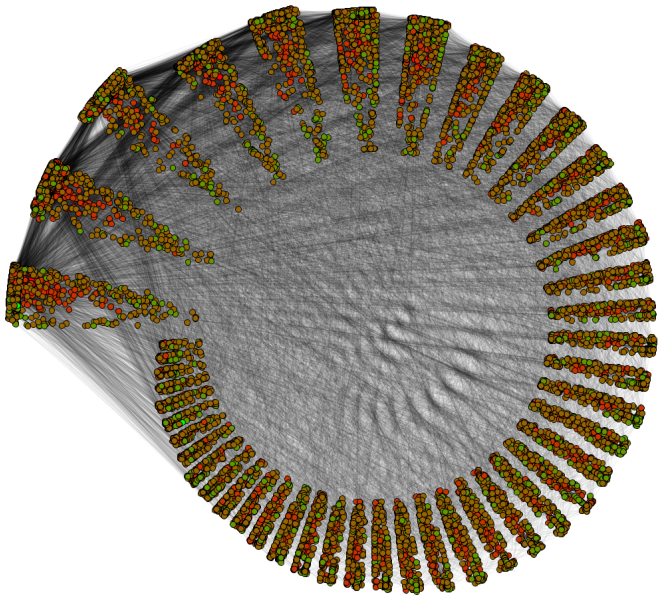


Figure 3.2.12. A network created with BRITE [160], designed to emulate the AS topology. All parameters are set as in Figure 3.2.11.

For Figure 3.2.11 and 3.2.12 a full annulus has been chosen due to the high number of *shells* (56 and 45). Figure 3.2.11 diplays the AS network, decomposed by community structure that has been identified by a greedy modularity based clustering algorithm [57]. The clusters are sorted by size. Figure 3.2.12 shows the same decomposition for a topology with the same number of nodes and edges, created with BRITE [160], an AS topology simulator. Quite clearly, BRITE fails to feature any of the peculiarities the AS network exhibits, such as high inhomogeneity in community sizes, the large number of tiny clusters or the fact, that most *shells* are almost exclusively connected to the two largest *shells*. An analysis yields clustering coefficients of 0.002 and 0.375 for BRITE and the AS network, respectively, and transitivities of 0.011 and 0.001, which agrees with these observations.

AS vs. BRITE decomposition. = clustering

BRITE fails

Figures 3.2.13-3.2.15 are drawings of AS network snapshots from spring 2002, 2004 and 2006, respectively. In all three drawings, *k-cores* are used for decomposition and color indicates a node's degree. In order to separate *shells* well but still keep the hierarchy obvious, we chose an exemplary total angle of 80% of a full circle. Several well known evolutionary facts about the AS network can be observed from these three drawings. To name a few, note the general densification of the network, the increasing depth of the hierarchy, the rather stable relative *shell* size with respect to the hierarchical position and, an observation yet to be investigated, a potential transition from a growing $(\max -1)$-*shell* to its merge with the max-*shell*.

evolution of AS core-hierarchy

increasing depth, stable rel. sizes

Figure 3.2.16 illustrates the *core decomposition* of an email network as described in Section 5.1.1. The nodes represent computer scientists at KIT, color coded by their department and sized by their *betweenness*, and edges are email contacts over a period of eight months. As an exception, we used the sum of degrees for the radial extent with a square-root scaling for this *LunarVis* layout. From the multitude of observable features we point out the fact

email graph

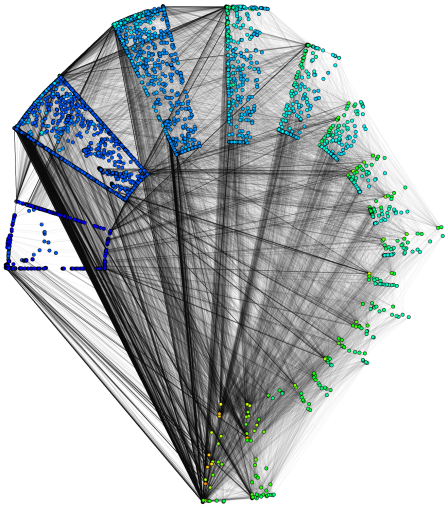


Figure 3.2.13. AS netw., spring 2002, *core*-decomposed

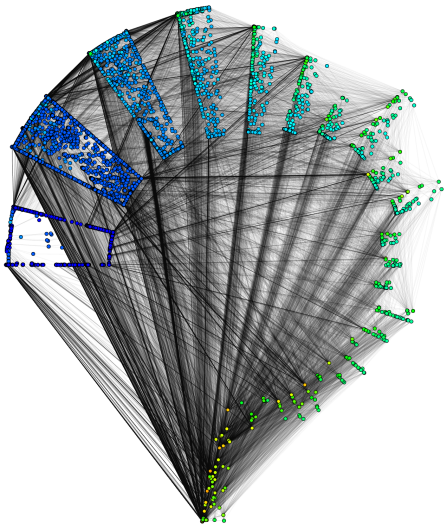


Figure 3.2.14. AS netw., spring 2004, *core*-decomposed

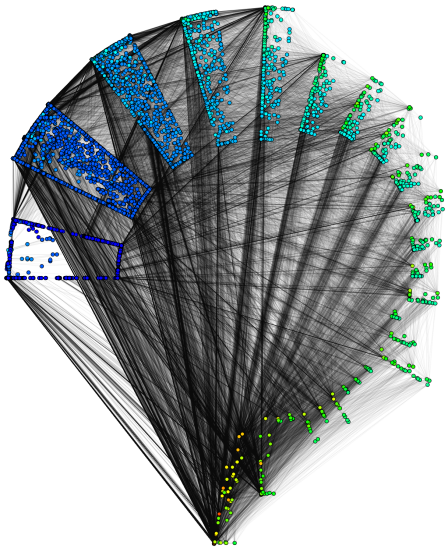


Figure 3.2.15. AS netw., spring 2006, *core*-decomposed

decomposition = external data

high ℓ_{intra}

that community structure within departments is corroborated by the groupings in the top *core-shells*. As an example, the dark blue department, although being well interconnected (gathered), seems to have many contacts to lower *shells*, thus it sits at the inner rim of *core* 17. In the following two large Figures 3.2.17 and 3.2.18, the email network has been decomposed in a more intuitive way by the structure of the department of computer science which is divided up into a number of institutes, that now make up the annular segments. Figure 3.2.17 focuses on the structure inside each institute via a high value of ℓ_{intra}, yielding nicely filled

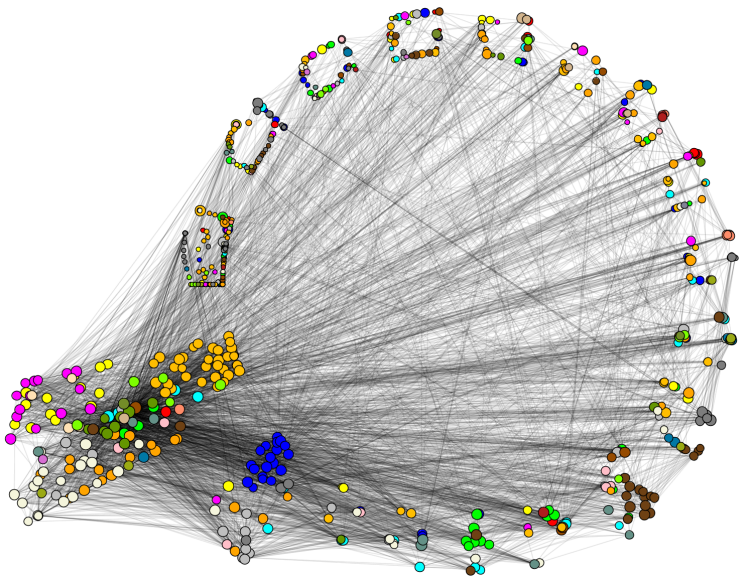


Figure 3.2.16. Email network of the computer science department at KIT. Nodes and edges represent scientists and email contacts, respectively. The nodeset is decomposed by *cores* and colored by department. The size of a node reflects its *betweenness centrality*.

segments and clear neighborhoods whithin. In most segments the chair and the secretary of an institute are the most prominent and central nodes. In contrast, for Figure 3.2.18 a higher value of ℓ_{inter} has been used to give more weight to the relations between institutes, such that the internal structure still relaxes the layout in a meaningful way, but contacts between institutes dominate the node positioning process.

high ℓ_{inter}

Modern algorithms for route planning exploit numerous characteristics of road graphs for efficient shortest path computations, for an overview see, e.g., [217]. Figures 3.2.19-3.2.22 display road maps of the Czech Republic and of the city of Munich, provided by PTV AG for scientific use, and Figures 3.2.23-3.2.24 display the European network of railway connections, provided by HAFAS. On the left hand side *betweenness centrality*, indexed into eleven logarithmically scaled intervals, served as the decomposition, and the figures on the right hand side are decomposed by *reach centrality* [128], colors are used vice versa. The degree of a node is reflected by its size. The stunning similarity of all corresponding drawings indicate that transportation networks share strong characteristics with respect to both *reach* and *betweenness*. However, several details can be observed that reflect intrinsic differences between these networks. Towards a taxonomy for transportation networks we can immediately observe that the railway network has very few hubs, both with respect to *betweenness* and *reach*. These are mainly capitals that, additionally, have exceptionally high degrees. The general correlation between *reach* and *betweenness* (color versus *shell* index) corroborates the fact that railroads constitute a so-called *scale-free* network. This does not apply to either road network, which is due to the fact that road networks tend not to have unique shortest paths – recall Munich's surrounding autobahn and Luxembourg's rural nature. The road networks strongly resemble each other, however, observe that in Munich, nodes of both maximum (autobahn segments) and minimum (residential dead-end streets) *betweenness* have a rather small degree. This cannot be observed in Luxembourg, where only nodes of minimum *betweenness* have an exceptionally small degree. From the facts revealed by the edge connectivity, note that hardly any peripheral nodes are adjacent to nodes of maximum centrality.

route planning

decomposition = betweenness

taxonomy

reach $\sim$ betweenness

road networks $\neq$ railway networks

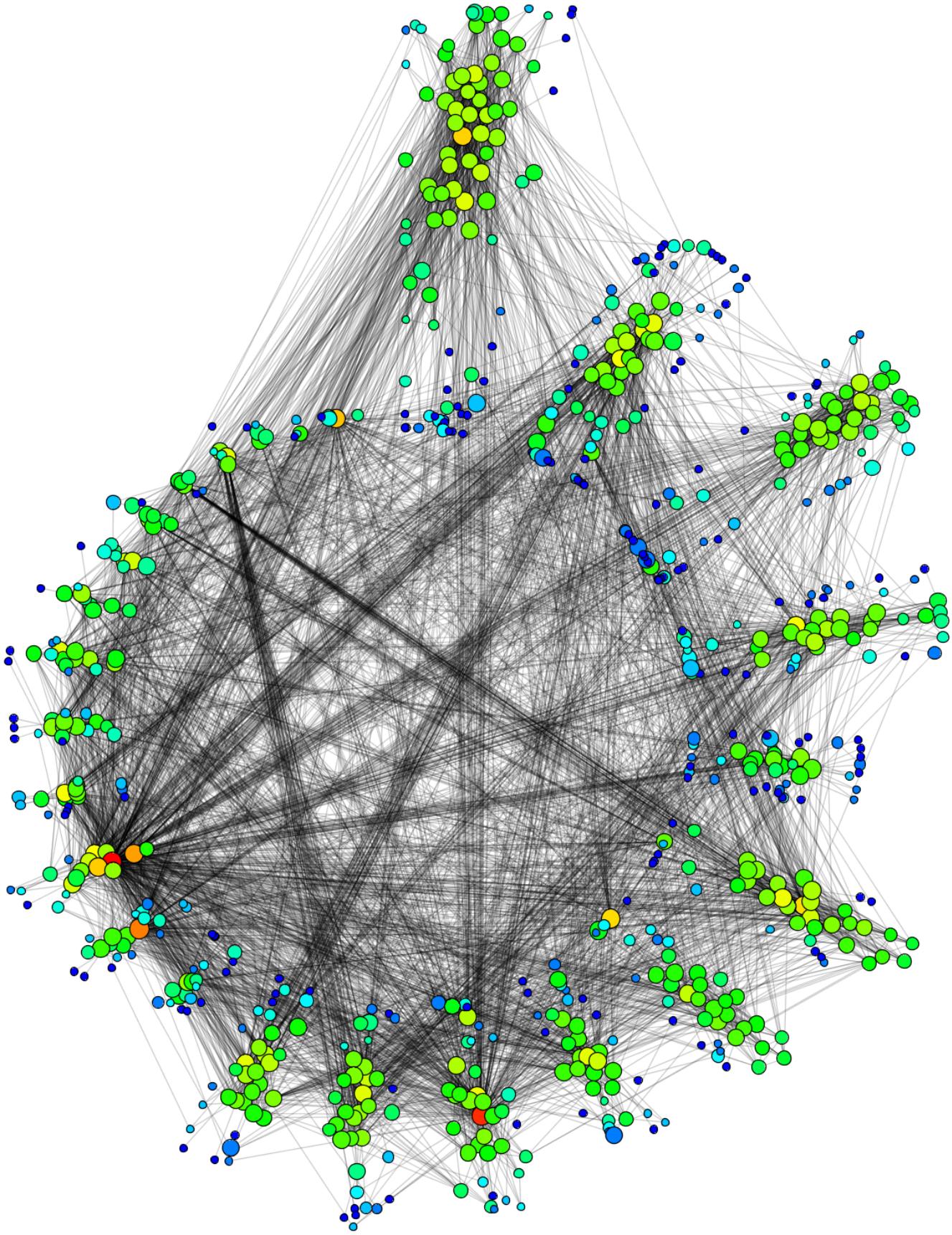


Figure 3.2.17. This is an alternative visualization of the email network in Figure 3.2.16. The network is decomposed by departments, with the largest one pointing upward. Color indicates the degree of a node, with red representing a high degree, while the size of a node indicates its *betweenness*.

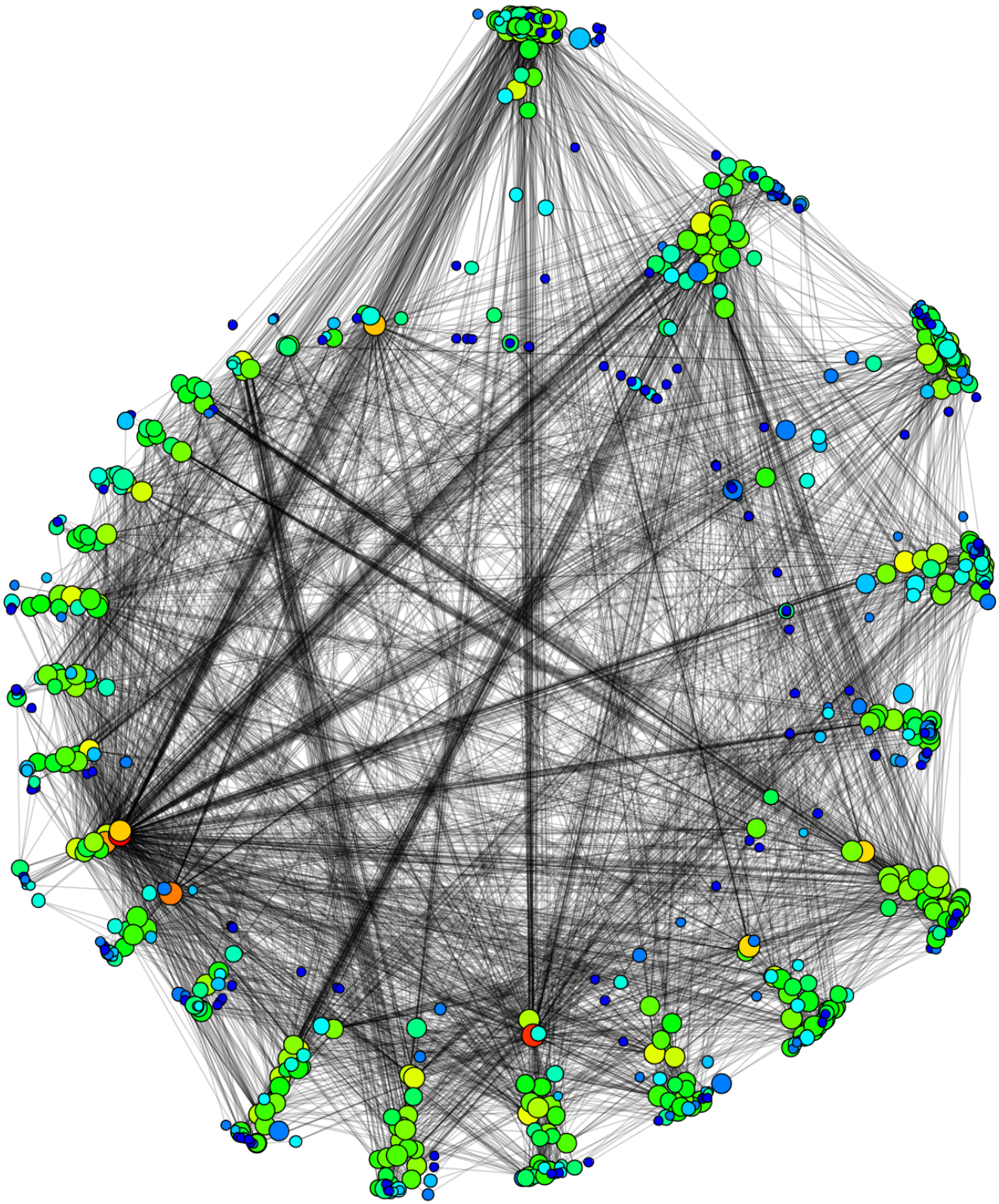


Figure 3.2.18. Another alternative visualization of the email network in Figure 3.2.16, as on the previous page. This time the emphasis has been set on inter-*shell* connectivity. Clearly, most departments are well connected, since most nodes sit in the back of their segment, especially in large departments. For large to medium sized departments, nodes of small degree and *betweenness* are the exception, alongside one or two nodes of large *betweenness* that seem to serve as bridges to other departments. Note that the two red nodes (very large degree), sitting near the inner rim, are well connected to many other departments.

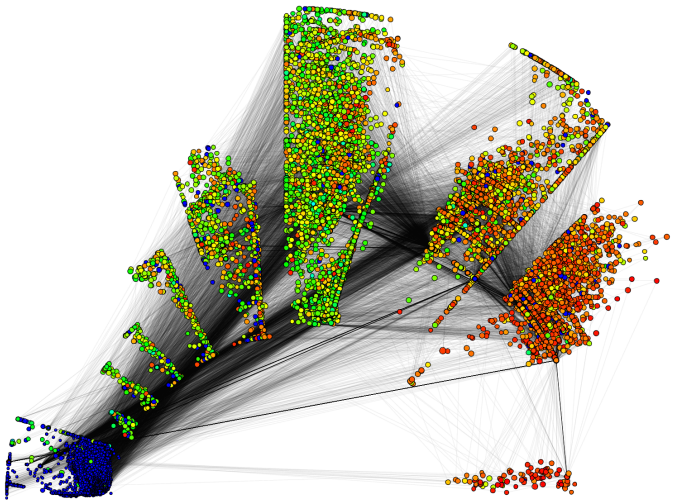


Figure 3.2.19. Luxembourg roads, decomposed by *betweenness*, color indicates *reach*.

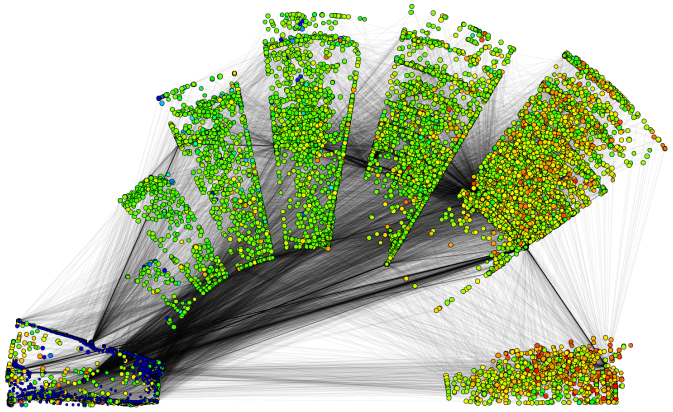


Figure 3.2.20. Luxembourg roads, decomposed by *reach*, color indicates *betweenness*.

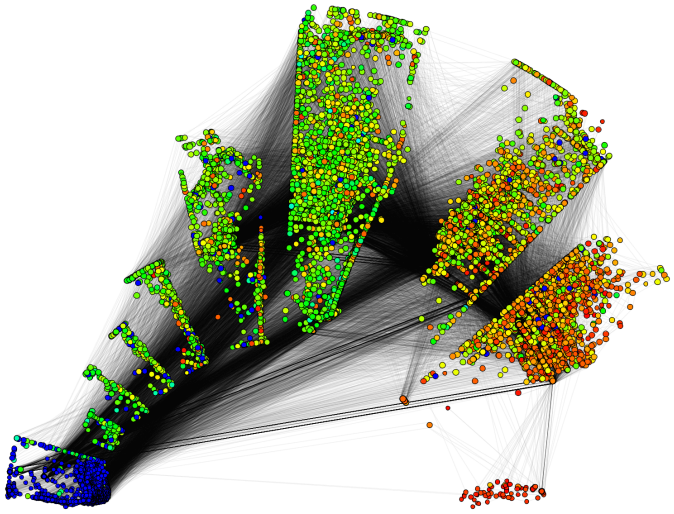


Figure 3.2.21. München roads, decomposed by *betweenness*, color indicates *reach*.

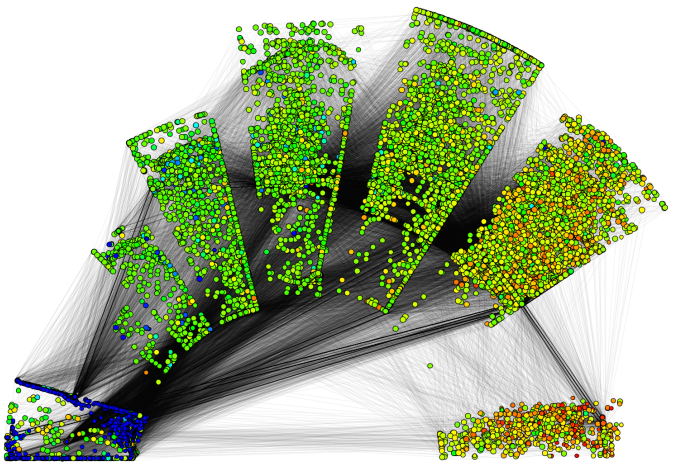


Figure 3.2.22. München roads, decomposed by *reach*, color indicates *betweenness*.

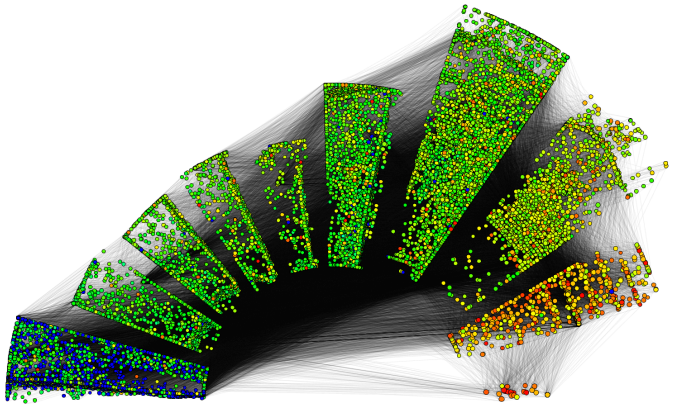


Figure 3.2.23. European railroads, decomposed by *betweenness*, color indicates *reach*.

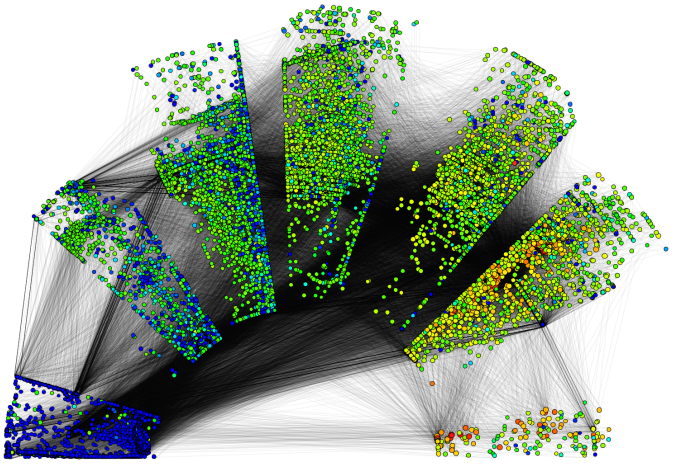


Figure 3.2.24. European railroads, decomposed by *reach*, color indicates *betweenness*.

Section 3.3

Overlay-Underlay Exploration Driven by Analytic Visualizations

I love the smell of bat guano in the morning.

(Vaarsuvius,
The Order of the Stick,
strip #20)

MANY APPLICATIONS CONSTITUTED THE DRIVING FORCE behind the results of the previous section on *LunarVis*. One particularly interesting application concerns the analysis of *overlay*- and *underlay* correlations in the Internet, and is described in this section. By virtue of its case-study character, this section thus slightly differs in structure from the rest of this work.

In recent times, the design of many real-world applications has changed from a monolithic structure to modular, yet highly customizable services. Network implementations from scratch are usually too time-consuming and expensive, and thus, these services are superimposed on some already existing *underlay* infrastructure as an *overlay*. A well-known example arises in logistics. The highways and streets we use everyday constitute a huge transportation network. However, traffic in this network is far from structured. In fact, countless companies and institutions rely on this network to accomplish their regular shipping of commodities and services, and by doing so, they cause the traffic on the road network to develop in certain patterns. In technical terms the road network constitutes an *underlay network*, while the commodity exchange network of a set of companies, implicitly building upon this network, forms an *overlay network*. The *overlay* network uses the *underlay* to actually realize its tasks. Another *underlay* network of prime interest is the Internet, which serves as the workhorse of countless data transfers, multimedia services and filesharing protocols. Almost anytime we use the Internet, we participate in some *overlay* network that uses the physical Internet, comprised of routers, links, cables, wires, to actually transmit the data packets. In turn, the Internet itself started out as an *overlay* built over the telephone network *underlay*. Within the Internet, a particular breed of *overlays* that has received a lot of attention lately are peer-to-peer (P2P) applications [204], which range from file-sharing systems like Gnutella and Bittorrent, to real-time multimedia streaming and VoIP phone systems like Skype and GoogleTalk. Clearly, there is a crucial interdependence between *overlay* and *underlay* networks. In particular, the emergence of *overlay* networks heavily affects and poses new requirements on the *underlay*. The major advantage of *overlays* is that they provide high-level functionality while masking the intrinsic complexity of the *underlay* structure. However, this abstraction entails a certain trade-off, namely independence versus performance. To gain a deeper understanding of the interdependency between the *overlay* and the *underlay*, this trade-off needs to be included in in the corresponding analysis.

In fact, the long-term goal behind this study is much more far-reaching. Deutsche Telekom Laboratories in collaboration with TU Berlin are considering a so called *oracle* which mediates

between peer-to-peer applications and a provider, in order to arrive at a mutual advantage in terms of load and performance. Although this fascinating project is actually taking shape[2], we shall in the following focus on our modest part in it, the preliminary study on the peculiarities of Gnutella's load on the Internet. On the one hand, our analysis, roughly speaking, points out that and how Gnutella's topology differs from randomly generated networks that mimic the principles and prerequisites of Gnutella; this even leads to sound refinements of the simulation which let us better understand the real-world instance. On the other hand—which in this work is the more important point—this section showcases how the methodology of analyzing networks by analytic visualizations offers a powerful and flexible tool.

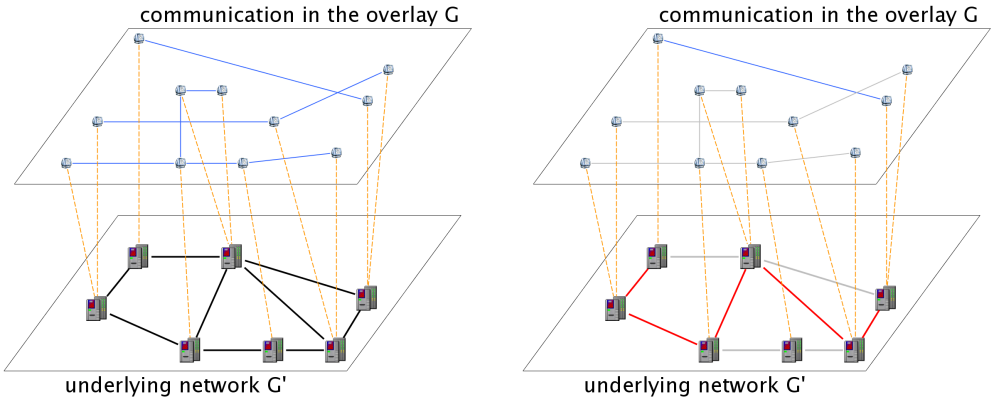
The work in this section would not have been conducted without the admirable efforts of my former colleague Marco Gaertler to incite and press ahead with our collaboration with the group of Anja Feldmann and Vinay Aggarwal. Our collaborators moved from TU München to Deutsche Telekom Laboratories / TU Berlin during our work, which was conducted within the FET Open Project "DELIS"[3] of the European Commission. Initially we were unsure about a platform for presenting our work, which we first cast into a technical report [11]. We were then surprised to receive an outstanding paper award [12]. Having finished *LunarVis*, an improved tool for visual analysis for the task at hand, shortly later, this then led to our work [13] appearing in the proceedings of the final DELIS[3] workshop and in the list of "stories of success"[4], shortly followed by a journal inviting a revised version [14] of our contribution to an issue on visualization-driven analysis. Finally, recognition![5] Most of the content of this section has been published in one of the above works which are based on joint work with the abovementioned coauthors and with Dorothea Wagner.

Main Results

- We introduce a theoretical model for *overlay-underlay* analysis using graph theoretic concepts. (Section 3.3.1)
- In a case study which compares measured Gnutella to simulated random *overlay* communication, we showcase how analytic visualizations, *LunarVis* in particular, help to identify key characteristics of Gnutella. (Section 3.3.3)
- Gnutella is different from random *overlay* communication in specific ways. (Section 3.3.3.2)
- Our observations lead to sound insights on Gnutella peering and motivate refined simulation parameters. (Section 3.3.3.3)

Related Work. Due to the explosive growth of P2P file sharing applications with respect to total Internet traffic [204], there has been an unprecedented interest in their analysis [10, 15, 200]. Several attempts have been made to investigate the *overlay-underlay* correlations in P2P systems. Using game theoretic models, Liu et al. studied in [156] the interaction between *overlay* routing and traffic engineering within an Autonomous System (AS), which is a network under a single administrative entity, normally corresponding to an Internet Service Provider (ISP). An analysis of routing around link failures [200] finds that tuning *underlay* routing parameters improves *overlay* performance. Most investigations tend to point out that the *overlay* topology does not appear to be correlated with the *underlay* (e.g., [10]), but the routing dynamics of the *underlay* do affect the *overlay* in ways not yet well understood. To address the apparent lack of *overlay-underlay* correlation, some schemes, e.g., [171, 190], have been proposed. More recently, [15] has made a case for collaboration between ISPs and P2P systems as a win-win solution for both. This section follows some of the spirit of that latter work.

[2]For an overview and further pointers, see `http://www.net.t-labs.tu-berlin.de/research/isp-p2p/`
[3]Dynamically Evolving, Large-scale Information Systems
[4]`http://delis.upb.de/specials/story_of_success/delis_ispp2p.html`
[5]Quoting Dr. John A. Zoidberg, Futurama



(a) Both networks G and G' with the mapping ϕ.

(b) Highlighting one edge e in G and the corresponding path $\pi(e)$ in G'.

Figure 3.3.1. Example of an *overlay* $\mathcal{O} := (G, G', \phi, \pi)$. The mapping ϕ is represented by dash lines between nodes in G and G'.

3.3.1 Modeling Underlays and Overlays

In this section, we introduce our model and methodology for analyzing the relation between under- and *overlays* as well as a first discussion about different modeling aspects. *under- and overlays*

Basically, an *overlay* consists of a network structure that is embedded into another one. More precisely, each node of the *overlay* is hosted by a node in the *underlay* and every edge of the *overlay* induces at least one path between the hosting nodes (in the *underlay*) of its endnodes. The formal definition is given in Definition 3.1.

Definition 3.1 *An* overlay *is given by a four-tuple* $\mathcal{O} := (G, G', \phi, \pi)$*, where* *overlay*

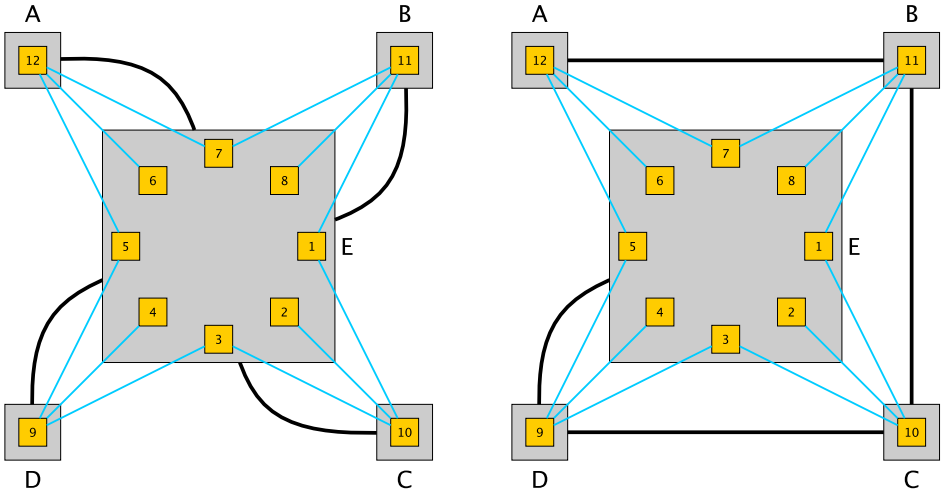
- $G = (V, E, \omega)$ *and* $G' = (V', E', \omega')$ *are two weighted graphs with* $\omega\colon E \to \mathbb{R}$ *and* $\omega'\colon E' \to \mathbb{R}$*,* *$\omega$, ω'*
- $\phi\colon V \to V'$ *is a mapping of the nodes of* G *to the nodeset of* G'*, and* *ϕ*
- $\pi\colon E \to \{p \mid p$ *is a (un-/directed) path in* $G'\}$ *is a mapping of edges in* G *to paths in* G' *such that* $\{\text{source}(\pi(\{u,v\})), \text{target}(\pi(\{u,v\}))\} = \{\phi(u), \phi(v)\}$. *$\pi$*

The interpretation of Definition 3.1 is that G models the *overlay* network itself, the graph G' corresponds to the hosting *underlay*, and the two mappings establish the connection between the two graphs. An example is given in Figure 3.3.1. As direct communications in the *overlay*, which corresponds to the edges of G, is realized by routing information along certain paths in the G', not all parts of the *underlay* graph are equally important. In order to focus on the relevant parts, we associate an *induced underlay* with an *overlay*. The corresponding definition is given in 3.2.

Definition 3.2 *Given an overlay* $\mathcal{O} := (G = (V, E, \omega), G' = (V', E', \omega'), \phi, \pi)$*. The induced underlay* $\widetilde{\mathcal{O}} := H := (V'', E'', \omega'')$ *is a weighted graph, where* *induced underlay*

- $V'' := \{v \in V' \mid \exists\, e \in E\colon \pi(e)$ *contains* $v\}$, *nodes*
- $E'' := \{e' \in E' \mid \exists\, e \in E\colon \pi(e)$ *contains* $e\}$*, and* *edges*
- $\omega''(e') := \sum_{e \in E} \omega(e) \cdot [e'$ *contained in* $\pi(e)]$.[6] *appearance weight ω''*

[6] The definition of ω'' is given in the Iverson Notation [149]. The term inside the squared parentheses is a logical statement and depending on its value, the term evaluate to 1, if its value is true, and to 0 otherwise.



(a) star topology **(b)** path topology

Figure 3.3.2. Examples of two *overlays* where only the topology in the *underlay* network G' changes. Nodes in the *overlay* network are numbered with integers and edges are drawn blue, while nodes in the *underlay* network are labeled with characters and edges are drawn black. In both cases the routing π is done by a shortest-path scheme.

property	A	B	C	D	E
number of hosting nodes	1	1	1	1	8
number of edge in the *overlay* network having an endnode in the node	3	3	3	3	12
UN weighted degree (star top.)	1	1	1	1	4
UN weighted degree (path top.)	1	2	2	2	1
IU weighted degree (star top.)	3	3	3	3	12
IU weighted degree (path top.)	3	9	15	21	12

Table 3.3.1. Table with degree information of the examples given in Figure 3.3.2. The weighted degree corresponds to the weighted degree in the *underlay* network (UN) and the induced *underlay* (IU), respectively.

The weight function ω'' is also called appearance weight.

ω'' = *load caused by overlay*

In other words, the *induced underlay* corresponds to the subgraph of the *underlay* graph that is required to establish the communication in the *overlay* graph. Note that the defined weight can be interpreted as the load caused by the communication and thus is independent of a weighting in the *underlay* network.

3.3.1.1 Analysis

In the analysis of *overlays*, we focus on two important aspects: the identification of key features with respect to the *underlay* and the comparison of different *overlays*.

The first part, the identification of key features, consists of standard tasks of network analysis, e. g., determining important and relevant nodes or edges, clustering nodes with similar patterns, and detecting unusual constellations. As existing techniques can be applied to the *overlay* network and the *induced underlay*, these standard tasks are reasonably well understood in the case of the analysis of a single network. However, these techniques do

not incorporate the relationship between the two networks. An example showing such dependencies is given in Figure 3.3.2 with the corresponding information about the degrees in Table 3.3.1. We use the degree, which is a popular feature, for illustration. However in our studies we noted that these observations carry over to other characteristics. First note, that the number of hosting nodes and the number of communications a node in the *underlay* participates in gives a first impression about its role in the network. Both pieces of informations can be read off the overlying graph G. However, they are completely independent from the routing structure in the *underlay*. As the example illustrates, the degree of a node (in the *induced underlay*) heavily depends on the routing structure. In the case of the star topology, both the weighted degree in the underlying network and in the *induced underlay* are fairly similar, here they are even proportional and clearly identify the center node of the star to be central for the network. The situation drastically changes when using a path topology. Although all communications start/terminate at node E, it is not very central. The nodes C and D take on very active roles, due to the fact that most/all communication has to be routed through them. In many cases, the information provided by the *induced underlay* sufficiently codes the relation between the *overlay* and *underlay* networks, while still enabling us to use standard notation of network analysis. On the other hand, there are some scenarios where the provided view is too coarse. For example, it could make a difference, whether a heavy edge is caused by a single heavy communication or by a multitude of small communications or, conversely, whether all communication of a node in the *induced underlay* have only one target in the *overlay* or are distributed over many targets.

degree and weighted degree

star vs. path topology

some details are masked

One motivation for identifying key features is to build a proper model that can be used for extensive simulations. For example, simulations are used to predict scaling behavior or to experimentally validate heuristics, enhancements, or novel techniques. As such, it is a major issue to structurally compare different *overlays* with each other. On the one hand, our model already reflects all dependencies between the *underlay* and the *overlay* network and, thus, it does not require the *underlay* network, embedding, or routing to be fixed for different instances. On the other hand, due to this elaboration of our model, a simple matching of nodes or edges will not suffice. Our idea is to match key features. For example, one can try to match the *appearance weight* of an edge with structural properties of its endnodes. If both *overlays* have a sufficient number of such matches, it is reasonable to assume that they are created by the same mechanism.

motivation: model for simulation

ω'' vs. endnode

Both parts, the identification of key features and the comparison of *overlays*, benefit from proper analytic visualizations that emphasize relevant aspects of the corresponding networks. Before presenting two visualization techniques (Section 3.3.2), we briefly demonstrate our model and methodology with some experimentally generated examples.

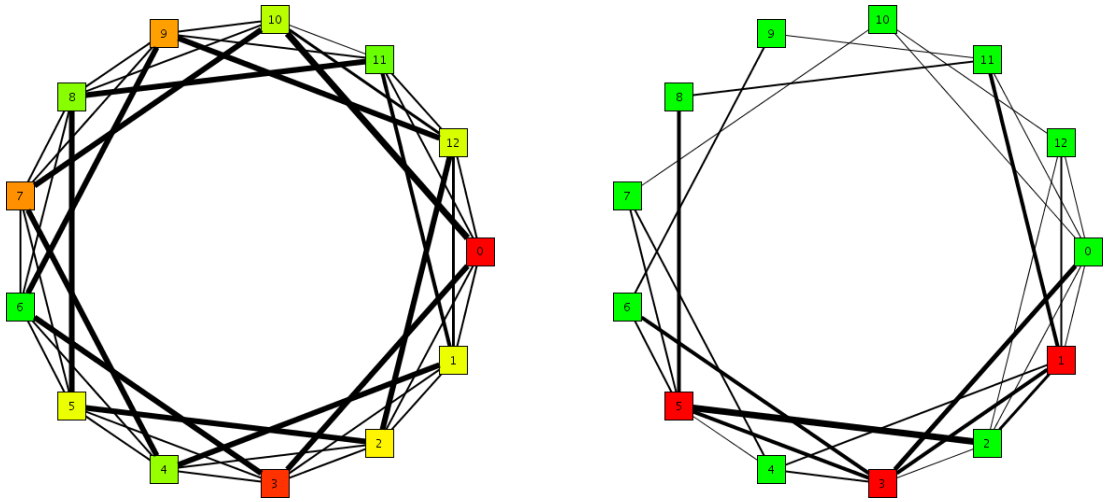
3.3.1.2 Examples

In the following, we demonstrate our model and methodology with simple examples. Before looking at a specific *overlay*, we give two further intuitions.

First, assume a fixed given underlying network. The *overlay* communication can thus be interpreted as a sampling process of pairs in the *underlay*. Depending on the application, different patterns occur. For example, in services such as Internet broadcast, one can expect few highly active nodes, which correspond to the hosts of the service while the majority of nodes participate in only a few communications. Using the *induced underlay*, we can extract such patterns and reconstruct the sampling parameters. Second, assume the underlying network is unknown and acts as a black box, i. e., no information about routing policy and so on is available. By uniformly choosing a sample with sufficiently many communications as the *overlay*, we can not only discover the *underlay*, but also partly reverse engineer the routing mechanism of π. In the special case that the *overlay* network is complete, i. e., every pair of node is connected, the *appearance weight* of the *induced underlay* is highly similar to the (edge-)betweenness of the original underlying network.

underlay as black box

overlay can discover underlay and π



(a) uniform sampling **(b)** star-like sampling

Figure 3.3.3. Example of *induced underlays* for different *overlay* networks in the same underlying network. In the left figure, the communication is uniformly at random distributed over the network and the color codes the (relative) amount of participation. In the right figure, all communications use at least on red node and select the other uniformly at random. In both cases, the thickness of an edge corresponds to the appearance weight.

uniform sampling
star-like sampling

As an example, we consider an underlying network with 13 nodes and a 3-cycle topology, i. e., nodes are cyclic-ordered and each node is connected to 3 of its immediate predecessors and successors. Traffic is routed using shortest path scheme. For simplicity, we set the nodeset of the *overlay* network to the nodeset of the *underlay* and thus ϕ to be the identity function. We define two *overlays*: the first one $\mathcal{O}_1$ (*uniform sampling*) uses uniformly at random selected pairs of nodes for communication, while in the second *overlay* $\mathcal{O}_2$ (*star-like sampling*) the communication takes place between three predefined nodes and all other nodes chosen uniformly at random. The resulting *induced underlays* are displayed in Figure 3.3.3. As can be clearly seen, the short-cuts, i. e., edges that connect two nodes that have a distance of three, have the largest appearance weight and all other edges have relatively small weights for the uniform sampling. This is not surprising as the *appearance weight* resembles the betweenness of edges. The situation drastically changes, when modifying the sampling mechanism. As in the case of the *induced underlay* of $\mathcal{O}_2$, the edges relatively close to the initial set have large weights and edges far away have small weights or do not appear at all. For example, the non-existence of the edges $\{9, 10\}$ is due to the fact that no shortest path between a red node and any other node uses that edge. On the other hand, the edge $\{6, 7\}$ is contained in a shortest path, namely between 3 and 7. However, its absence reveals certain aspects of the *underlay* routing, i. e., the routing between 3 and 7 will either use the path $(3, 4, 7)$ or $(3, 5, 7)$, but never the path $(3, 6, 7)$, which is an arbitrary choice that can be discovered.

3.3.2 Analytic Visualization

Although we have thoroughly discussed *LunarVis* and related visualization techniques—the technique of Baur et. al. in [31] in particular—in Section 3.2, we here briefly recall the points relevant to this case study. Both highlight a given hierarchical decomposition of the network while displaying all nodes and edges. They have already been applied successfully to the network of Autonomous Systems (AS), which is an abstraction of the physical Internet, yet are highly flexible and can be easily adjusted to other networks.

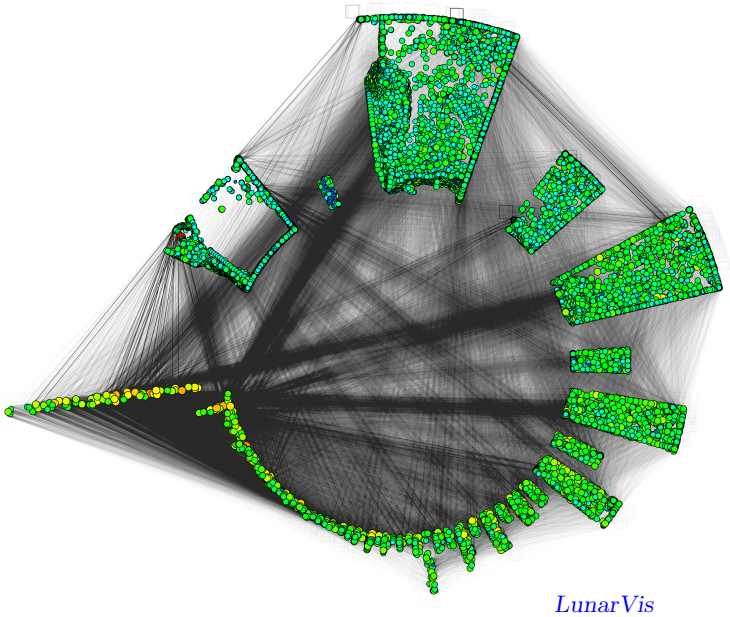
core decomposition

We use the concept of *cores* [29, 201] for the required hierarchical decomposition of the network. Briefly recalling, the k-*core* of an undirected graph is defined as the unique subgraph obtained by recursively removing all nodes of degree less than k. A node has *coreness* ℓ, if it belongs to the ℓ-*core* but not to the $(\ell+1)$-*core*. The ℓ-*shell* is the collection of all nodes having *coreness* ℓ. The *core* of a graph is the non-empty k-*core* such that the $(k+1)$-*core* is empty. Generally the *core decomposition* of a graph results in disconnected sub-graphs, but in the case of the AS network we observe that all k-*cores* stay connected, which is a good feature regarding connectivity. *Core* have been frequently used for network analysis, e. g., [104, 109].

AS: all k-cores connected

landscape metaphor

The first technique employing the concept of *cores* was proposed by Baur et. al. in [31]. More precisely, their algorithm lays out the graph incrementally starting from the innermost *shell*, iteratively adding the lower *shells*. Their implementation uses the *core decomposition* and a combination of spectral and force-directed layout techniques. This layout technique is a *network fingerprint*. Such pseudo-abstract visualizations offer great informative potential by setting analytic characteristics of a network into the context of its structure, revealing numerous traits at a glance. The fingerprint drawing technique *LunarVis* that focuses on the connectivity properties of a network decomposition has been presented in Section 3.2, and shall be used here. Recall that *LunarVis* lays out each set of a decomposition—which are the *core-shells* in our case—individually inside the segments of an annulus. The rough layout of *LunarVis* is defined by analytic properties of the decomposition, allowing the graph structure to determine the details. By virtue of a sophisticated application of force-directed node placement, individual nodes inside annular segments reflect global and local characteristics of adjacency while the inside of the annulus offers space for the exhibition of the edge distribution. Combined with well-perceivable attributes, such as the size and the color of a node, these layouts offer remarkable readability of the decompositional connectivity and are capable of revealing subtle structural characteristics. For more details we refer the reader back to Section 3.2.



LunarVis

Figure 3.3.4. An example visualization of the *core decomposition* (segments) of the AS network using LunarVis. Each nodes represent an AS with size and color reflecting the size of its IP-space. Angular and radial extent of a segment reflect the number of nodes and intra-*shell* edges respectively. Note the extremely large AS (upper left red node) in the minimum *shell*.

3.3.3 Case Study: Overlay Graphs of P2P Systems

peer-to-peer

In this section, we exemplify our analysis technique with a case study of a P2P *overlay*. For our analysis we choose Gnutella [227], an unstructured file-sharing system which relies on flooding connectivity pings and search queries to locate content. Each message carries a TTL (time to live) and message ID tag. To improve scalability, nodes are classified in a two-level hierarchy, with high-performance *ultrapeer* nodes maintaining the *overlay* structure by connecting with each other and forwarding only the relevant messages to a small number of shielded leaf nodes. Responses to pings and queries are cached, and frequent pinging or repeated searching can lead to disconnection from network. More details about Gnutella can be found at [227].

Gnutella

pings and queries

TTL, ID

ultrapeer

leaf nodes

3.3.3.1 Sampling and Modeling the P2P Network

sampling edges

servents caching and churn

crawling insufficient

In order to analyze the *overlay* structure, we first need to identify a representative set of connections, called edges, between nodes in the P2P network. To reduce the bias in our sample, we identify edges where neither of the two endnodes is controlled by us. We refer to such nodes as remote neighbor *servents*. Due to message caching and massive churn in P2P networks (we measured the median incoming/outgoing connection duration to be 0.75/0.98 seconds), a simple crawling approach using pings, e.g., as employed in [194], is not sufficient. However, pings identify nodes that should have been remote neighbor *servents* at some point.

active + passive exploration

passive: ultrapeer

active: crawler

We thus deploy a combination of active and passive techniques to explore the Gnutella network [10]. Our passive approach consists of an *ultrapeer* that participates in the network and is attractive to connect to. It shares 100 randomly generated music files (totaling 300 MB in size) and maintains 60 simultaneous connections to other *servents*. The passive approach gives us a list of active *servents*. The active approach consists of a multiple-client crawler that uses pings with TTL 2 to obtain a list of candidate *servents*. Since queries are difficult to cache, we use queries with TTL 2 to obtain a set of remote neighbor *servents*. These *servents* are then contacted actively to further advance the network exploration. This approach allows us to discover P2P edges that existed at a very recent point of time. When interacting with other *servents*, our crawler pretends to be a long-running *ultrapeer*, answering incoming messages, sharing content, and behaving non-intrusively. This pragmatic behavior avoids bans. The client uses query messages with broad search strings, e.g., mp3, avi, rar to obtain maximum results. We then combine active and passive approaches by integrating the crawler into the passive *ultrapeer*.

discovering Gnutella

IP mapping to underlay

BGP, Routeviews

comparison: random overlay

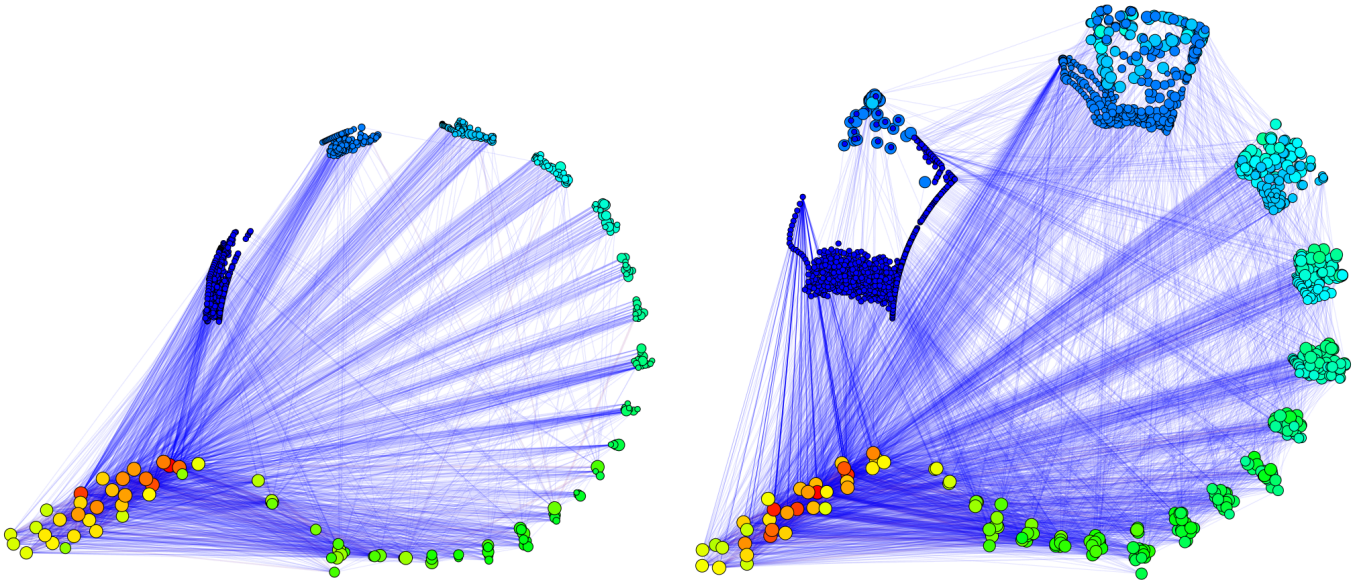
Using this setup, we sample the Gnutella network for one week starting April 14, 2005. The *ultrapeer* logs $352,396$ sessions and the crawler discovers $234,984$ remote neighbor *servents*, a figure significantly higher than most reported results during this period. For each edge of the Gnutella network we map the IP addresses of the Gnutella peers to ASes using the BGP table dumps offered by Routeviews [164] during the week of April 14, 2005. This results in 2964 unique AS edges involving 754 ASes, after duplicate elimination and ignoring P2P edges inside an AS. For the random graph we pick end-points at the IP level by randomly choosing two valid IP addresses from the whole IP space. These edges are then mapped to ASes in the same manner as for the Gnutella edges. This results in 4975 unique edges involving 2095 ASes for the random network at the AS graph level. The different sizes of the graphs are a result of the generation process: we generate the same number of IP pairs for random network as observed in Gnutella, and apply the same mapping technique to both data sets, which abstracts the graph of IPs and direct communication edges to a graph with ASes as nodes and the likely *underlay* communication path as edges. This way, the characteristics of Gnutella are better reflected than by directly generating a random AS network of the same size as Gnutella network.

ϕ: IP $\rightarrow$ AS

π = BGP

ω'' = load caused in underlay

For our analysis, we apply the model and methodology from Section 3.3.1 as follows. The *overlay* $\mathcal{O} = (G, G', \phi, \pi)$ as given in Definition 3.1 uses the direct communication in Gnutella as graph G, the graph G' corresponds to the hosting Internet, in our case the AS level. The mapping ϕ corresponds to the IP to AS mapping, while π is the routing in the AS network. Apart from the already introduced *induced underlay*, we also investigate the network of direct *overlay* communication, yet abstracted to the level of ASes in order to be comparable to the *induced underlay*. Note that in a simplified model, where each communication causes uniform costs, the *appearance weight* in the *induced underlay* (ω'') corresponds to the total load caused by the *overlay* routing in the *underlay* network. As exact traffic measurements on each *underlay* link are non-trivial, this can be interpreted as an estimate of the actual load on *underlay* links due to the *overlay* traffic.



(a) P2P network **(b)** Random network

Figure 3.3.5. Visualization of the *core decomposition* of the *overlay* communication networks. Core-*shells* are drawn into annular segments, with the 1-*shell* at the upper left. Angular and radial extent of a segment reflect the number of nodes and intra-*shell* edges respectively. Inside each *shell* nodes are drawn towards their adjacencies. Colours represent the degree of a node while the size represents their *betweenness centrality*. Edges are drawn with 10% opacity and range from blue (small weight) to red (large weight).

3.3.3.2 Overlay-Underlay Correlation in a P2P System

overlay in LunarVis

Figure 3.3.5 shows visualizations of the direct *overlay* communication of both the network and a random network. Employing *LunarVis* described in Section 3.3.2—and in Section 3.2 in broad detail—these drawings focus on the decompositional properties of the *core* hierarchy. Numerous observations can be made by comparing the two visualizations. Note, first, the striking lack of intra-*shell* edges for all but the maximum *shell* in the Gnutella network (small radial extent). This is also true for edges between *shells*, as almost all edges are incident to the maximum *shell*. This means that almost always at least one communication partner is in the maximum *shell*, a strongly hierarchical pattern that the random network does not exhibit to this degree. Note furthermore that in Gnutella, *betweenness centrality* (size of a node) correlates well with *coreness*, a consequence of the strong and deep *core* hierarchy, whereas in the random network the two- and even the one-*shell* already contain nodes with high centrality, indicating that many peerings heavily rely on low-*shell* ASes. The depth of the Gnutella hierarchy (26 levels) strongly suggests a highly connected network kernel of *ultrapeers*, which are of prime importance to the connectivity of the whole P2P network. However, note that the distribution of degrees (node colors) does not exhibit any unusual traits and that no heavy edges are incident to low-*shell* ASes, in either network.

G.: intra-edges only in max. shell

G.: "all roads lead to max. shell"

R.: coreness ≈ betweenness

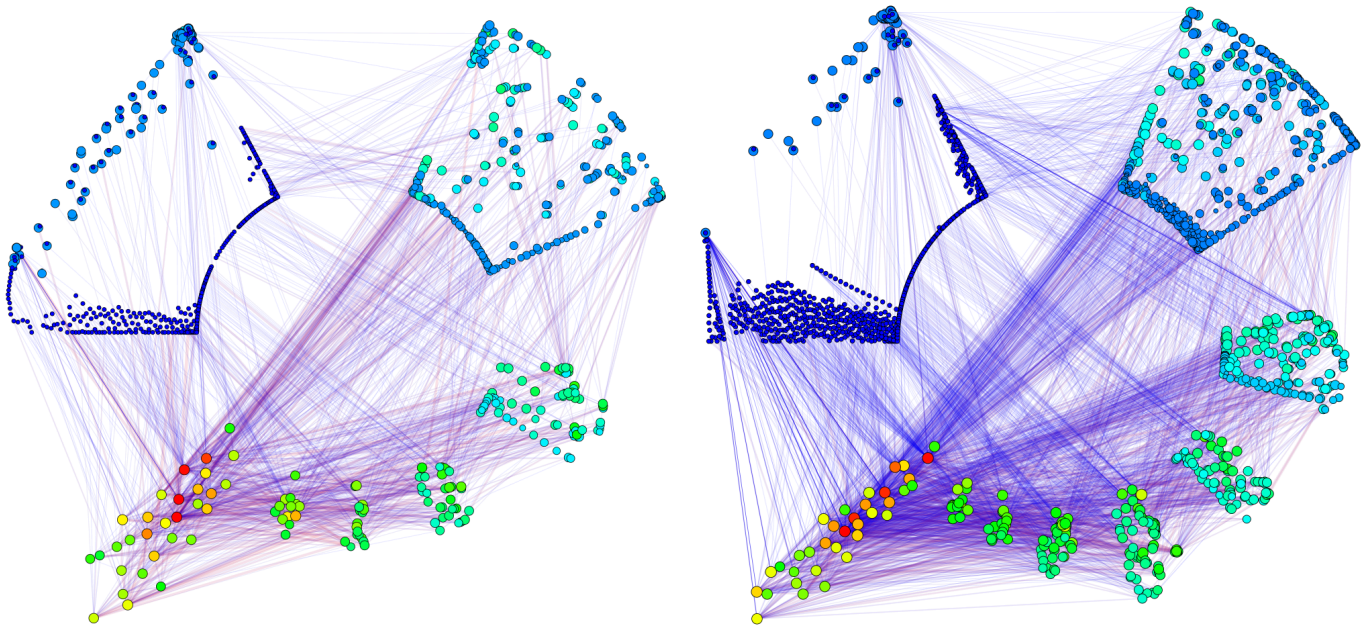
degree dist. similar

underlay in LunarVis

Figure 3.3.6 visualizes the *induced underlay* communication of both the Gnutella network and a random network, employing the same technique and parameters as in Figure 3.3.5. The drawings immediately indicate the much smaller number of ASes and *overlay* nodes in the Gnutella network. As a consequence, more heavy edges (red) exist and the variance in the *appearance weight* (edge color) is more pronounced. This is because of the fact that not all the ASes host P2P users (this is in accordance with our measurements in Section 3.3.3.1), as is the case for the random network. Again, the distributions of degrees do not differ significantly.

fewer ASes ⇒ different load dist.

empty ASes



(a) P2P network **(b)** Random network

Figure 3.3.6. Visualization of the *core decomposition* of the *induced underly* communication network. These drawings use the same parameters as Figure 3.3.5.

R.: high degree in periphery

Gnutella peers inside core of AS

⇝ targeted analysis

edge weight vs. degree or coreness

weight ≁ degree

here: R. ≈ G.

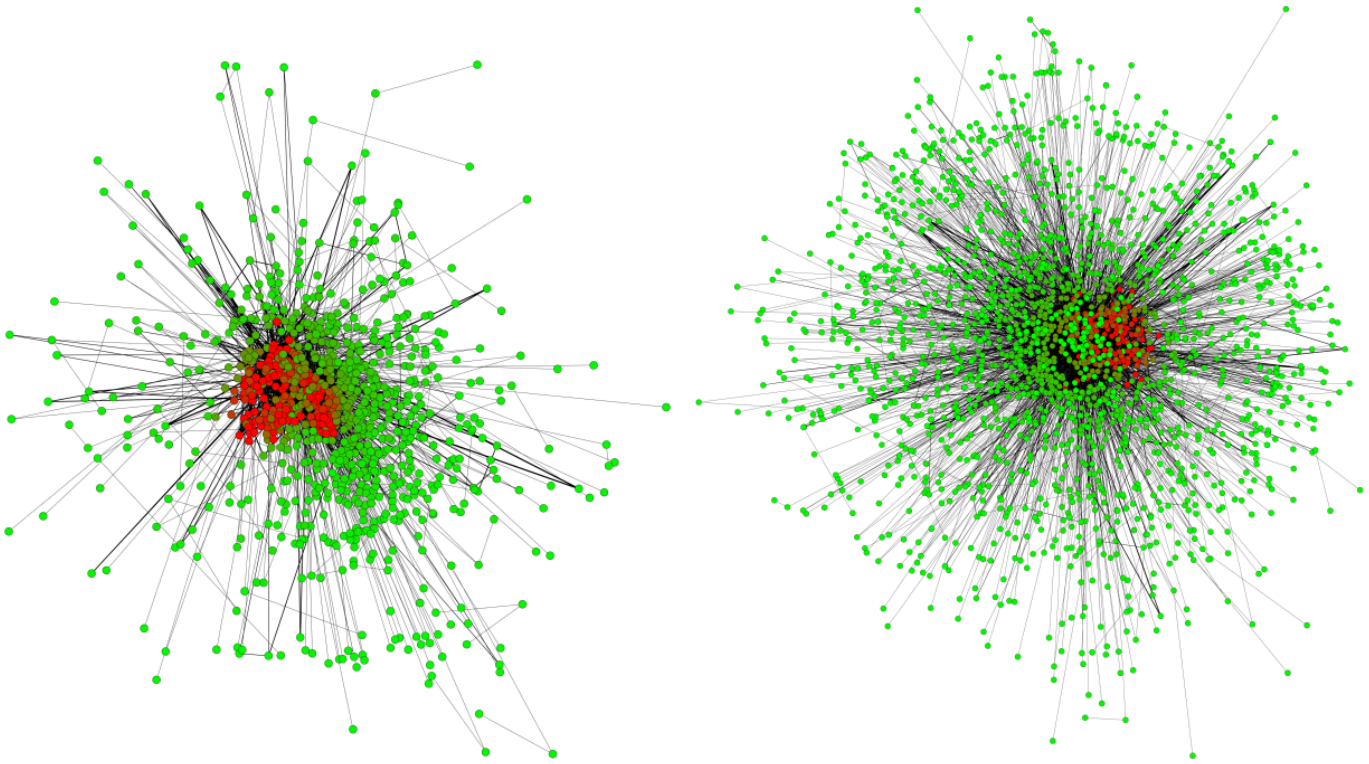
weight vs. coreness

For a closer comparison, Figure 3.3.7 shows a top-down view of the visualizations of communication edges in Gnutella and random network. The visualization technique places nodes with dense neighborhoods (tier-1 and tier-2 ASes) towards the center, and nodes with lesser degrees (tier-3 customer ASes) towards the periphery. We can observe that while both networks have many nodes with large degrees in the center, the random network possesses several nodes with large degree in the periphery. Gnutella, on the other hand, has almost no nodes with large degree in the periphery in either model. Moreover, this pattern is more pronounced for Gnutella in the direct *overlay* communication model, while the random network is largely similar in both models. In other words, it appears that Gnutella peering connections tend to lie in ASes in the *core* of the Internet where there may be high-bandwidth links available.

To further corroborate our observations, we investigate structural dependencies between the *induced underlay* communication model and the actual *underlay* network, by comparing the *appearance weight* with node-structural properties of the corresponding endnodes in the original *underlay*. We focus on the properties degree and *coreness*, as both have been successfully applied for the extraction of customer-provider relationship as well as visualization [207, 104], due to the ability of these properties to reflect the importance of ASes. We systematically compare the weight of an edge with the minimum and maximum degree and *coreness* of its endnodes. Figure 3.3.8 shows the corresponding plots.

From the plots of minimum and maximum degree, it is apparent that the *appearance weight* of an edge and its endnodes' degrees are not correlated in either the Gnutella or the random network, as no pattern is observable. Also, the distributions are similar as the majority of edges are located in the periphery of the network where the maximum degree of the endnodes is small. We thus hypothesize that the relation of load in the P2P network and node degree in the underlying network is the same in both the Gnutella and the random network. In other words, the Gnutella network does not appear to be significantly affected by the node degree of *underlay* nodes.

However, considering the *coreness* reveals interesting observations. From the graphs of minimum and maximum *coreness* in Figure 3.3.8, we can observe that although there is no



(a) Gnutella **(b)** random network

Figure 3.3.7. Comparison of occurring communication in the P2P network and the Random network, using a *landscape metaphor* visualization, see Section 3.3.2.

correlation in either of the two networks, their distributions are different. In the random network the distributions are very uniform, which is a reflection of its random nature. But in the case of Gnutella almost no heavy edge is incident to a node with small *coreness*, as can be seen in the minimum-*coreness* diagram. Positively speaking, most edges with large *appearance weights* are incident to nodes with large minimum *coreness*. Interpreting *coreness* as importance of an AS, these Gnutella edges are located in the backbone of the Internet, an important observation. The same diagram for the random network does not yield a similarly significant distribution, thus denying a comparable interpretation. For instance, in the random network, there exist edges located in the periphery that are heavily loaded. As an aside, backbone edges need not necessarily be heavily loaded in either network.

R.: uniform

G.:heavy edges ≁ low-core nodes

here: R. ≠ G.

All these observations and analysis show that the Gnutella network differs from random networks and there appears to be some correlation of Gnutella topology with the Internet *underlay*.

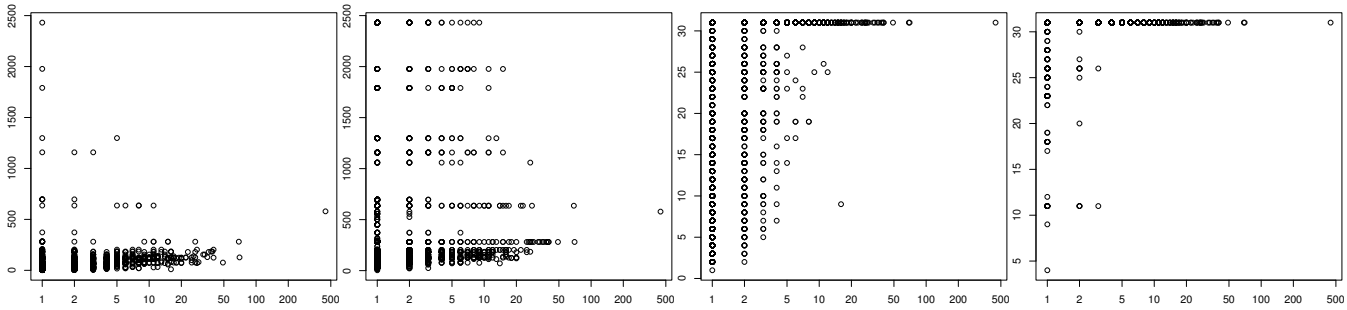
3.3.3.3 Sensitivity Analysis for Refining the Model

The analyses conducted in Section 3.3.3.2 and the conclusions drawn, solely rely on analytic visualizations. Based on these we now aim at a deeper understanding of the properties of the *underlay* communication the P2P network induces. Modifying the generation process for the random networks in ways suggested by our observation, we are now able to conduct a sensitivity analysis, in order to find parameters for the random network that lead to a more aligned edge-*coreness* distribution with the observed P2P network.

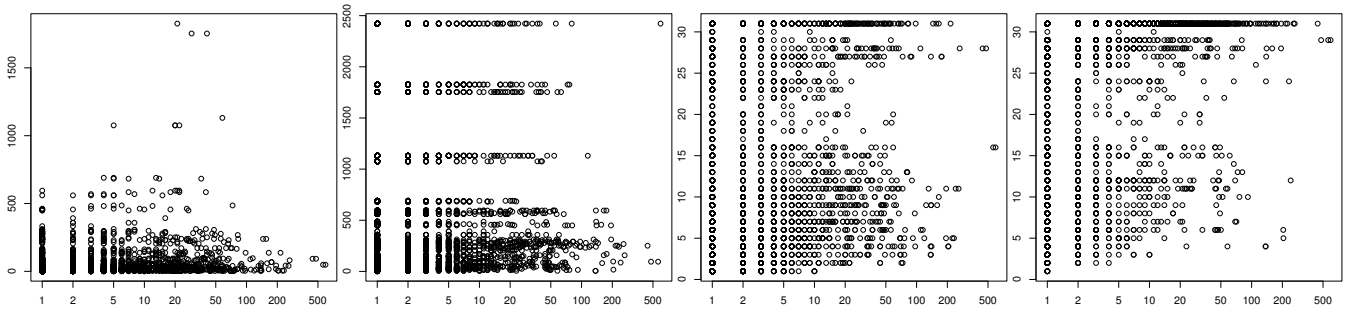
from AV to model refinement

It is both reasonable to assume that many nodes are in lower *shells* (customer ASes) and that heavy nodes (*ultrapeers*) are in higher *shells*. Therefore we consider two modifications:

refined peering



Gnutella measurements, edge *appearance weight* vs. endnode properties



random peers, edge *appearance weight* vs. endnode properties

Figure 3.3.8. Comparing *appearance weight* with minimum and maximum degree and *coreness* of the corresponding endnodes in Gnutella and the random network. Each data point represents an edge, the x-axis denotes the *appearance weight* and the y-axis reflects the degrees (*coreness*) of the endnodes. All axes use a logarithmic scale.

directed routing

The *low coreness communication* restricts the IP-spaces that are available for communications to those hosted by ASes with low *coreness*. Analogously the *high coreness communication* uses only IPs located in ASes with high *coreness*. For reasons of space and simplicity we present only the plots of two of our various experiments. In order to model the routing in the Internet more accurately, we considered the AS network as directed and thus had to adjust the *coreness* calculation properly. As a rule of thumb, the values roughly double compared to the original scenario described in Section 3.3.3.2.[7] Figure 3.3.9 shows the plots that correspond to the right four diagrams in Figure 3.3.8. Again a data point is plotted for each edge in the *induced underlay*, with coordinates that correspond to its *appearance weight* (x-axis) and to its minimum/maximum incident node *coreness* (y-axis). The corresponding plots of the degree distributions are omitted as they did not differ much.

closer to Gnutella

At a first glance we can observe that the restriction to low *coreness* communication does not yield a significant difference to the corresponding plot of our initially unrestricted random network (Figure 3.3.8 lower right). Although the distributions are shaped in a highly similar manner, they differ in the maximum occurring *appearance weight*. On the other hand, the high *coreness* communication exhibits a very different pattern. Its distributions are more similar to those from Gnutella than the random ones. A very interesting observation, is that although communicating IPs are located in ASes with high *coreness*, some routing paths use low-*coreness* ASes.

[7] Undirected, i.e., bidirectional edges are replaced by two unidirectional edges, see [29] for details on *cores* in directed graphs.

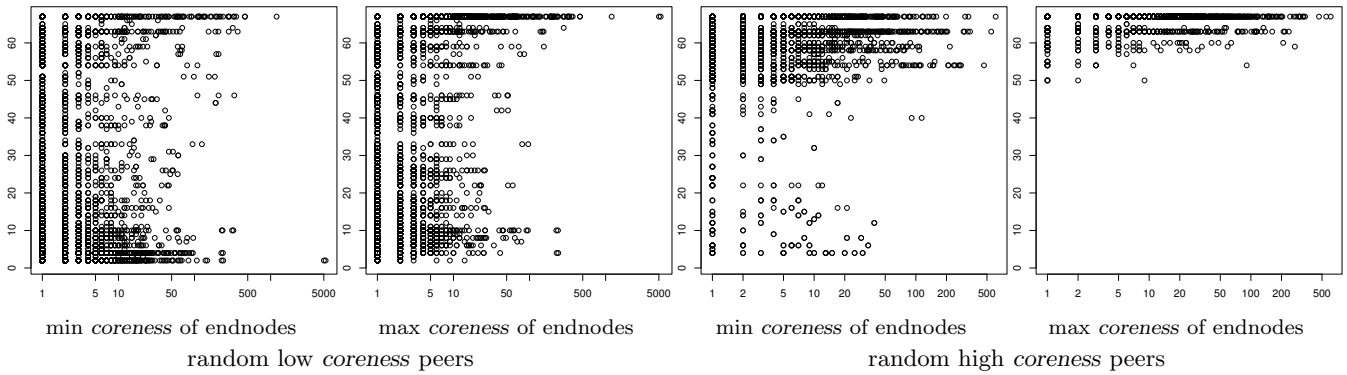


Figure 3.3.9. Comparison of *appearance weight* with minimum and maximum *coreness* of the corresponding endnodes in tuned random networks. In the first and second pairs of figures, the communicating IPs that are hosted by ASes with low *coreness* (≤ 2) and with high *coreness* (≥ 25), respectively. Otherwise the plots use the same settings as in the right hand half of Figure 3.3.8.

Interpreting these findings, we conclude that the observed part of Gnutella mainly corresponds to a large part of the network spanned by the *ultrapeers* and only few leaf nodes are included. Typically *ultrapeer* nodes maintain a connection to a certain (small) number of leaf nodes. On the other hand, the leaf nodes possess only slow Internet connections and connect to the well-performing *ultrapeers*, who in turn shield them from a large amount of P2P traffic, yet enable them to locate and share content. The well-know effect of rampant free-riding corroborates our interpretation. More precisely, the phenomenon refers to the fact that a large number of nodes remain online for very short durations, share no content, and are only interested in finding content, while a small percentage of nodes participate in the network for very long durations, and provide most of the content sought after in the network. Hence, they participate in much more communications as compared to the other P2P nodes.

free-riders hard to detect

Section 3.4

k-Core-Driven Random Graphs using Preferential Attachment

The first all-natural line of fully cooked refrigerated entrees (with no funny-sounding ingredients you can't pronounce).

(Harris Ranch ad, some BART car, San Francisco)

NETWORK ANALYSIS DRIVEN BY VISUAL ANALYTICS appears to work well, if it takes into account the *core decomposition* of a network—especially if a focus on the structure of connectivity in a network is needed. The last two sections showcased these tools and showed us that for some networks the *core* structure can certainly be claimed to be of prime relevance. This, however, immediately raises a family of theoretical questions centered around the following: "What *k-core* hierarchies can exist with a given number of nodes and edges?"

By now, the *k-core* structure is commonly applied in order to identify central parts of networks, as it peels the network layer by layer, filtering out less important parts that are sparsely connected with the remaining graph. Example applications are network *fingerprinting* with LaNet-vi [18] or *LunarVis* (see Section 3.2), protein network analysis [228], or the exploration of modern online social networks [82]. The interest in a special direction of this field, the modeling of classes of graphs, has significantly increased recently, yielding studies of *complex systems* such as the Internet, biological networks, river basins, or social networks. While random graphs have been studied for a long time, the standard models appear to be inappropriate because they do not share certain abstract characteristics observed for those systems (see below). A crucial field of application of graph generators is the simulated evolution of a given network, granting insights in both its past development and its anticipated future behavior. One prominent example is the Internet at the Autonomous System level where various models have emerged over the last few years, including *BRITE* [160], *Inet* [142], *nem* [158], and various models presented by Pastor-Satorras and Vespignani [183]. While this network has been observed to possess a very distinct *k-core* structure, kept track of over a long period of time [104], all generating tools so far ignore this structure, and thus largely fail to do justice to this significant property. Overall, up to our knowledge an approach to create networks with a given *k-core* structure is missing so far.

In this section, we first establish a number of theoretical bounds related to the above question. Building on these, we then develop a random network generator for a predefined *k-core* structure. To address this issue we refine the abstract measurement of *core* sizes to a *core fingerprint* that additionally includes information on the inter-connectivity of each pair of *shells*. This allows us to design a simple and efficient method to incrementally generate randomized networks with a predefined *k-core* structure, starting with the maximum *core*. By utilizing two results on edge rewiring we achieve a structure that precisely matches the

core fingerprint. We shall see that predefining the *core fingerprint* of a network still leaves many degrees of freedom open. Since we focus on the network of Autonomous Systems as a case study, we exploit this fact and optionally bias the randomness in the adjacency of nodes towards *preferential attachment*, as described by Barabási and Albert [16]. This paradigm of setting up links in a network has been proven to introduce a *power-law degree distribution*, which has first been observed by Faloutsos et al. [86] for the Internet. Our approach imposes almost no modifications on a vanilla realization of *preferential attachment*, a fact that is reflected by our experimental results. We thus manage to coalesce two of the most fundamental concepts in the theory of complex networks of the recent past, *k-cores* and *preferential attachment*. To see how our generator performs in practice, we finally perform a comparative evaluation with two well-known AS network[8] generators, *BRITE* and *Inet*, and with reality, based on a number of established criteria from network analysis. Our results yield that our generator is highly suitable for the simulation of AS topologies, confirming the importance of the *core decomposition*. Moreover we show that *BRITE* largely fails to capture significant characteristics of the AS network, including its *core decomposition*, and that *Inet* roughly matches the reference except for its general tendency to be too densely connected. A major drawback of *Inet* is its generation time of several minutes, whereas our *core* generator and *BRITE* create a topology within seconds. Despite the good fitness of our generator it still offers degrees of freedom: The high customizability of our rather generic *core* generator suggests several adaptations that can further increase the fitness to the specific peculiarities of the AS network. Such adaptations to special networks can be realized by employing a number of structural modifications such as swapping and rewiring without interfering with the *core decomposition*.

Without the diligent work on a survey of Internet topology generators conducted by Lin Huang (student's thesis), a resourceful student of mine, this section would quite probably lack its case study on the AS network. In my personal records, pondering about how to wed the *core fingerprint* and *preferential attachment* correctly and efficiently, and brooding over a succinct and conclusive proof together with my former colleague Michael Baur, was among my most interesting pieces of algorithm design. At the same time this section marks a turning point in my course of work, since after finishing it, I decided to finally dare approaching dynamic clustering, something I postponed a bit until then. A preliminary version of this section—in fact, a generator *without* using *preferential attachment*—has been published in [33], based on joint work with Michael Baur, Marco Gaertler, Marcus Krug and Dorothea Wagner. Due to an invitation to a journal, which followed the latter work, we then decided to attempt integrating *preferential attachment* and finished a publication [34] which closely resembles this section. I would like to thank Jorge Busch for pointing out a problem in the first version of Lemma 3.4.1, and for valuable comments and a discussion on its resolution, which we were then able to use in this revised version.

Main Results

- We establish tight bounds on the number of nodes and the number of edges present in the *cores* of a network. (Lemma 3.4.3)
- Two fundamental operations on edges, *rewiring* and *swapping* do not change the *k-core decomposition* of network (but allow to fully explore the theoretical bounds). (Lemmata 3.4.2 and 3.4.1)
- We present a simple and efficient algorithm for generating networks that strictly adhere to a given *k-core* structure, called *core fingerprint*, and prove its correctness. (Section 3.4.2 and Algorithm 12)
- *k-core* generation can be augmented as to use *preferential attachment*, yielding a power-law degree distribution. (Section 3.4.2)

[8] AS stands for Autonomous Systems (in the Internet)

- We exemplify the feasibility of our technique in a case study using the AS network of the Internet, comparing our generator to the established topology generators *BRITE* [160] and *Inet* [142], and to measurements of the real AS network. (Section 3.4.3)

Related Work on Random Models and *Preferential Attachment*. A plethora of models for random graphs have been proposed in the past. The most prominent and fundamental include the *Erdős-Rényi* model [85], also known as $\mathcal{G}(n, m)$, *Gilbert's* model $\mathcal{G}(n, p)$ [107] and the *Watts and Strogatz* model [222], which is also known as the *small-world*-model. However, in a number of real-world graphs some properties have been identified that are unlikely to emerge in these models, most notably a distribution of node degrees that roughly obeys a *power-law*, a fact that has been identified by Faloutsos et al. [86]. More precisely, the number of nodes with degree d is proportional to $d^{-\gamma}$ for some constant γ. Graphs with this property are commonly referred to as *scale-free*. Barabási and Albert describe a growth process coined *preferential attachment* [16] that generates graphs with such a degree distribution. Starting out with an empty graph, this process iteratively adds a new node that is adjacent to a fixed number of already existing nodes. The choice of a specific neighbor is made with a probability proportional to the current degree of the nodes. In the following, we closely adhere to the particularly efficient implementation of *preferential attachment* proposed by Batagelj and Brandes [27].

$\mathcal{G}(n, m)$, $\mathcal{G}(n, p)$

power-law

preferential attachment

Future Work. As mentioned above our generator is not tightly packed with constraints but still offers space for adding in peculiarities of a specific family of networks. Thus, whenever the focus is on a particular application which must be simulated as best as possible in a randomized way, such peculiarities may easily be built into the generator, filling some of its degrees of freedom; we propose a few such approaches in Section 3.4.2.4.

degrees of freedom

3.4.1 Preliminaries

3.4.1.1 *Core* Decomposition

We briefly recall the concept of the *core decomposition* from Section 3.1.3, as it is fundamental to this section, and introduce some refined nomenclature. A nested decomposition of G is a finite sequence $(V_0, \ldots, V_k)$ of subsets of nodes such that $V_0 = V$, $V_{i+1} \subseteq V_i$ for $i < k$, and $V_k \neq \emptyset$. The concept of the *core decomposition* was originally introduced by Seidman [201] and generalized by Batagelj and Zaversnik [29]. Constructively speaking, the *i-core* of an undirected graph is defined as the unique subgraph obtained by iteratively removing all nodes of degree less than i. In the following we often use the existence of a *removal order* σ, which we further specify in Section 3.4.2.2. This is equivalent to the closed definition of the *i-core* as the set of all nodes with at least i adjacencies to other nodes in the *i-core*. The *core number* of a graph is the smallest i such that the $(i+1)$-*core* is empty, and the corresponding *i-core* is called the *core* of a graph. Figure 3.1.1 depicts the *core decomposition* of an example graph with a *core* number of 4. A node has *coreness* i, if it belongs to the *i-core* but not to the $(i+1)$-*core*. Thus, the collection of all nodes having *coreness* i make up the *i-shell*. An edge $\{u, v\}$ is an *intra-shell edge* if both u and v have the same *coreness*, otherwise it is an *inter-shell edge*. Informally speaking, the *coreness* of a node can be viewed as a robust version of the degree, i. e., a node of *coreness* i retains its *coreness* even after the removal of an arbitrary number of nodes of smaller *coreness*. The *core decomposition* can be computed in linear time with respect to the graph size [28]. It has frequently been used for network analysis, see e. g., [104, 109]. In the following section we state some observations on *core* structures, that are crucial to our approach.

nested decomposition

i-core

σ

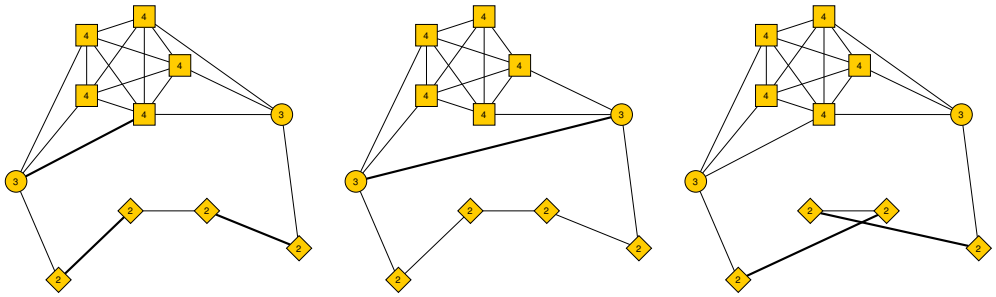
core number

core of a graph

coreness i

i-shell

intra- and inter-shell edges



(a) original graph **(b)** after rewiring **(c)** after swapping

Figure 3.4.1. Rewiring and swapping edges, the labels indicate the *coreness* of the nodes.

3.4.1.2 Edges in a *Core* Hierarchy

The following two lemmata summarize two facts about the relation of intra- and inter-*shell* edges. Note that Lemma 3.4.1 corrects a flaw present in a previously published preliminary version of this section [33]. We later exploit this interaction and interchangeability of edges in our network generation algorithm.

2 inter-shell e. ↔ 1 intra-shell e.

Lemma 3.4.1 (Rewiring) *Let $G = (V, E)$ be a graph. Let $u, v \in V$ be two non-adjacent nodes with the same coreness and $\{u, w\}, \{v, w'\} \in E$ two edges such that* $\text{coreness}(u) < \min\{\text{coreness}(w), \text{coreness}(w')\}$. *Then $G' := (V, E')$ with $E' := E \setminus \{\{u, w\}, \{v, w'\}\} \cup \{u, v\}$ has the same core decomposition as G. Conversely, let $u, v \in V$ be two adjacent nodes with the same coreness k and with at most $k - 1$ neighbors in higher cores, and $w, w' \in V$ such that* $\text{coreness}(u) < \min\{\text{coreness}(w), \text{coreness}(w')\}$ *and $\{u, w\}, \{v, w'\} \notin E$. Then $G'' := (V, E'')$ with $E'' := E \setminus \{u, v\} \cup \{\{u, w\}, \{v, w'\}\}$ has the same core decomposition as G.*

swapping intra-shell edges

Lemma 3.4.2 (Swapping) *Let $G = (V, E)$ be a graph, $u, v, w, w' \in V$ be four nodes all having the same coreness, $\{u, v\}, \{w, w'\} \in E$ be two intra-shell edges, and $\{u, w\}, \{v, w'\} \notin E$. Then the graph $G' := (V, E')$ with $E' := E \setminus \{\{u, v\}, \{w, w'\}\} \cup \{\{u, w\}, \{v, w'\}\}$ has the same core decomposition as G.*

these lemmata in prose

It is not hard to see that the correctness of both lemmata follows from the definition of *cores*. The cumbersome prerequisites can be understood more easily by the concept of a *removal order* that will be introduced later in Section 3.4.2.2. Informally speaking, Lemma 3.4.1 allows for most pairs of disconnected nodes of the same *coreness* to each remove one edge to some nodes of higher *coreness* and instead become connected, and vice versa, without changing the decomposition. Furthermore, according to Lemma 3.4.2 we can swap the endnodes of intra-*shell* edges if this does not interfere with existing connections. Figure 3.4.1 illustrates these two lemmata for an example graph. Using these statements, we can now establish (tight) bounds of the sizes of *cores* and *shells*.

bounds on core- and shell sizes

Lemma 3.4.3 (Size of i-Cores) *Let $G = (V, E)$ be a graph, $(V_0, \ldots, V_k)$ its core decomposition and $G_i := G[V_i]$ the i-core. Then the size of the i-core is bounded as follows:*

$$i + 1 \leq |V_i| \quad \text{and} \quad \frac{(i+1)i}{2} \leq |E_i| \ . \tag{3.4.1}$$

Let $n_i := |V_i \setminus V_{i+1}|$ be the number of nodes with coreness i and $m_i := |E_i \setminus E_{i+1}|$ the number of all edges whose endnodes with minimum coreness has coreness i for $0 \leq i \leq k$ (we define $V_{k+1} := \emptyset$ and $E_{k+1} := \emptyset$). Then the size of the i-shell is bounded as follows:

$$0 \leq \quad n_i \quad \leq |V| \tag{3.4.2}$$

$$\left.\begin{matrix} \left\lceil \frac{i \cdot |n_i|}{2} \right\rceil & , \text{ if } n_i > i \\ \binom{n_i}{2} + n_i \cdot (i - n_i + 1) & , \text{ if } n_i \leq i \end{matrix}\right\} \leq \quad m_i \quad \leq \begin{cases} i \cdot n_i & , \text{ if } i < k \\ i \cdot n_i - \frac{i^2+i}{2} & , \text{ if } i = k \end{cases} \tag{3.4.3}$$

intra-shell edges "contribute twice"

Note that the bounds for the *i-core* (Eq. 3.4.1) are trivially obtained from the definition. The bounds for the *i-shell* (Eq. 3.4.2 and 3.4.3), however, use the above two lemmata, i. e., the *shell* has the minimum number of edges, if it has the maximum possible number of intra-*shell* edges, since each such edge contributes twice, and a minimum number of inter-*shell* edges. An analogous reasoning yields the upper bounds. We omit proofs for the bounds of this lemma except of the following, which is the only one not obvious.

Proof. [Upper Bound in Eq. 3.4.3] By definition, there exists a *removal order* σ that iteratively removes a node v from V_k with $\deg(v) \leq k$, such that eventually all nodes in V_k are removed. We now count the maximum number of edges that still allow such an order of removal $\sigma(v)$, $v \in V_k$ by adding up the number of edges the removed nodes in such a *removal order* can maximally be incident to. For the first $n_k - (k+1)$ nodes (which can be zero), the *removal order* σ implies that the current node v can have a maximum degree of k. For the last $k+1$ nodes (minimum number of nodes for a *k-shell*) however, the number of incident edges during the *removal order* is even less, resulting in a $(k+1)$-clique supported by $(k^2+k)/2$ edges. Thus, we arrive at

 # edges allowed by σ

$$\underbrace{(n_k - (k+1)) \cdot k}_{\text{by nodes beyond } k+1} + \underbrace{\frac{(k+1) \cdot k}{2}}_{\text{by clique of last } k+1 \text{ nodes}} = k \cdot n_k - \frac{k^2+k}{2} \tag{3.4.4}$$

edges in total, which proves the bound. It is easy to see that this bound is tight, since our arguments induce an immediate construction. Note, that this bound also applies to lower *shells* when excluding edges to higher *shells*. □

3.4.2 *Core* Generator

In this section, we first introduce a set of relevant parameters for the construction of *core* structures and discuss which combinations of these lead to feasible instances, i. e., are capable of realizing a graph with a predefined *core* structure. Then we describe our basic algorithm that generates such graphs, and point out several variations. As the 0-*shell* only contains isolated nodes and in order to reduce technical peculiarities, we restrict ourselves to generating graphs with an empty 0-*shell*.

3.4.2.1 Input Parameters

 # nodes per shell
edge-matrix

There are several possibilities to specify *core* structures. Of the quantitative approaches, the most obvious is to give the number of nodes per *shell*, the number of intra-*shell* edges, and the number of inter-*shell* edges (for each pair of *shells*). This can be coded as a vector $N \in \mathbb{N}_0^k$ where n_i is the number of nodes in the *i-shell* and a symmetric matrix $M \in \mathbb{N}_0^{k \times k}$, where $m_{i,j}$ contains the number of edges connecting the *i-shell* with the *j-shell*. We call this the *core fingerprint*. For example, the graph (omitting isolated nodes) given in Figure 3.1.1 has the following *fingerprint*:

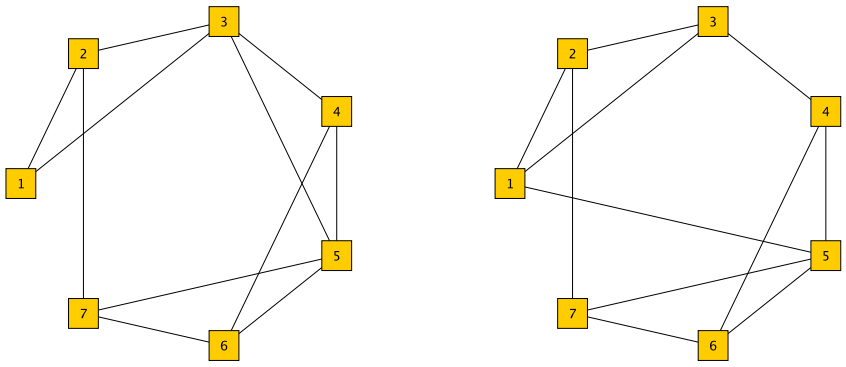
$$N := (4,3,2,5) \quad \text{and} \quad M := \begin{pmatrix} 3 & 1 & 0 & 0 \\ 1 & 2 & 2 & 0 \\ 0 & 2 & 0 & 6 \\ 0 & 0 & 6 & 10 \end{pmatrix}$$

Clearly, the implied sizes of the *shells* have to respect the bounds established in Lemma 3.4.3. This kind of specification of *core* structures provides the maximum degree of freedom, i. e., the user can configure the size distribution of each *shell* and is only limited by constraints ensuring consistency.

Algorithm 12: *Core* Generator

Input: integer k, vector $N \in \mathbb{N}_0^k$, valid symmetric matrix $M \in \mathbb{N}_0^{k \times k}$
Output: graph $G = (V, E)$

1 $V \leftarrow \emptyset$; $E \leftarrow \emptyset$; targetNodes $\leftarrow \emptyset$
2 **for** $i \leftarrow k$ **to** 1 **do** // introduce next *shell*
3 list $V_i \leftarrow \{n_i$ new nodes$\}$
4 $\sigma_i : V_i \to \{1, \ldots, n_i\}$ defined by $\sigma_i^{-1}(\ell) = V_i[\ell]$; // *removal order*
5 $u \leftarrow V_i[n_i]$; // last node in *removal order*
6 list sourceNodes $\leftarrow V_i \setminus \{u\}$; // u cannot source intra-edges
7 list targetNodes$[i] \leftarrow \{u\}$; // u into PA-list
8 list unconnectable $\leftarrow \{u\}$; // see line 22
9 **for** $j \leftarrow i$ **to** k **do** // select target *shell*
10 **for** $m \leftarrow 1$ **to** $m_{i,j}$ **do** // introduce m_{ij} edges
11 $s \leftarrow$ sourceNodes[random] ; // source of new edge
12 $\overline{C} \leftarrow N(s) \cup \{s\}$; // invalid target candidates
13 **if** $i = j$ **then**
14 $\overline{C} \leftarrow \overline{C} \cup \{\ell \in V_i \mid \sigma(\ell) < \sigma(s)\}$; // ℓ violates σ
15 $C \leftarrow$ targetNodes$[j]$; // target candidates list
16 C.removeAllOccurences$(\overline{C})$
17 $t \leftarrow C$[random] ; // target of new edge
18 $E \leftarrow E \cup (s, t)$
19 **if** *outdeg(s)* $= i$ **then** // source saturated
20 sourceNodes.remove(s)
21 **else if** $i = j$ **and** *outdeg(s)* $\geq n_i - \sigma_i(s)$ **then**
22 sourceNodes.remove(s) ; // no more intra-targets
23 unconnectable.append(s) ; // store for inter-targets
24 targetNodes$[i]$.append(s) ; // populate PA-lists
25 targetNodes$[j]$.append(t)
26 **if** $i = j$ **then**
27 sourceNodes.appendAll(unconnectable) ; // restore
28 remove direction of edges
29 list poorNodes $\leftarrow \{v \in V_i \mid \deg(v) < i\}$
30 list richNodes $\leftarrow \{v \in V_i \mid \deg(v) > i\}$
31 **while** poorNodes $\neq \emptyset$ **do** // rewire unsaturated nodes
32 $v \leftarrow$ poorNodes[random]
33 $w \leftarrow$ richNodes[random]
34 $C \leftarrow N(w) \setminus N(v)$; // pivot candidates
35 $c \leftarrow C$[random]
36 $E \leftarrow E \setminus \{\{w, c\}\} \cup \{\{v, c\}\}$
37 **if** $\deg(v) = i$ **then** // v saturated
38 poorNodes.remove(v)
39 **if** $\deg(w) = i$ **then** // w no longer rich
40 richNodes.remove(w)
41 $V \leftarrow V \cup V_i$; // *shell* i completed
42 **return** *graph* $G = (V, E)$



(a) before rewiring **(b)** after rewiring

Figure 3.4.2. Example of rewiring. The *fingerprint* $N = (0, 0, 7)$ and $m_{3,3} = 11$ resulted in the left hand graph. Clearly, node 1 has insufficient degree. In the rewiring phase we can choose either node 3 or 5 as the RICH node. For the right hand graph we selected node 3 and node 5 as the RICH node and the pivot node, respectively. Thus, we arrive at $E = E \setminus \{\{3, 5\}\} \cup \{\{1, 5\}\}$.

3.4.2.2 Algorithmic Approach

start at max. core

Our generator builds a graph by iterativeley adding new *shells* beginning at the maximum *core*. When adding a new *shell* we create nodes and edges according to the given *core fingerprint* and take care to not change the *coreness* of nodes in previously built higher *shells*. The detailed pseudo code is given in Algorithm 12.

σ_i per shell

In order to guarantee that the *coreness* of nodes in the *i-shell* will not exceed i, we define an order σ_i which will be maintained as a valid *removal order* for this *shell* (line 4). It is of vital importance to ensure that for every node in V_i the sum of the number of neighbors in the *shell* i with a higher value of σ_i and the number of neighbors in higher *shells* does not exceed i. To model this, newly created edges are directed such that inter-*shell* edges point from the lower *shell* to the higher *shell* and intra-*shell* edges are directed in accordance to our predefined order σ_i and each node in V_i is restricted to a maximum out-degree of i (line 20). For inserting an edge between nodes in different *shells* it is sufficient to choose any non-adjacent node pair (lines 11-25). We are left to guarantee that the *coreness* is exactly i and not less. An example where this not yet satisfied is given in Figure 3.4.2a.

consider edges directed, by coreness and σ_i

element generation phase

While lines 3 to 27, called the *element generation phase*, avoid erroneously high values of *coreness*, the *rewiring phase* in lines 29 to 40 solves the problem of erroneously low values of *coreness* by a sophisticated movement of edges. We choose a node v with insufficient degree and a node w with degree greater than i. Then we select a neighbor $c \in N(w)$ and replace this adjacency by a new edge $\{v, c\}$.

rewiring phase

list SOURCENODES

Revisiting element generation, observe that some subtlety has been put into the choice of incident nodes of new edges. We maintain a list of SOURCENODES which contains all nodes of the current *shell* i that can serve as the source of an edge. SOURCENODES contains all nodes of a *shell* at most once, excluding those with a saturated out-degree (see line 20). However, in the special case of $i = j$, nodes that are unconnectable because they are already connected to all nodes with a higher value of σ, must also be excluded (see line 22). These nodes are not yet saturated and must thus be re-inserted into SOURCENODES. Since edges can be directed towards any higher *shell*, we maintain a list of TARGETNODES$[i]$ for each *shell* i throughout the algorithm. These lists are the key for realizing *preferential attachment*. At any time TARGETNODES$[i]$ contains each node of *shell* i with multiplicity equal to its degree (by lines 24-25). We initialize TARGETNODES$[i]$ with $u = \operatorname{argmax}_{v \in V_i} \sigma(v)$ since u is a feasible target for all $v \in V_i$. For each choice of s in line 11, s and its neighborhood $N(s)$ have to be

key for preferential attachment

removed from the list of target nodes. In the special case of $i = j$, s must not connect to nodes with a lower value of σ. This pruning is done in lines 12-16. Based on the observations in this section, we analyze Algorithm 12 in terms of correctness and running time in the following section.

3.4.2.3 Analysis of the Algorithm

correct for $\max m_{ii}$

Observation 3.4.1 *Algorithm 12 generates valid core structures for the maximum number of intra-shell edges, i.e.* $m_{ii} = i \cdot n_i - (i^2 + i)/2$.

Proof. Let $m = i \cdot n_i - (i^2 + i)/2$. A node is removed from sourceNodes if either its out-degree is equal to i or it is connected to all nodes with a higher value of σ, i.e. for every $s \in$ sourceNodes there is at least one valid target node $t \in$ targetNodes. If sourceNodes is empty we have inserted $(n_i - (i+1)) \cdot i + (i+1) \cdot i/2 = m$ edges (see Equation 3.4.4). □

inductive proof

Based on this observation Lemmas 3.4.4 and 3.4.5 prove the correctness of Algorithm 12 inductively.

element generation phase correct

Lemma 3.4.4 *Given a valid matrix M, and a valid subgraph $G(V_k, \ldots, V_{i+1})$ the generation phase constructs the subgraph $G(V_k, \ldots, V_i)$ such that M is obeyed and all nodes in V_ℓ have coreness $\leq \ell$, $i \leq \ell \leq k$.*

Proof. Let $i = j$. Lines 20 and 14 guarantee that σ is a valid *removal order*. Thus, all nodes in V_i have *coreness* $\leq i$ and the *coreness* of all other nodes remains unchanged. Due to Observation 3.4.1 the upper bounds in Lemma 3.4.3 can be attained, thus any valid m_{ii} can be realized.

Let now $i < j$. Analogously, requiring $\text{outdeg}(v) < k$ preserves the *removal order* and thus a *coreness* of i or less for nodes in V_i. Again, the *coreness* of all other nodes remains unchanged, and the upper bound in Lemma 3.4.3 can trivially be attained. □

coreness not exceeded

Since the above lemma shows that the element generation phase fits in all nodes and all edges required by matrix M and grants to each node a *coreness* equal to or less than the required value, we are left to prove that the rewiring phase refines the edge set such that equality holds.

rewiring phase correct

Lemma 3.4.5 *Given a valid matrix M, and a valid subgraph $G(V_k, \ldots, V_{i+1})$. If $\text{coreness}(v) \leq i$ holds for all $v \in V_i$, then the rewiring phase moves edges such that the subgraph $G(V_k, \ldots, V_i)$ is valid, i.e. M is obeyed and all nodes in V_ℓ have coreness ℓ, $i \leq \ell \leq k$.*

coreness exactly met

Proof. The lemma is proven if equality holds in the invariant, i.e., the list poorNodes defined in line 29 is empty. Suppose there exists at least one node $v \in$ poorNodes, then, clearly $coreness(v) < i$. Assume now for contradiction all nodes in V_i have degree $\leq i$, then, Lemma 3.4.3 is violated. Thus, there exists $w \in$ richNodes $\in V_i$ with $\deg(w) > i$. Since $\deg(w) > \deg(v)$, $C = N(w) \setminus N(v) \neq \emptyset$. Let $c \in C$, the new set of edges $E' = E \setminus \{\{w, c\}\} \cup \{\{v, c\}\}$ still obeys M, decrements $\deg(w)$ and increments $\deg(v)$, increasing $coreness(v)$ by at most one. Thus, the rewiring phase maintains the stated invariant. Furthermore, due to the strict increase and decrease of $\deg(v)$ and $\deg(w)$, respectively, |poorNodes| strictly decreases to 0, which terminates loop 31. □

By induction, Lemmas 3.4.4 and 3.4.5 yield that Algorithm 12 constructs a graph in accordance with M and V_i, $0 \leq i \leq k$, since the base case, i.e. the empty graph, is trivial.

In terms of running time the crucial parts of the algorithm are the updates and random accesses of the lists sourceNodes, targetNodes, poorNodes, and richNodes and creation of the target candidate and pivot candidate lists (lines 16 and 34). We use array-backed

array-backed lists

lists to guarantee constant-time access to random elements. When we remove an element e, we fill its position with the last element of the list, avoiding moving all successive elements of e. Since we only have random access to the lists, preserving their orders is not required.

running time

Lemma 3.4.6 *The asymptotic running time of the generator in Algorithm 12 is bounded by* $O((m^2 + n^2k)\log(n))$.

Proof. The runtime of the element generation phase is dominated by the assembling of target node candidates in line 16. Building a decision tree from $\overline{C}$ in $O(n \log n)$ time, based on the ordering σ, we can prune list C in $O(m \log n + n \log n)$ time per edge, which dominates lines 3 to 27.

The running time of the rewiring phase is dominated by determining the list of pivot candidates in line 34 using $O(n \log n)$ time per rewiring. The total number of rewirings is bounded by $n \cdot k$. This dominates lines 29 to 40 as well as the element generation phase and all peripheral steps. Assuming the graph is connected, in total, both phases sum up to a running time of $O((m^2 + n^2k)\log(n))$. □

in practice
eager → lazy

Since real-world networks seldom exhibit pathologic characteristics, we replaced the eager computation of the candidate list in lines 12 to 17 by a *lazy* selection from TARGETNODES[i] that is repeated until a valid t has been drawn. Clearly this does not improve worst-case running time but works faster for virtually all applications.

We performed our experiments on a recent standard PC, running SUSE Linux 10.2 with an implementation in Java. Absolute running times ranged between 100 and 500 milliseconds for the AS network which is comparable to *BRITE*. The running time of *Inet* is in the order of minutes. See Section 3.4.3.1 for the description of these generators.

3.4.2.4 Refinements

future work: possible refinements

Although the *core fingerprint* is the prime characteristic we focus on in this work, together with the inclusion of a *preferential attachment* mechanism, a number of potentially describing features of a network exist. In this section, we briefly discuss other relevant features, that can easily be integrated in our generator.

connectivity within shells

Connectivity is a very basic characteristic of a network, boiling down to the number of connected components. Building upon the *core decomposition*, this can be refined to the number of connected components *per shell*. While the whole graph or even the *i-core* can be connected, the *i-shell* can still have several disconnected components. If this is not desired, the user can specify the number and the sizes of connected components. The generator will then first create a spanning forest, where each tree is the seed of a component, and mark these edges as not rewirable. Note that requiring a specific set of connected components restricts the set of valid *shell*-connectivity matrices. However, this can be resolved by allowing the number of edges or the number and sizes of connected components to slightly deviate from the predefined values, depending on the user's interests.

predefined degree distribution

targeted rewiring

Returning our focus to the degree distribution, the approach described in Section 3.4.2.2, depending on not a single parameter, can clearly be further elaborated. We tested two variants of our implementation of *preferential attachment*. In the first variant, we require the degree distributions of each *shell* as an input. Based on these we then pre-fill the array TARGETNODES[i] in line 7 with the nodes in V_i, using the exact multiplicities as given by the degree distribution and an ordering analogous to σ. This approach clearly biases the *preferential attachment* process towards the desired degree distribution (see Figure 3.4.3). Alternatively, we can solely rely on a post-processing step. In this case we can completely abandon *preferential attachment* and simply apply a sequence of rewirings (Lemma 3.4.1) and swappings (Lemma 3.4.2) in order to approach a given degree distribution. Although both these techniques yielded very good results, we exclude them from further evaluation, due to their requiring rather specific parameters in addition to the *core fingerprint*.

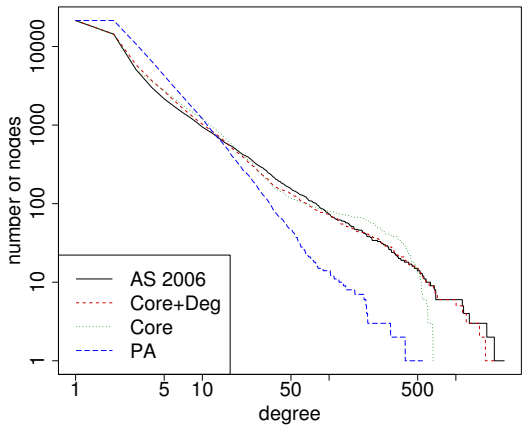


Figure 3.4.3. The number of nodes with degree at least d for the AS network, the original, and the refined *Core* generator for January 2006. A graph generated by *preferential attachment* of approximately the same size is shown for comparison.

3.4.3 Modeling the AS Network

AS-simulation

An important application of a *core*-aware network generator is the simulation of the Internet at the AS level. In this section we compare networks generated by our method and established topology generators with three exemplary snapshot of the real AS network at the router level taken by the Oregon Routeviews project [212] at midnight on January 1, 2002 (oix-full-snapshot-2002-01-01-0000), on January 1, 2006 (oix-full-snapshot-2006-01-01-0000), and on July 1, 2007 (oix-full-snapshot-2007-07-01-0000). Table 3.4.1 shows the sizes of these graphs.

Routeviews

3.4.3.1 Topology Generators

The first methods to generate networks with Internet-like structure date back to the 1990s and a multitude of techniques has been proposed since then. Among the most popular and widely used tools we have chosen *Inet*-3.0 [142] and *BRITE* [160] for our comparison since these are commonly included in other studies which cover a broader range of existing models [159, 142]. Although nem [158] also seems promising we do not take it into account because of its limitation to networks not greater than 4000 nodes. We decided to *not* use the specific degree distribution of the reference graph for our mechanism of *preferential attachment*. The reason for this is, that a generation purely based on *k-cores* and raw *preferential attachment* is much more instructive. The effect of using the reference degree distribution as a blueprint are showcased in Figure 3.4.3.

Inet, BRITE

The *Internet topology generator Inet* [142] generates an AS-level representation of the Internet. Its developers claim that *"it generates random networks with characteristics similar to those of the Internet from November 1997 to February 2002, and beyond"*. Basically, *Inet* generates networks with a degree distribution which fits to one of the power laws originally found by Faloutsos et al. [86], namely that the frequency of nodes with degree d is proportional to d raised to a power of a constant α: $f(d) \propto d^{\alpha}$. Since this law does not cover all nodes

power law degree distribution

	AS 2002-01	AS 2006-01	AS 2007-07
Number of Nodes	12,485	21,419	25,787
Number of Edges	25,980	45,638	53,014

Table 3.4.1. Sizes of the AS network snapshots.

and in order to match other relevant properties as well, optimizations for various specific conditions were added to the original procedure over time. The complete generation method is explained in [142]. Since the procedures of *Inet* are already customized to AS networks, only a small number of input parameters can be specified: the total number of nodes, the fraction of degree-one nodes, and the size of the square used for node placement in drawings.

The *Boston university Representative Internet Topology gEnerator BRITE* [160] can generate networks for different levels of the Internet topology. Beside this, it offers various other options to customize the generation procedure.

BRITE's parameters

Drawing area. The nodes of the generated topology are distributed in a square of a certain size.

Node distribution. In the drawing area, nodes are either distributed uniformly at random or Pareto.

Outgoing links. New nodes are connected with a specific number of outgoing links to other, already existing nodes.

Connectivity. The neighborhood of a node is selected based on certain guidelines such as geometric locality, *preferential attachment*, or a combination of both.

Procedure. Nodes can either be placed before the addition of edges or in an incremental fashion. In the latter case each new node introduces a number of new edges that can only connect to already existing nodes.

3.4.3.2 Characteristics

compared characteristics

In [142], an extensive collection of characteristics is evaluated that judge the fitness of a generated graph with respect to its real world counterpart. We repeated this evaluation for a representative selection of these properties with a focus on the assessment of the *core* generator. In the following, we summarize the properties we employed in our analysis.

min-, max-, avg.-degree

General statistics. To see how well the generated networks fit to the most obvious characteristics we computed some basic properties: the number of edges, the minimum and the maximum degree. Note that all models strictly meet the given number of nodes, so the number of edges corresponds to density and average degree.

core decomposition

Cores. The *core decomposition* is a significant structural property of an AS network. We compare not only the *core* number but the extensive *core fingerprint*.

clustering coeff., triangles, triples, transitivity

Clustering coefficient. The clustering coefficient is a measure for the local density around a node. It counts how many of a node's pairs of neighbors are themselves adjacent. These values are averaged to get a single measure for the network. Closely related characteristics are the numbers of triangles and triples and the transitivity [197].

avg. path length, eccentricity

Path length. We compare two properties based on path length: *characteristic path length*, which is the average of the distances of all node pairs and average *eccentricity*. The eccentricity of a node is its maximum distance to all other nodes. Average eccentricity then is the average of all nodes' eccentricities.

degree distribution

Frequency versus degree. One of the classic power laws found by Faloutsos et al. [86] is $f(d) \propto d^{\alpha}$, that is, the frequency of nodes with degree d, is proportional to d raised to a power of a constant α. Since this power law does not hold for nearly 2% of the highest degree nodes, we use a modified version [50, 55]:

$$F(d) = \sum_{i>d} f(i) \propto d^{\alpha} \ .$$

neighborhood size

Size of k-neighborhood. Another power law identified in [86] is $\mathcal{N}(k) \propto k^{\beta}$, where $\mathcal{N}(k)$ is the sum over all nodes of their neighborhood sizes within distance k, i. e., $\mathcal{N}(k) = \sum_{u \in V} \sum_{v \in V} \mathrm{dist}_k(u, v)$, where

$$\mathrm{dist}_k(u, v) = \begin{cases} 1 & \text{, if } \mathrm{dist}(u, v) \leq k \\ 0 & \text{, otherwise.} \end{cases}$$

	AS 2002-01	*Core*	*BRITE*	*Inet*
Number of Nodes	12,485	12,485	12,485	12,485
Number of Edges	25,980	25,980	24,967	27,494
Minimum Degree	1	1	2	1
Maximum Degree	2,538	644	302	2,154
Core Number	20	20	2	9
Number of Triples	7,258,817	3,140,777	347,443	6,821,628
Number of Triangles	22,832	17,272	157	11,144
Transitivity	0.009	0.016	0.001	0.005
Clustering Coeff.	0.45	0.24	0.00	0.29
Avg. Path Length	3.63	3.69	5.09	3.29
Avg. Eccentricity	8.74	9.71	8.35	6.85

Table 3.4.2. Characteristics of the AS network of January 2002 and the three generators.

	AS 2006-01	*Core*	*BRITE*	*Inet*
Number of Nodes	21,419	21,419	21,419	21,419
Number of Edges	45,638	45,638	42,835	58,069
Minimum Degree	1	1	2	1
Maximum Degree	2,408	662	411	3,572
Core Number	26	26	2	19
Number of Triples	12,161,105	5,631,122	637,716	30,643,658
Number of Triangles	46,256	36,052	177	75,770
Transitivity	0.011	0.019	0.001	0.007
Clustering Coeff.	0.38	0.17	0.00	0.53
Avg. Path Length	3.81	3.84	5.31	3.07
Avg. Eccentricity	8.52	10.36	8.63	6.45

Table 3.4.3. Characteristics of the AS network of January 2006 and the three generators.

Note that this characteristic can also be measured as an average over all nodes, and it is also known as the *number of pairs within k hops*.

3.4.3.3 Evaluation

In the following, we detail the findings of our systematic evaluation. We gathered results on the three generators as described in Sections 3.4.2 and 3.4.3.1 and on the real AS network for all the properties listed in Section 3.4.3.2. The exact results for these properties can be read off Tables 3.4.2, 3.4.3 and 3.4.4 for the years 2002, 2006 and 2007 respectively, and in Figures 3.4.4-3.4.5.

Based on the previous studies we set appropriate parameters for the generators *Inet* and *BRITE*. For *Inet* we have chosen the default input parameters except for the number of nodes and the random seed. As the results in [161] suggest, we have used *preferential attachment* and incremental growth for *BRITE*. Furthermore, we add two edges for each new node to fit the average degree of AS networks.

Inet and BRITE settings

By construction, the numbers of nodes match the reference AS network, however, the numbers of edges already differ heavily. While the number of edges is only slightly lower for graphs generated by *BRITE*, and exactly fits the reference for the *core* generator (called *Core* in the following), the edge set created by *Inet* is larger by one third.

Inet denser

The well-known phenomenon of highly connected hubs in the AS network accompanied by the power-law degree distribution is regarded as one of the most significant properties

	AS 2007-07	*Core*	*BRITE*	*Inet*
Number of Nodes	25,787	25,787	25,787	25,787
Number of Edges	53,014	53,014	51,571	76,467
Minimum Degree	1	1	2	1
Maximum Degree	2,391	838	393	5,168
Core Number	22	22	2	26
Number of Triples	13,889,150	6,759,443	757,653	56,514,215
Number of Triangles	39,646	29,612	174	162,889
Transitivity	0.009	0.013	0.001	0.009
Clustering Coeff.	0.33	0.15	0.00	0.65
Avg. Path Length	3.89	3.92	5.39	2.99
Avg. Eccentricity	10.24	10.64	8.72	6.52

Table 3.4.4. Characteristics of the AS network of July 2007 and the three generators.

degree distribution

of the Internet. *Inet* reproduces these quite well, but overstates the maximum degree. In contrast, the degree distribution of *Core* oscillates around the reference but fails to produce nodes with a very high degree since *preferential attachment* is not extreme enough; the degree distribution of *BRITE* suggests that the preference of new nodes to connect to existing hubs is not strong enough either. These facts can be observed in Figure 3.4.4.

BRITE: only 2-core

At a first glance, *BRITE* clearly fails to build up any kind of deep *core* structure (the *core* number is 2). The reason for this becomes evident from the incremental generation process of *BRITE*: the iterative addition of nodes incident to two new edges can simply be reversed, resulting in a valid removal sequence for the 2-*core* that ultimately yields an empty 3-*core*. Figure 3.4.6 plots both the number of nodes and the number of edges per k-*core* exemplary for January 2006. *Inet* builds up a decent *core* hierarchy but fails to attain a sufficient depth, obviously resulting in larger mid-level *shells*, in terms of both nodes and edges. By construction, *Core* perfectly matches the reference. The plots in Figure 3.4.5 show the numbers of nodes and edges per k-*shell*, again exemplary for January 2006. They confirm the above observations and additionally grant an insight into the absolute numbers of elements per *shell*.

Inet too shallow

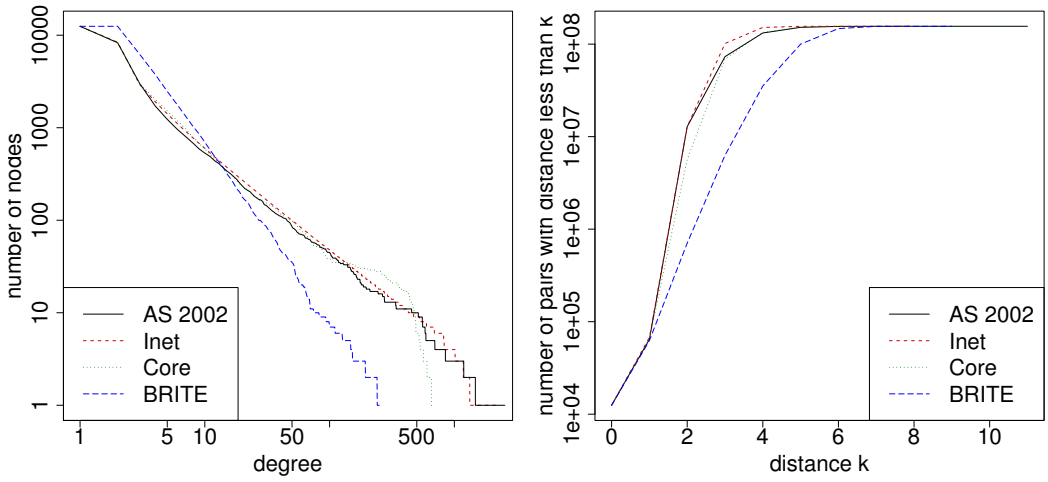
BRITE connectivity

The shallow *core* structure created by *BRITE* is accompanied by a very low transitivity alongside a negligible number of triangles and a tiny clustering coefficient, suggesting that the *BRITE* graph is primarily composed of a set of paths of length two. The high average path length further corroborates this conjecture, since by virtue of *preferential attachment* hubs of high degree evolve, which, however, are interconnected via paths of length two by construction.

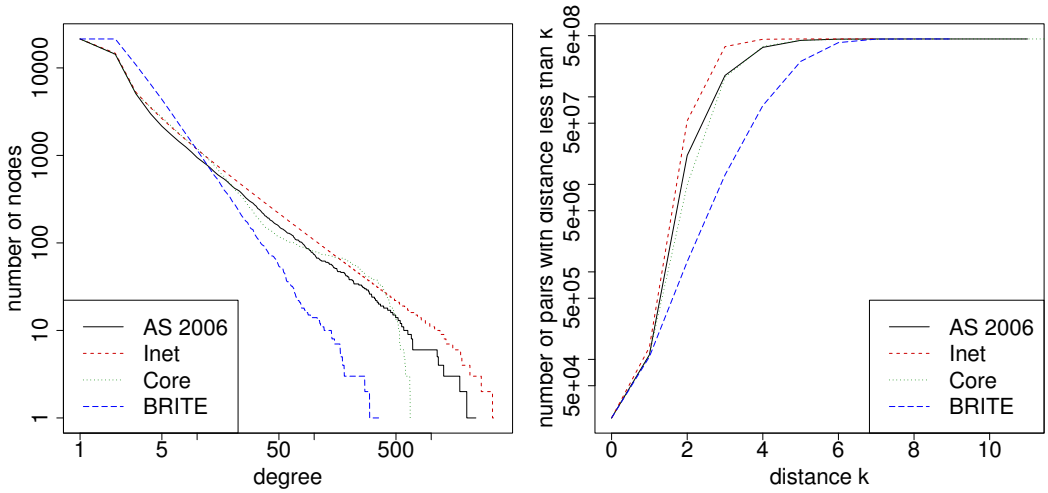
Inet and Core connectivity

The absolute numbers of triples and triangles as well as the transitivity and the clustering coefficient are acceptable for both *Core* and *Inet*. The discrepancy of the latter generator from the reference can quite generally be explained by the increased number of edges. The behavior of *Core* with respect to these values is largely due to the absence of high-degree nodes, since, intuitively speaking, star-shaped structures yield a high number of triples. The relatively high number of triangles thus yields an increased transitivity. The low clustering coefficient, however, suggests, that there is large number of nodes with a sparse direct neighborhood. Since, at the same time, *Core* exhibits a high number of triangles, the majority of these triangles is incident to nodes with higher degree.

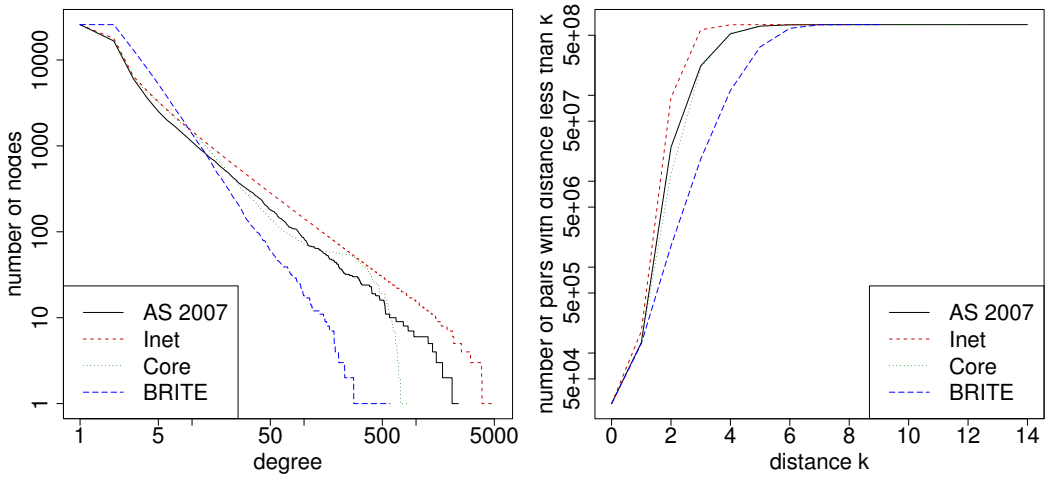
Figure 3.4.4 depicts the size of the neighborhood within k hops (sum over all nodes). Note that the high average path length of *BRITE* mentioned earlier comes along with the slow growth of the neighborhood size. The low average path length and the low average eccentricity exhibited by *Inet* are, again, due to the large edge set. With respect to these values, *Core* excels. Both the average path length and the k-neighborhood practically match the reference.



(a) January 1st, 2002



(b) January 1st, 2006



(c) July 1st, 2007

Figure 3.4.4. The number of nodes with a degree at least d (left) and the k-neighborhood for distances $k \in [0, 10]$ (right) for the AS network and the generated graphs for January 2002, January 2006, and July 2007.

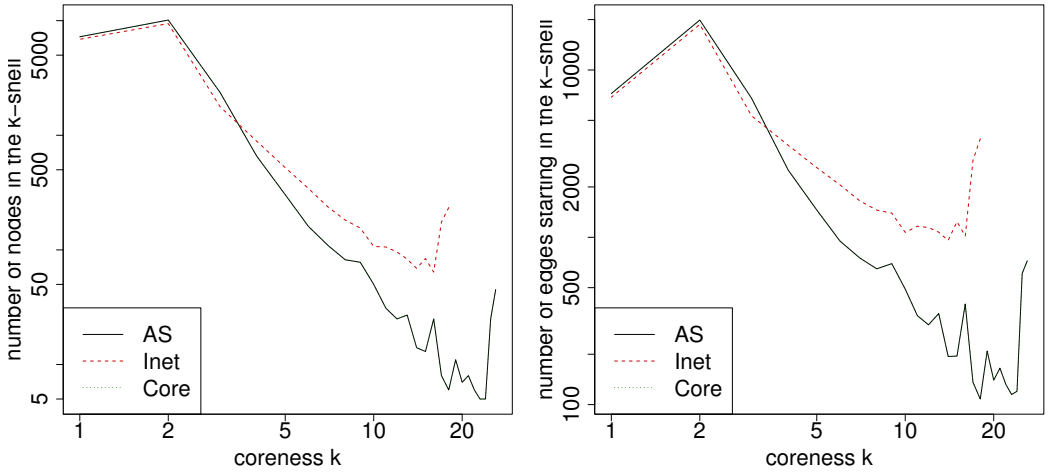


Figure 3.4.5. The numbers of nodes (left figure) and of edges (right figure) per *k-shell* (*BRITE* omitted). An edge is considered to belong to the *k-shell* if its endnode with smallest *coreness* has *coreness k*. Note that the lines of the AS 2006 and of *Core* perfectly match by construction.

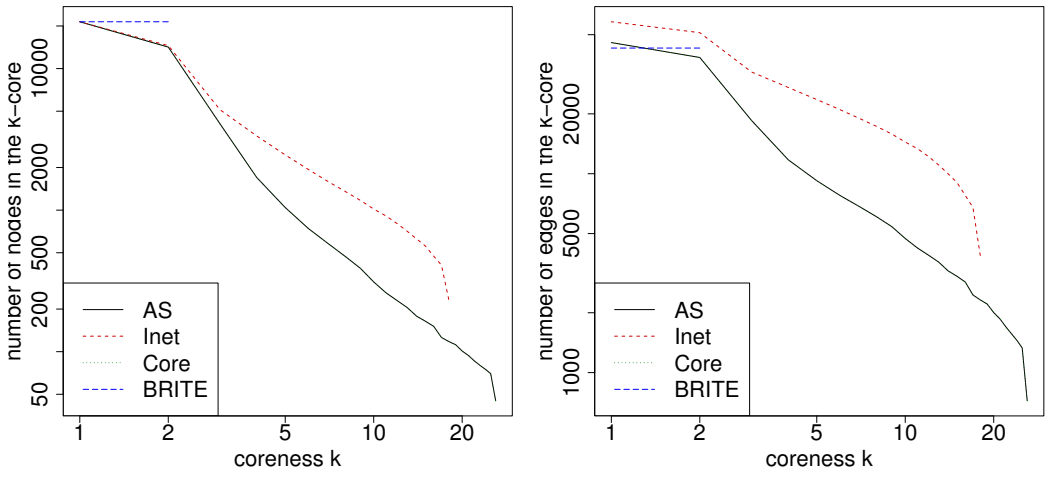
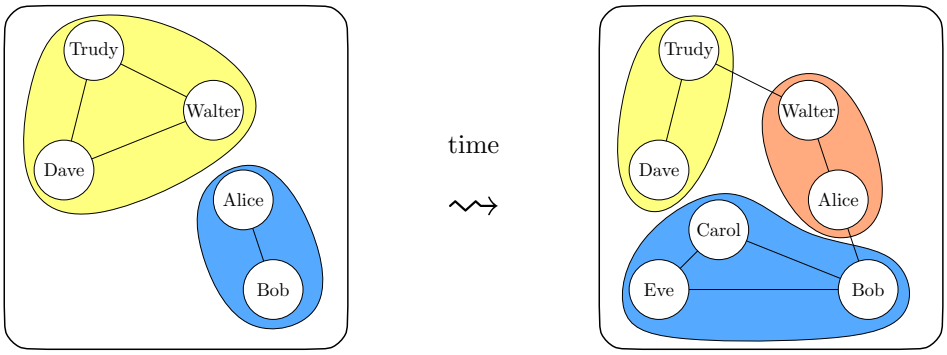


Figure 3.4.6. The numbers of nodes (left figure) and of edges (right figure) per *k-core*. Note that *BRITE* generates only nodes in the 2-*core* and that the lines of the AS 2006 and of *Core* perfectly match by construction.

Chapter 4

Clustering a Dynamic Graph



"The world is changing, I feel it in the earth, I feel it in the water, I smell it in the air." Even without Galadriel's eldritch wisdom [210] we can see that most networks we take a static view of, in fact change over time. Can we transfer the questions we ask about static networks? What new challenges await us? Should we even dare approaching a dynamic view if we know that many questions about static networks are left unanswered?

Contents

Section 4.1

Preface to Dynamic Graph Clustering

Yesterday we thought the world might end.
Today we'd be happy about that!

(Bastian Katz, paraphrasing spiegel.de's increasingly apocalyptic headlines concerning the great financial crash late in 2008)

EXACTLY WHAT IS THE ESSENTIAL QUESTION in the field of dynamic graph clustering? The obvious foothold of dynamic clustering is that most networks in practice are not static. On the contrary, networks evolve and so do their group structures. Along the lines of classic dynamic problem statements, a canonic question can certainly be phrased as depicted by Figure 4.1.1: A graph G is updated by some *change* Δ, yielding G'. Can we find pro-

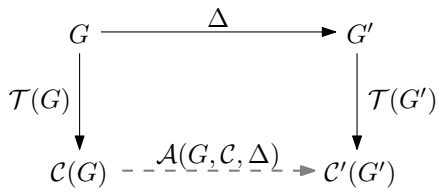


Figure 4.1.1. Problem setting of an update to a clustering $\mathcal{C}$ after a graph G changes (Δ). Can we find an algorithm $\mathcal{A}$ that renders this diagram close to being commutative?

cedures $\mathcal{A}$ that update the clustering $\mathcal{C}(G)$ to $\mathcal{C}'(G')$ without re-clustering from scratch, but work towards the same aim as a static technique $\mathcal{T}$ does? We shall call the static snapshots of a changing graph *time steps*. Figure 4.1.1 depicts an *online setting* of clustering dynamic graphs, i.e., without further knowledge about future *time steps*, a solution to the current single *time step* is to be found. To this end the clustering of the preceeding *time step* can and should be built upon. Clusterings of further past *time steps* might also be consulted. Without such a procedure $\mathcal{A}$, we are left with re-clustering the updated graph instances G' from scratch, which neither requires any new assumptions nor much further implementational work. However, iteratively clustering *time steps* of a dynamic graph from scratch with a static method has several disadvantages:

time step
online setting

Running Time. Even though very fast clustering methods have been proposed recently (e.g. [38] and Section 2.5), running times cannot be neglected for large instances or environments where computing power is limited, as for example in sensor networks [196].

Local Optima. Whole families of clustering methods—*modularity*-based techniques in particular—suffer from local optima, when maximized with common approaches, due to their resilience to optimization. Dynamically maintaining a good solution has the potential to overcome and avoid this pitfall, whereas re-clustering promises becoming trapped in local optima over and over again.

Smoothness.[1] Most static clustering methods do not react in a continuous way to small changes in a graph, they might even cause an oscillation of "orthogonal" clusterings. This effect is highly undesirable in terms of both readability and practicality and can easily be avoided in a dynamic setting.

offline setting

Opposed to this setup, the *offline setting* of dynamic clustering also follows the traditional notion of *offline* problems: The full timespan is known and a solution to a single *time step* can be based on past and future *time steps*, possibly taking into account some global optimization goal, which cannot be hoped for in an *online setting*, e.g., *smoothness*.[1] The distinction between these two setups, however, is slightly blurry concerning the recent literature, as *offline* methods do not really take future information into account, but mostly deserve this predicate for their emphasis on matching and comparing the clusterings of consecutive *time steps*. We shall deal with the *online setting* in Sections 4.2-4.4 and touch the *offline setting* in Section 4.5.

4.1.1 Related Work on Dynamic Clustering

Dynamic graph clustering has so far been a scarcely trodden field. Not a single method can claim to be anything close to established. However, both on a theoretical side and in the shape of case studies, some previous work exists on the topic; we will briefly review those results related to the work in this thesis.

dynamic min-cut tree clustering

Recalling from Section 2.1.3 the concept of *min-cut tree clustering*, recent efforts by Saha and Mitra [193, 192] suggested a method that was supposed to provably dynamically maintain a clustering based on *min-cut trees*. Unfortunately we found a grave error in the methods of the former work, and can give simple counter-examples. We will scrutinize these issues and the corresponding static clustering technique [87] in Section 4.4. Other approaches can roughly be divided up into those with an emphasis on the evolution of the graph and its sequence of static clusterings, and those with an emphasis on quickly finding a new clustering after a change in the graph.

Matching *Offline* Snapshots. Apart from the above, there have been attempts to track communities over time and interpret their evolution, using a sequence of static *time steps* of the network [132, 182], we will come back to this point of view in Section 4.5.2. The former work [132] identifies communities of scientific works in a citation network, i.e., nodes represent publications and directed edges represent citations. Their static clustering algorithm is based on the *cosine-similarity* of the adjacency vectors of nodes. In an agglomerative approach, roughly speaking, the most similar nodes are merged, and only those identified clusters—or parts of them—are actually used which are found in several modified runs of the algorithm and thus considered stable. Then for the two *time steps* the authors use, an overlap criterion is used to track clusters.

set-overlaps of stable clusters

In the latter work [182] the *clique percolation method* (see Section 2.1.2) is again used. The authors exploit the following neat (but arguably strongly counterintuitive) property of this method: Given a graph and a clustering found by this method, then creating additional edges in the graph leads to the algorithm finding either the same clustering as before or a coarsening of it. To actually track clusters the authors thus also cluster the "union graph" of two consecutive *time steps*, and match those clusters of different *time steps* that are contained in the same "union cluster".

evolving CPM

Application-specific case studies using conceptually related matching techniques have been performed for phenomena like web communities or blogspaces in the Internet in [211, 151]. The latter work proposes as a model the so-called *time-graph*, something remotely related to the *time-expanded graph*, we shall propose later. In [9] a parameter-based *dynamic graph clustering* method is proposed which allows *offline* user exploration and *online* clustering.

[1] We clarify this notion in Section 4.3.1.1, roughly speaking it refers to the degree of change between two consecutive clusterings.

Frames (*time steps*) of the graph are stored in hierarchical *tiers* which help to find a relevant *frame* for a specific user query. The evolution of the clustering is approached via *differential graphs* between *frames*, in which subgraphs are identified that are subject to heavy change, by means of change in their edge set.

minimum description length

***Online* Approaches.** An interesting approach is presented in [208] where the information-theoretic *minimum description length* of the *time steps* of a graph sequence and their respective clusterings is used to identify clusterings and points of change in both the graph and the clusterings. Here the authors do not enforce a *smoothness* between the clusterings of *time steps*, but do try to exploit consecutive similarities for speed. Again, the matching of clusters between *time steps* is done with a separate technique, but once more using the *minimum description length*.

data mining

Beyond graph theory, in data mining the issue of clustering an evolving data set has been addressed in, e.g., [53], where the authors pursue the goal of finding a smooth dynamic clustering. The authors define the objective function of a clustering for a given *time step* of a dynamic instance to consist of two parts. The *snapshot quality* sq is the quality of a static data clustering, i.e., the quality regarding an $n \times n$ matrix M_t which describes the relations between n data points, and the *temporal costs* ct, which are high if a clustering strongly differs from its predecessor. A clustering is sought which optimizes the function

sq − ct

$$\overbrace{\mathrm{sq}(C_t, M_t)}^{\textit{snapshot quality}} - \overbrace{\mathrm{hc}(C_{t-1}, C_t)}^{\textit{temporal costs}}, \tag{4.1.1}$$

which quite obviously can be generalized to taking into account further past or future *time steps*, and can thus even be cast into an offline problem. The authors use this concept on two levels, first for defining a *similarity* relation between data points, and then for specifying an objective function for a greedy agglomerative clustering approach. On a point basis, the *snapshost quality* for two points' similarity is the *cosine similarity* of the two points in terms of the number of features shared with other points. The *temporal cost* is the correlation of the time series which count the occurrences of the points in past *time steps*. Roughly speaking, the objective function for clustering then consists of the *snapshost quality* of a clustering being the sum of point similarities and a *temporal cost* which is defined by the similarity between the current and the preceeding *dendrograms* which represent the agglomerative clustering. In general terms, this is close to what we discuss in Sections 4.3 and 4.5, however, as our focus is on *graph* clusterings, we clearly require functions different from those proposed in [53]. We recommend the latter work for further references on dynamic clustering approaches in the context of data mining. In [136] an explicitly bicriterial approach for low-difference updates and a *partial* ILP[2] are proposed, the latter of which we also discuss in Section 4.5.2.

4.1.2 Summarizing Remarks

new field

Hitherto techniques for clustering dynamic graphs are newcomers. Very roughly speaking they can be divided up into two conceptional groups: The first group relies on purely static clustering methods, be that established or homespun, and then, in a second step, apply an additional technique for matching the clusters of consecutive *time steps*. The second group consists of highly innovative pieces of work that propose novel techniques, either separately for both of the above steps, or as an overall concept. The rather confusing collection of possible optimization goals for *dynamic graph clustering* explains this tendency: While the problem for static graph clustering already allows for much interpretation and many formalizations, things seriously get worse when we add in dynamics. For this reason most techniques *for dynamic graph clustering* are indeed able to offer arguments and case studies that support their feasibility, but so far no conclusive arguments about their appropriateness have been given and nothing close to a comparative or systematic evaluation.

no established results

[2] Integer Linear Programming

Undeniably, techniques that solve a given clustering task for a changing network are necessary and cannot wait until the field agrees on all underlying formalizations. In fact, we agree with [53] in terms of their paradigmatic postulations for dynamic clustering and we will come back to this in Section 4.5.2. However, in most of the following we shall address more clearly defined problems and try to avoid either of the two above extremes, as we deem it inevitable to commence this field by building upon known results from static graph clustering.

build upon static knowhow

Motivating Questions. Two general directions, for static graph clustering, spectral methods, e.g., [226], and techniques based on random walks [187, 213], do not lend themselves well to dynamization due to their non-continuous nature, in mathematical terms. Variants of index-based greedy agglomeration [57, 38], however, are well suited. Recalling from Section 2.1, the literature on static graph clustering based on *modularity* maximization is quite broad. To the best of our knowledge, however, there has been no attempt to dynamize any approach for *modularity* maximization. Is an *online* dynamic approach for this algorithm feasible and what can we hope for? Is there a way to dynamize a clustering algorithm in the *online* setting for which actual rigorous properties are to be complied with? How can we actually answer questions about the usefulness of a dynamic approach without also relying solely on handpicked real-world data? Is there a way to transfer knowhow from static graph clustering to the *offline* setup? Can we avoid the introduction of many new degrees of freedom—and thus uncertainty—which is inevitable when using some arbitrary matching procedure for clusters between *time steps*?

questions

Answers in this Thesis. We approach the above questions as follows. First we propose and describe a ready-to-use generator for dynamic random clustered graphs, that then serves as a tool for systematic evaluations and measurements. Our generator is based on the *Erdős-Rényi* model [85], but adds (i) clustering structure, (ii) dynamics with respect to all properties and (iii) a sound probabilistic setup for the evolution of both the graph and its *ground-truth* clustering. We tackle the problem of updating a *modularity*-driven clustering by dynamizing the currently fastest and the most widespread heuristics for *modularity*-maximization. With respect to the three central criteria we postulate, *speed*, *quality* and *smoothness*, our algorithms exhibit a clear superiority to their static counterparts, and we found strong evidence for a *locality assumption*: local changes in a graph are to be reacted to in a largely local way. Exhibiting many new insights into the structure of minimum s-t-cuts subjected to graph dynamics, we show how the *min-cut tree clustering* algorithm can be dynamized—again with positive results in a rigorous assessment with respect to the above three criteria. We briefly sketch out how approximation factors for *min-cut trees* carry over to the guarantees of this algorithm. Finally, we propose a novel approach towards true *offline* clustering which explicitly avoids the introduction of multiple new procedures but solely relies on a reasonable *time-expanded* graph model of a dynamic graph and on established techniques for static graph clustering. This technique implicitly handles the task of matching the clusters of consecutive *time steps*, and thus allows the exploration of cluster evolution "for free".

answers

locality assumption

Parts of this chapter have previously been published in [103, 120, 117, 118]. (We will point out the respective publications in the corresponding sections.)

4.1.3 Preliminaries and Notation

In the following we coin the basic terms in the context of dynamic graphs and their clustering. A *dynamic graph* $\mathcal{G} = (G_0, \ldots, G_{t_{\max}})$ is a sequence of graphs, with $G_t = (V_t, E_t, \omega_t)$ being the state of the dynamic graph at *time step* t. Informally speaking, the *change* $\Delta(G_t, G_{t+1})$ between *time steps* comprises a sequence (with pot. only 1 or 0 entries) of b *atomic* events on G_t, which we detail below (see Tab. 4.1.1). In our *online* setting the sequence of changes arrives as a stream, in the *offline* setting it is completely known in advance.

dynamic graph

time step

online/offline

Table 4.1.1. *Atomic events* in graphs; optional edge weights are in square brackets; the superscripts of δ are often omitted, if irrelevant in the context.

description	*pseudocode*	*symb.*	*formal*	*formal, abbrev.*	*prerequisite*
node insertion	insert(u)	δ^{i}	$V \leftarrow V \cup \{u\}$	$V + u$	$u \notin V$
node removal	remove(u)	δ^{r}	$V \leftarrow V \setminus \{u\}$	$V - u$	$u \in V$
edge creation	connect$(u, v\,[, \omega])$	δ^{c}	$E \leftarrow E \cup \{u, v\}$	$E + \{u, v\,[, \omega]\}$	$\{u, v\} \notin E$
edge removal	discon(u, v)	δ^{d}	$E \leftarrow E \setminus \{u, v\}$	$E - \{u, v\}$	$\{u, v\} \in E$
weight increase	incW(u, v, x)	δ^{+}	$\omega(u, v) \leftarrow \omega(u, v) + x$	$\omega(u, v) + x$	$\{u, v\} \in E$
weight decrease	decW(u, v, x)	δ^{-}	$\omega(u, v) \leftarrow \omega(u, v) - x$	$\omega(u, v) - x$	$\{u, v\} \in E$

4.1.3.1 Formalization of Graph Changes

graph change

A dynamic graph is composed of its static *time steps* and the *changes* in between them. We now formalize *changes* to graphs, which can, together with a starting state, fully define a dynamic graph. In particular we discuss the *changes* we distinguish.

atomic events

edge creation/removal

weight change

node creation/removal

δ

Atomic Events. We call the most elementary changes to a graph *atomic events* or *atomic changes*. These cannot be broken up into smaller parts. Table 4.1.1 lists all *atomic events* and their nomenclature. Most commonly, *edge creations* and *removals* take place, and they require the incident nodes to be present before and after the event. Given edge *weights*, changes to these require the edge's presence. *Node creations* and *removals* in turn only handle isolated (degree zero) nodes, i.e., for an intuitive node deletion we first have to remove all incident edges. If graph G' results from applying an *atomic event* δ to graph G, we write $\delta(G) = G'$. In fact, we can take on the view of δ being a (bijective) function in $\mathbb{G}$:

$$\delta : \mathbb{G} \to \mathbb{G} \tag{4.1.2}$$

$$G \mapsto \delta(G) \tag{4.1.3}$$

non-atomic change

Δ

Non-Atomic Changes. Since *atomic events* have a rather small impact, and quite a few are necessary to, e.g., remove a non-isolated node from a graph, we generalize our view to "larger" *graph changes*. Let $\Delta = (\delta_1, \ldots, \delta_b)$ be a sequence of *atomic events*, then if $\delta_b \circ \delta_{b-1} \circ {}^{3} \ldots \circ \delta_1(G) = G'$, we write $\Delta(G) = G'$. We call Δ a *(non-atomic) change*; this is also a (bijective) function in $\mathbb{G}$:

$$\Delta : \mathbb{G} \to \mathbb{G} \tag{4.1.4}$$

$$G \mapsto \Delta(G) \tag{4.1.5}$$

Obviously, for any two weighted, simple, undirected graphs $G, G' \in \mathbb{G}$ there exists a sequence $\Delta_{G,G'} = (\delta_1, \ldots, \delta_b)$ of atomic events δ_i, such that the subsequent application of the *atomic events* δ_i $(i = 1, \ldots, b)$ yields G'. To refer to a *graph change*, we sometimes also write $\Delta(G, G')$, especially if the particular sequence is not important—note that infinitely many changes lead from G to G'.

batch update

Batch Updates. For the purpose of viewing a continuous stream of *atomic events* in a discretized, manageable way, we coin the term *batch update*. For a clean definition of these *batch updates* we need one more special *atomic event* (Table 4.1.2):

Table 4.1.2. One more special *atomic event*

description	*pseudocode*	*math*	*formal*	*formal, abbrev.*	*prerequisite*
time step event	tStep	δ^{t}	$t \leftarrow t + 1$	$t + 1$	(special)

[3] The term $p \circ q$ denotes the concatenation $p(q())$ of the functions p and q, i.e., q happens first.

We now define a *batch update* as a *graph change* Δ consisting of $b+1 \geq 1$ *atomic events* $(\delta_1, \ldots, \delta_{b+1})$ such that δ_{b+1} is a *time step event*, and no other $\delta_i \neq \delta_{b+1}$ is a *time step event*. Taking the view of a dynamic clustering algorithm, informally speaking, we use *batch updates* to summarize compound graph changes into scalable collections of b *proper atomic events*, i.e., not *time-steps*, such that the trailing *time step event* indicates (to an algorithm) that a readily updated clustering must now be supplied for output. Between *time steps* it is up to the algorithm how it maintains its intermediate clustering. Note that b can be 0, yielding an empty change, or 1 yielding an atomic change. In the latter case, if the context is clear, we often simply omit the trailing *time step event*. An *atomic batch update* is a single *proper atomic event* followed by a *time step event*.

b

time step events delimit time steps

proper event

atomic batch update

4.1.3.2 Dynamic Clusterings and Preclusterings

Static clusterings have been discussed in depth in earlier parts of this work, and with these at hand, the concept of a *dynamic clustering* bears no real surprises: A dynamic clustering ζ of a dynamic graph $\mathcal{G} = (G_0, \ldots, G_{t_{\max}})$ is a sequence $(\mathcal{C}_0, \ldots \mathcal{C}_{t_{\max}})$ of clusterings, such that $\mathcal{C}_i$ is a clustering of G_i. In our view, a (online) *dynamic clustering algorithm* $\mathcal{A}$ is a procedure which, given the state G_t of a dynamic graph $\mathcal{G}$, a sequence of graph events Δ with $\Delta(G_t) = G_{t+1}$ and a clustering $\mathcal{C}(G_t)$ of the current state, returns a clustering $\mathcal{C}'(G_{t+1})$ of the current state. While the algorithm may discard $\mathcal{C}(G_t)$ and simply start from scratch, a good dynamic algorithm will harness the results of its previous work. If the context rules out ambiguities, we often omit a sub- or superscript of $\mathcal{C}$ indicating a new clustering or different *time step*. Furthermore, without listing all necessary arguments, we assume that a dynamic clustering algorithm $\mathcal{A}$ has access to G_t, $\mathcal{C}(G_t)$ and Δ. Note that we assume an *online* setting unless otherwise noted, in fact, until we reach Section 4.5.

dynamic clustering

ζ

dynamic clustering algorithm

A few generalizations of this definition are immediately imaginable. For example, a dynamic clustering algorithm might not merely rely on the current clustering and the *graph change* in a *Markov*-like manner, but instead take into account a longer history. On the other hand, if history is taken into account, then, certainly, knowing about future graph states would help as well. This, however, would be an *offline* setting of dynamic graph clustering. In this section we focus on the *online* situation (i.e. the future is unknown) and leave the offline problem to Section 4.5.

Markov vs. history

online vs. offline

Preclusterings. Suppose a *change* Δ to G yields G'. If we take on the view of a dynamic clustering that "listens" to a dynamic graph and tries to at least be well defined at all times, we are pretty close to thinking in terms of the *observer design pattern* in programming. This view is particularly helpful when pondering issues close to an implementation. In fact it is pretty straightforward to define *canonic updates* to a clustering for each *atomic event* on the graph, as shown in Table 4.1.3. The only notable actions are the creation of a *singleton* cluster upon node insertion and the removal of a node from its cluster—and the whole cluster, in

preclustering for well-definition

canonic updates

Table 4.1.3. *Canonic updates* for a clustering after *atomic events* on the graph

description	*formal, abbrev.*	*canonic clustering update*
node insertion	$V + u$	$\tilde{\mathcal{C}} := \mathcal{C} + \{u\}$
node removal	$V - u$	$\tilde{\mathcal{C}} := \begin{cases} \mathcal{C} \setminus \{\mathcal{C}(u)\} & \text{if } \mathcal{C}(u) \text{ is a singleton in } \mathcal{C} \\ (\mathcal{C} \setminus \mathcal{C}(v)) \cup \{\mathcal{C}(v) \setminus \{v\}\} & \text{otherwise} \end{cases}$
edge creation	$E + \{u, v\,[, \omega]\}$	$\tilde{\mathcal{C}} := \mathcal{C}$
edge removal	$E - \{u, v\}$	
weight increase	$\omega(u, v) + x$	
weight decrease	$\omega(u, v) - x$	
time step	$t + 1$	$(\tilde{\mathcal{C}} :=) \quad \mathcal{C}_{t+1} := \mathcal{A}(G, \mathcal{C}(G), \Delta)$

case it was a *singleton*—upon node removal. As discussed above, a *time step (event)* serves as a signal to an algorithm to compute an updated clustering. We call a clustering $\tilde{\mathcal{C}}$ which has only been updated *canonically* and not explicitly output by a dynamic clustering algorithm, a *preclustering*, and usually denote it by $\tilde{\mathcal{C}}$. Obviously, this generalizes to non-*atomic changes*.

preclustering $\tilde{\mathcal{C}}$

From a theorist's point of view it hardly matters whether a dynamic clustering algorithm takes as the input $G_t, \mathcal{C}(G_t), \Delta$, or rather $G_{t+1}, \tilde{\mathcal{C}}(G_{t+1}), \Delta$, but for an actual implementation this is a decision that does matter, as we shall see in our experiments in Section 4.3. However, the concept of a *preclustering* offers more than just a formalism and a design decision for an implementation: We could build upon the actions listed in Table 4.1.3 and state different updates resulting in a "better" *preclustering*, helping an algorithm—we will do this in Section 4.3. Until then, and unless noted otherwise, *preclusterings* will always be in accordance to Table 4.1.3.

***Online* Dynamic Clustering Tasks.** In this pargraph we briefly formulate reasonable *online* problem statements for clustering dynamic graphs, in order to clarify our view of what meaningful tasks are. These shall guide us in the sections to come. We will come back to *offline* tasks in Section 4.5. The *dynamic clustering problem* is to update a given clustering with respect to a static clustering algorithm $\mathcal{A}$ when the associated graph *changes*.

dynamic clustering problem

Problem 5 *Given a graph G, a clustering technique $\mathcal{A}$, a clustering $\mathcal{C} = \mathcal{A}(G)$ and a graph change Δ, with $\Delta(G) = G'$. Compute a clustering $\mathcal{C}' = \mathcal{A}(G')$.*

Naturally, Problem 5 can be solved by applying $\mathcal{A}$ to the modified graph G'. Adding postulations for *speed*, *quality* and *smoothness*, we obtain various possible multicriteria formalizations, which can involve somehow harnessing the previous result $\mathcal{C}$, e.g.:

speed

1. Add to Problem 5: with minimum running time.

smoothness

2. Add to Problem 5: with maximum *smoothness* among all correct solutions.

These statements are reasonable for algorithms like *centrality removal*, *min-cut tree clustering* or GMC. However, many clustering approaches such as those based on greedy index maximization are heuristic in nature and do not provide actual quality guarantees, furthermore many are nondeterministic. Thus the terms *correct solution* and *quality* are hardly reasonable. Softening these notions we get, e.g.:

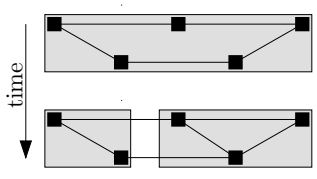
quality

3 Add to Problem 5: with maximum quality.

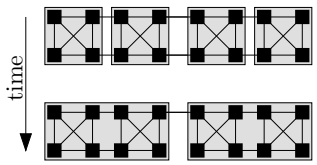
4 Add to Problem 5: with quality no less than α times the value a re-clustering from scratch would yield and otherwise with minimum running time.

multicriteria

5 Add to Problem 5 and item 4: such that a certain *smoothness* is guaranteed.



(a) Inserting an intra-cluster edge yields two cliques.



(b) Deleting an inter-cluster edge changes density.

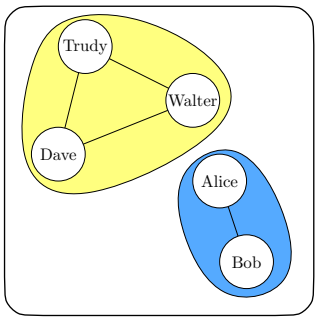
Figure 4.1.2. Examples of counterintuitive behavior

We will quantify the notion *smoothness* later (see Section 4.3.1.1) using our work from Section 2.6, but we strongly argue that this criterion, which, roughly speaking, avoids that other optimization criteria—or even indeterminism—lead to orthogonal clusterings for two consecutive graphs that hardly differ. Maintaining as much of a previous clustering as possible does not only increase the perceptibility of a result but also corresponds to the mostly continuous nature of network evolution in general. In the field of graph drawing, this additional restriction is also called the preservation of the *mental map* [152].

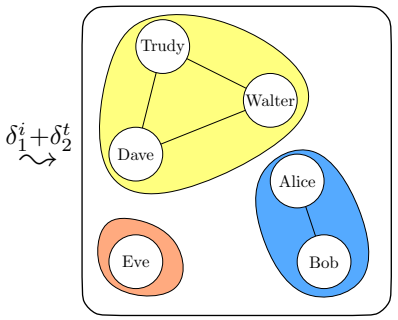
In practice, the above formulations might still be too strong, but formulation 5 is very close to what we pursue in Section 4.3 and we precisely use formulation 1 in Section 4.4. We conclude with giving a thought-provoking impulse. Obeying the paradigm of *inter-cluster density vs. intra-cluster sparsity*, one could expect that creating an additional intra-cluster edge should only *strengthen* an existing clustering and vice versa. However, density criteria can very well lead to counterintuitive behavior, as proposed in Figures 4.1.2a and 4.1.2b.

4.1.3.3 An Example of a Dynamic Graph Clustering

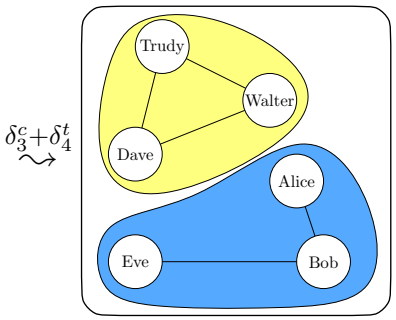
The notions and definitions in the preceeding subsection are nothing deep, however, their nomenclature potentially a bit overwhelming. This series of figures illustrates an evolving network of friendships and a dynamic clustering (colored splinegons) thereof, trying to exemplify these definitions. We take on the view of a (fictive) clustering algorithm, which observes the changing graph and springs into action (updates the clustering) as soon as a *time step event* arrives, processing *batch updates* en bloc. Between *time step events* we only have a preclustering (dashed), $\mathcal{A}$ does nothing there.



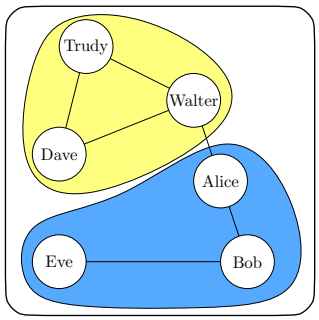
(a) The initial friendships G_0 and $\mathcal{C}(G_0)$



(b) Eve joins in...
δ_1^i = insert(Eve)
$\delta_2^t \circ \delta_1^i(G_0) = G_2 \overset{\mathcal{A}}{\rightsquigarrow} \mathcal{C}(G_2)$

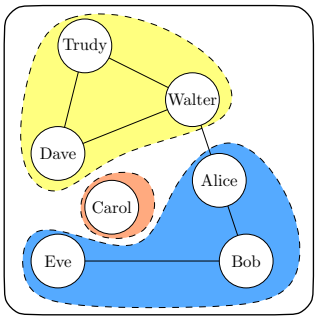


(c) ...and connects to Bob
δ_3^c = connect(Eve,Bob)
$\delta_4^t \circ \delta_3^c(G_2) = G_4 \overset{\mathcal{A}}{\rightsquigarrow} \mathcal{C}(G_4)$

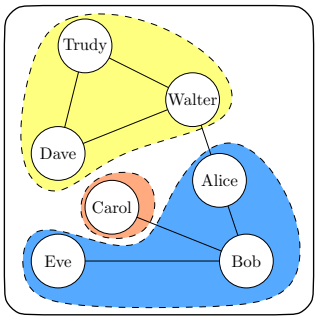


(d) Alice gets to know Walter
δ_5^c = connect(Alice,Walter)
$\delta_6^t \circ \delta_5^c(G_4) = G_6 \overset{\mathcal{A}}{\rightsquigarrow} \mathcal{C}(G_6)$

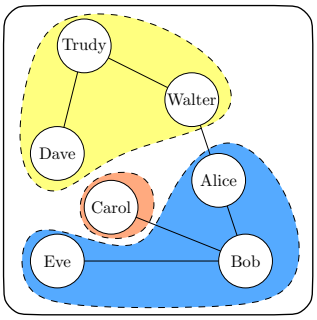
three atomic *batch updates*:
$\Delta_1 = (\delta_1^i, \delta_2^t)$
$\Delta_2 = (\delta_3^c, \delta_4^t)$
$\Delta_3 = (\delta_5^c, \delta_6^t)$
(one proper event plus one delimiting *time step event* each, $b = 1$)



(e) Carol joins in...
δ_7^i = insert(Carol)
$\delta_7^i(G_6) = G_7 \rightsquigarrow \tilde{\mathcal{C}}(G_7)$



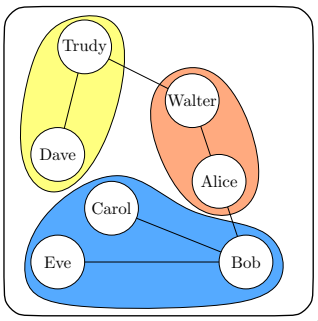
(f) ...and connects to Bob
δ_8^c = connect(Carol,Bob)
$\delta_8^c(G_7) = G_8 \rightsquigarrow \tilde{\mathcal{C}}(G_8)$



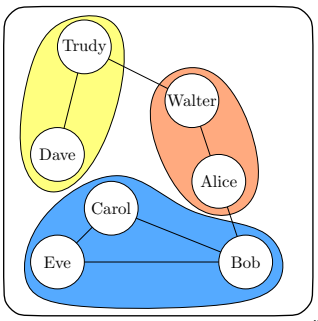
(g) Dave and Walter argue
δ_9^d = discon(Dave,Walter)
$\delta_9^d(G_8) = G_9 \rightsquigarrow \tilde{\mathcal{C}}(G_9)$

one *batch update*:
$\Delta_4 = \delta_{7\text{-}10}$
(three proper events plus one delimiting *time step event*, $b = 3$) see next Subfigure 4.1.3h for the clustering thereof

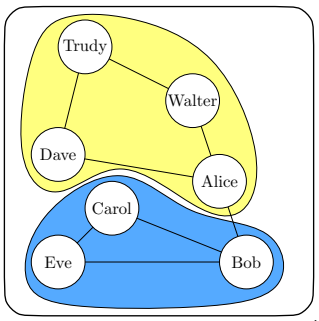
The lack of a *time step event* tells Algorithm $\mathcal{A}$ to treat $\delta_4, \delta_5, \delta_6$ as a compound event; thus, $\mathcal{A}$ does not compute $\mathcal{C}(G_{7-9})$



(h) After δ_{10}^t, $\mathcal{A}$ computes $\mathcal{C}(G_{10})$:
$G_{10} \overset{\mathcal{A}}{\rightsquigarrow} \mathcal{C}(G_{10})$
(trivial: $\delta_{10}^t(G_9) = G_9 = G_{10}$)



(i) Carol befriends Eve
δ_{11}^c = connect(Carol,Eve)
$\delta_{12}^t \circ \delta_{11}^c(G_{10}) = G_{12} \overset{\mathcal{A}}{\rightsquigarrow} \mathcal{C}(G_{12})$



(j) Alice likes Dave
δ_{13}^c = connect(Alice,Dave)
$\delta_{14}^t \circ \delta_{13}^c(G_{12}) = G_{14} \overset{\mathcal{A}}{\rightsquigarrow} \mathcal{C}(G_{14})$

$\mathcal{C}(G_{10})$ from the above Δ_4, and two more atomic changes:
$\Delta_5 = (\delta_{11}^c, \delta_{12}^t)$
$\Delta_6 = (\delta_{13}^c, \delta_{14}^t)$

Figure 4.1.3. An example scene of a dynamic graph, various proper *updates* δ^* transform the clustering structure and interspersed *time step events* δ^t, delimiting *batches* Δ, call $\mathcal{A}$ to work on the current *preclusterings* $\tilde{\mathcal{C}}$.

Section 4.2

A Generator for Dynamic Clustered Random Graphs

Nowhere is longer safe
The earth moves under our feet
The great world tree Yggdrasil
Trembles to its roots

(Tattered Banners and Bloody Flags, Amon Amarth)

The experimental evaluation of graph algorithms for practical use often involves both tests on real-world data and on artificially generated data sets. In particular, the latter are necessary for systematic and very specific and targeted evaluations. In the context of dynamic clustering algorithms, roughly speaking, we are interested in the generation of dynamic random graphs that feature a community structure of scalable clarity such that (i) the graph changes dynamically by node/edge insertions/deletions and (ii) the graph incorporates a clustering structure (communities), which also changes dynamically. Without such artificial data, any evaluation of dynamic clustering algorithms for practical use will suffer. Despite the wide variety of generators for random graphs, to the best of our knowledge, the literature has not yet tackled such dynamically changing preclustered graphs.

The line of random graph generators for static graphs—at least for our purposes—reaches back to the prominent and fundamental *Erdős-Rényi* model [85], also known as $\mathcal{G}(n, m)$, Gilbert's model $\mathcal{G}(n, p)$ [107]. This model was cast into a generator for random preclustered graphs for the purpose of experiments on clustering algorithms in [47, 48]. We further this line by adding an intuitive mechanism of dynamics to the preclustered Erdős-Rényi model. For more detailed material on other graph generators we refer the reader to [222, 71, 34] and references therein.

In this section we describe a random graph generator which is based on the Erdős-Rényi [85] model but adds to it tunable dynamics and a tunable and evolving clustering structure. More precisely, an evolving *ground-truth* clustering known by the generator motivates the *changes* to the graph by sound probabilities, such that the observable clustering changes accordingly. Thus, our generated graphs have the following properties:

- *dynamic*, i.e., representing the change of a network in the course of discrete time
- *clustered*, i.e., exhibiting a clustered structure based on *intra-cluster density* versus *inter-cluster sparsity* of edges
- *random*, i.e., generated according to a probabilistic model

One can think of the generator as a producer of *events*. They include small-scale *events* such as the creation or removal of a single edge or the introduction or removal of a node (see Section 4.1), but also large-scale *clustering events*, which cause clusters to gradually split or

merge. Such a generator allows us to examine how dynamic clustering algorithms cope with changing graphs with respect to an array of parameters such as its density, the clarity or the granularity of the *ground-truth* clustering, the *batch size* of changes, the size distribution of the *ground-truth* clusters, etc.

We detail our implementation as a module of the software tool *visone* and as a standalone tool, alongside the data structures we use.[4] Our software is free for use and download. Thus, if you do not want to hassle with any further introduction or details, we point you straight to Section 4.2.5.3 for how to download our implementation, and to Section 4.2.5.1 for input parameters and for how to read the output file. Otherwise we cordially invite you to continue reading and thoroughly learn about the mechanics of this generator.

While the field of random graphs and their generation is beautiful, and we even strayed into it before, in Section 2.3, the necessity to actually have dynamic instances for the systematic evaluation of algorithms for dynamic graph clustering gave birth to this project. In a number of cooperations we have laboriously collected several reliable real-world instances, which are very valuable for a representative assessment of how algorithms behave in practice (see Section 5.1.1). However, these instances are still few in number, they are very specific and are often subject to a confidentiality agreement. For controlled and focused experiments, a highly customizable generator is inevitable. The motivation behind this implementation was to have each parameter represent a proper stochastic value with an intuitive interpretation on the one hand and an effect which can precisely and mathematically be explained on the other hand. In its current version, the generator is an easy-to-use Java package, which by the choice of reasonable default values can be used off-the-shelf. Its compact binary output can be parsed with a simple procedure as detailed in Section 4.2.5.1. Most of the content of this section alongside the ready-to-use implementation has recently been made available in a technical report [120], based on joint work with Christian Staudt. I hope our generator finds application in the community it has been made for; we shall thoroughly use it in Section 4.3.

Main Results

- We provide a ready-to-use generator for dynamic random graphs with an implanted clustering structure. The generator works off-the-shelf and is downloadable as a Java package. (Sections 4.2.5 and 4.2.5.3)
- As the core of our generator, we devise a random process which follows a slowly changing *ground-truth* clustering in a sound probabilistic setup: At any time, edge insertions and deletions occur with a probability strictly proportional to specified values of p_{in} and p_{out}. We propose a data structure which supports this process in $O(\log n)$ time per update. (Section 4.2.4.5 and Lemma 4.2.1)
- We propose and implemented reasonable mechanisms for several ongoing dynamic processes which simulate real evolving networks. Among these are the evolution (e.g., a slow coarsening) of the *ground-truth* clustering, node volatility (churn) and a mapping of the *ground-truth* clustering to a reference clustering. (Sections 4.2.4)

Future Work As promised in Section 4.2.3, a GUI-version of the generator as a module for visone (a tool for the visualization and analysis of social networks) is waiting in the wings, but has to hang on until dynamic graphs are fully incorporated into an upcoming official release. Apart from engineering to reduce both space and running time consumption, there are two main issues that we plan to address in the near future: First, while the specification of values for our main parameters p_{in} and p_{out} is very handy it might sometimes be more convenient to set values for the average degree of a node, both for intra- and for inter-cluster

[4] At the time of finishing this document, no official release of a version of visone supporting dynamic graphs has been released. Thus we recommend the usage of the standalone version for the time being, see Section 4.2.5.3 for more information.

edges. This option will soon be integrated as it can easily be incorporated into the current data structures. Second, an aspect of dynamics that has not yet been realized is a gradual densification or sparsification of the network—or of parts of it.

Another algorithmic aspect is the question whether there is a different data structure for weighted selection which is similarly fast but uses less space. Our approach uses quadratic space but works very quickly and thus favors medium-size and very long running dynamic graphs.

4.2.1 The Rough Picture

a real-world analogy

In order to provide the reader with an informal overview of our approach, we will now sketch out an analogy of the generation process on a rough scale. In later sections we will then delve into the details. We thus avoid technicalities here and leave a number of questions open. With some slightly synthetic assumptions, the generator can be thought of as the head of some department organizing his personnel (the nodes) which collaborates (via edges) into groups (clusters).

Projects Come and Go. The head of department initially organizes his co-workers (i.e., the nodes of a graph) into groups such that each group works on a different project. From time to time, projects are finished or new ones are launched; however—as in real life—projects are not neatly scheduled sequentially, but they overlap or end before the next one arrives in a pretty random fashion. In case a project ends, the persons that were handling it are now available for other tasks, thus they are assigned to another project and assist the group which has already been working on it. The head of department then merges the two groups. In case a new project is launched, a new group needs to be assigned to it; to this end, an existing working group is split such that some people stay at the old project and others move on to the new one. This is how groups (i.e., clusters) evolve.

clusters split and merge

Collaborations Arise and Conclude. Suppose now a certain set of projects is being worked on. By any means people working on the same project (i.e., within the same group) need to collaborate heavily and rely on one another. However, these collaborations do not pop up the instance a project is launched, but they gradually evolve. On the other hand, people in different projects rarely need to collaborate. However, two persons that are newly separated into different groups might not immediately shut down their collaboration but might do this with some delay. This is how relations of collaboration (i.e., edges) evolve.

edge insertions and deletions

Co-Workers are Hired and Fired. Finally, as a process which is more often than not (and in our case *always*) independent from projects, our department has a certain fluctuation of personnel. On the one hand, new co-workers are employed and join some group – and immediately build up collaborations (otherwise they don't know what needs to be done). On the other hand, people leave the department or are fired, immediately breaking up their collaborations. The department might have a general tendency to grow, shrink or maintain its average manpower. This is how the set of co-workers (i.e., nodes) evolves.

nodes join and leave

Plans vs. Reality. As a last preparation for the concepts described later, consider how the department's personnel chooses tables during lunch break. On the long run, each project group will happily gather together for lunch to discuss open questions within their project. Thus the grouping during lunch break will match the organizational structure. However, a newly broken up group will still have a lot to discuss and might want to have lunch together; conversely a newly merged group might not yet know each other. To summarize things, the community structure during lunch follows the organizational structure with some delay. Gradually the arising and concluding collaborations have it adapt to the group structure, but an outside observer (during lunch break) will not be able to discern the project groups

ground truth $\mathcal{C}$ $\neq$ reference

correctly until this has happened to a sufficient degree. This is how the observable group structure (i.e., the set of observable clusters) evolves. It is crucial to grasp the difference between what governs changes in collaboration (edges), namely the *ground truth* given by the projects' group structure, and the observable group structure that is more likely to be discovered by observers (clustering algorithms) who can only see people and their current collaboration structure (i.e., the graph).

However, as any observer is subjective, the clustering she observes will often differ from the clustering the generator deems *observable*. Thus it is helpful to keep in mind that there are three clusterings around: (i) a *ground-truth* clustering, which motivates the changes in the graph, (ii) a reference clustering which the generator deems observable, and (iii) the clustering which a subjective observer identifies. In a static scenario the former two are equal, and—given an algorithm that perfectly agrees with the generator—the latter two agree as well.

three clusterings

4.2.2 Definitions and Preliminaries

The Static Case. Generators for static graphs with an implanted clustering structure have been proposed and used in several works [222, 71, 34, 47, 48]. We only briefly review the idea taken from [47], as it is an easy to use and intuitive technique, derived from one of the oldest approaches on random graphs [85], and constitutes the base case for our dynamic generator. The Erdős-Rényi model [85] creates for a given set V of n nodes an edge between each pair of nodes with a uniform probability, such that the expected number of edges in the graph is some fixed parameter. For brevity we pass over the large array of works that deal with such random graphs.

The *random preclustered graph generator* [47] needs two such edge probabilities: the *intra-cluster edge probability* p_{in} for node pairs within clusters and the *inter-cluster edge probability* p_{out} for node pairs between clusters. Given such probabilities the generator then predetermines a partition of V in some fixed or random manner and sets the elements of the partition to be the clusters. Given this clustering of an edgeless graph, edges are introduced according to p_{in} and p_{out} as in Definition 4.1:

the static start

p_{in} and p_{out}

Definition 4.1 *For each pair of nodes $\{u, v\}$, its edge probability is defined as*

$$p(u,v) = \begin{cases} p_{in}(C) & \text{if } u, v \in C \\ p_{out} & \text{else} \end{cases}$$

The choice of these two parameters that govern edge density, p_{in} and p_{out}, determines the "clarity" of the clustering that is implanted into the random graph.

A typical evaluation run for some static graph clustering algorithm could thus look like this: Take the above generator and preset some n and some $|\mathcal{C}|$, then let p_{in} and p_{out} iterate through some range of values and for each choice let the clustering algorithm tackle the output graph. This can be done until, e.g., statistical significance with respect to some quality or runtime measurement is attained, and shows how well the algorithm works on dense or sparse graphs with a clear or rather obfuscated clustering structure. A comparison to the quality of the *ground-truth* clustering known to the generator can be useful as well.

p_{in}, p_{out} tune clarity

In order for the result to be a clustered graph according to the density vs. sparsity paradigm, these probabilities p_{in} and p_{out} should be chosen such that $\forall C : p_{\text{in}}(C) > p_{\text{out}}$. However, note that in the common case that the size of clusters is in $o(|V|)$, the parameter p_{out} has great impact on obfuscating the clustering as it affects far more node pairs than p_{in}; this means that although the above condition holds true, far more inter-cluster edges than intra-cluster edges may be expected. Being aware of this pitfall, we avoid the adaptation of [71] where p_{out} is replaced by the ratio of inter- to intra-cluster edges.

p_{out}'s impact

Choices for Dynamics. As the reader might already suspect, our dynamic generator as sketched out in section 4.2.1, is parameterized by a number of options to steer the randomness. How often do groups split, are edges more prone to changes than nodes, how quickly do edges adapt to the planned clustering? We postpone details on our procedures and parameters to the next section, and start very simple.

In a nutshell, the generator maintains a clustering ζ in a sequence of discrete *time steps*. This clustering indirectly steers where edges are randomly created or removed as it steers the probabilities with which such *events* happen: Each cluster C has the universal or an individually associated *intra-cluster edge probability* $p_{\text{in}}(C)$. Together with (the universal) p_{out}, the *inter-cluster edge probability* of the current graph G_t, this yields an edge probability for each pair of nodes as noted above.

However, we do not only want to have dynamics in the set of edges, we also want the set of nodes to dynamically change, and—as sketched out above—we even want the clustering to change. After some brief words on *visone*, we detail these mechanics.

4.2.3 Java Implementation Based on Visone

visone

This project was started as an extension to *visone*[5], an application designed for the analysis and visualization of social networks. Visone has been started as a project within the priority program *Algorithmics of Large and Complex Networks* (SPP 1126) of Deutsche Forschungsgesellschaft (DFG), and is now maintained at Universität Konstanz. In a graphical user interface this tool provides all general tools for graph manipulation and editing but also many methods for tasks of visualization and analysis. A recent feature—which still has beta status—is the support of dynamically changing networks and their smooth visualization [35], a tool of great value for the initial evaluation of our generator, which we were lucky to have access to, thanks to our co-workers Michael Baur and Thomas Schank.

Unfortunately, things recently slowed down in the development of a new release of *visone*. Depending on a usable and publicly available generator we thus moved away from our prototypical version for *visone* to a standalone version. Although we still plan to integrate our generator into *visone*, once a future official version that supports dynamic graphs is released, we do not discuss our *visone*-module any further, but just give a teaser screenshot (Figure 4.2.1) of the plugin as a first impression and continue with detailing the standalone generator.

4.2.4 Description of Generator Mechanics

In this section we detail the procedures we use to generate a dynamically changing preclustered graph. We recommend an occasional glance at the generator's schematic decision tree given in Figure 4.2.2 for an overview. Technical details on how to use the Java tools are given later.

4.2.4.1 Decision Tree

overview: decision tree

Figure 4.2.2 shows the generator's decision tree. Decision nodes are drawn as a rhombus while operations are drawn as a rectangle. For each decision node a pseudo-random number $x \in [0, 1)$ is generated and then compared to p . If $x \leq p$, the first branch will be taken, if $x > p$, the second branch will be taken. Before we detail how decisions and operations are done in later sections, we give a rough overview of how, given an initial instance, the generator produces a single *time step*, a process which is iterated until the desired number of *time steps* has been generated.

In each *time step*, two bigger decisions are made, the first of which is whether a change in the clustering is to be attempted (with probability p_ω) or not (otherwise). In the affirmative case, a *split event* is chosen with probability p_μ, otherwise a *merge event* is chosen. The second decision is whether to perform an *edge event* (with probability p_χ) or a *node event*

[5] `http://www.visone.info`

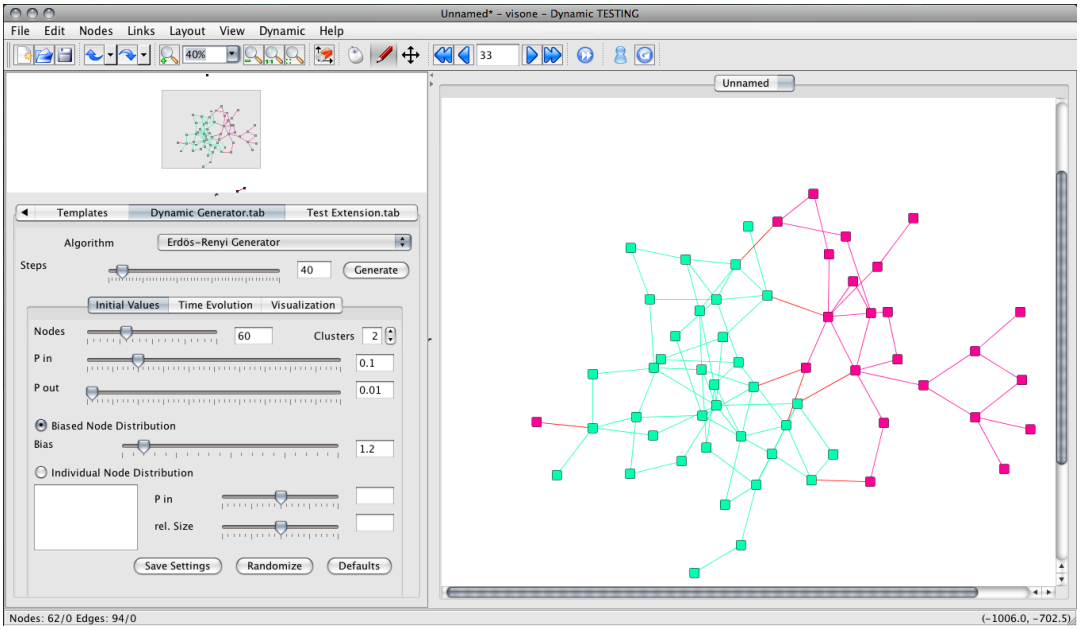


Figure 4.2.1. Screenshot of *visone* and its toolbar for the dynamic generator

(otherwise). For an *edge event* we then decide—in a non-trivial manner—whether to add or remove an edge, and which edge this shall be. For a *node event* a similar but simpler choice of whether to add (with probability p_ν) or delete (otherwise) is made. When a new node is added, it will instantly be connected to the existing nodes inside and outside of its cluster, according to p_{in} and p_{out}, respectively. Conversely, when a node is removed, its incident edges are also removed in the same step.

4.2.4.2 Initial Instance

The starting state of the dynamic graph is constructed in a way similar to [47, 101]. Given a number n of initial nodes and a number k of initial clusters, we choose uniformly at random for each node to which cluster it shall belong; i.e., for each node v, $v \in C_i$ if $x \in [i/k, (i+1)/k)$, for a pseudorandom number $x \in [0, 1)$. For each cluster this yields a *binomially* distributed size around the expected size n/k. This converges toward the *normal* distribution for large n. Once each node is assigned to some cluster, edges are drawn. Each inter-cluster node pair becomes connected with probability p_{out}; each intra-cluster node pair becomes connected with probability $p_{\text{in}}(C)$, which can be universal or specific to each cluster.

initial cluster sizes

Biased Selection. If a uniform distribution of cluster sizes is not desired, a skewed distribution can be enforced. This is done by introducing an exponent β for the random number which selects a cluster from the set of all clusters. Raising the pseudo-random number $x \in [0, 1)$ to the power of β, for some $\beta \geq 1$, returns $x' \leq x$. The formerly uniform distribution of the random number is thereby shifted to the lower end of $[0, 1)$. In order to select an element from an array a with bias we calculate the index

$$i = \lfloor x^\beta \cdot |a| \rfloor \tag{4.2.1}$$

and return the element at $a[i]$. Thus the elements at the beginning of the list have a higher probability of being selected, depending on β.

As a method for imbalancing cluster sizes, biased selection is nothing particularly sophisticated, but serves the general purpose and is very simple to implement and understand;

imbalance via biased selection

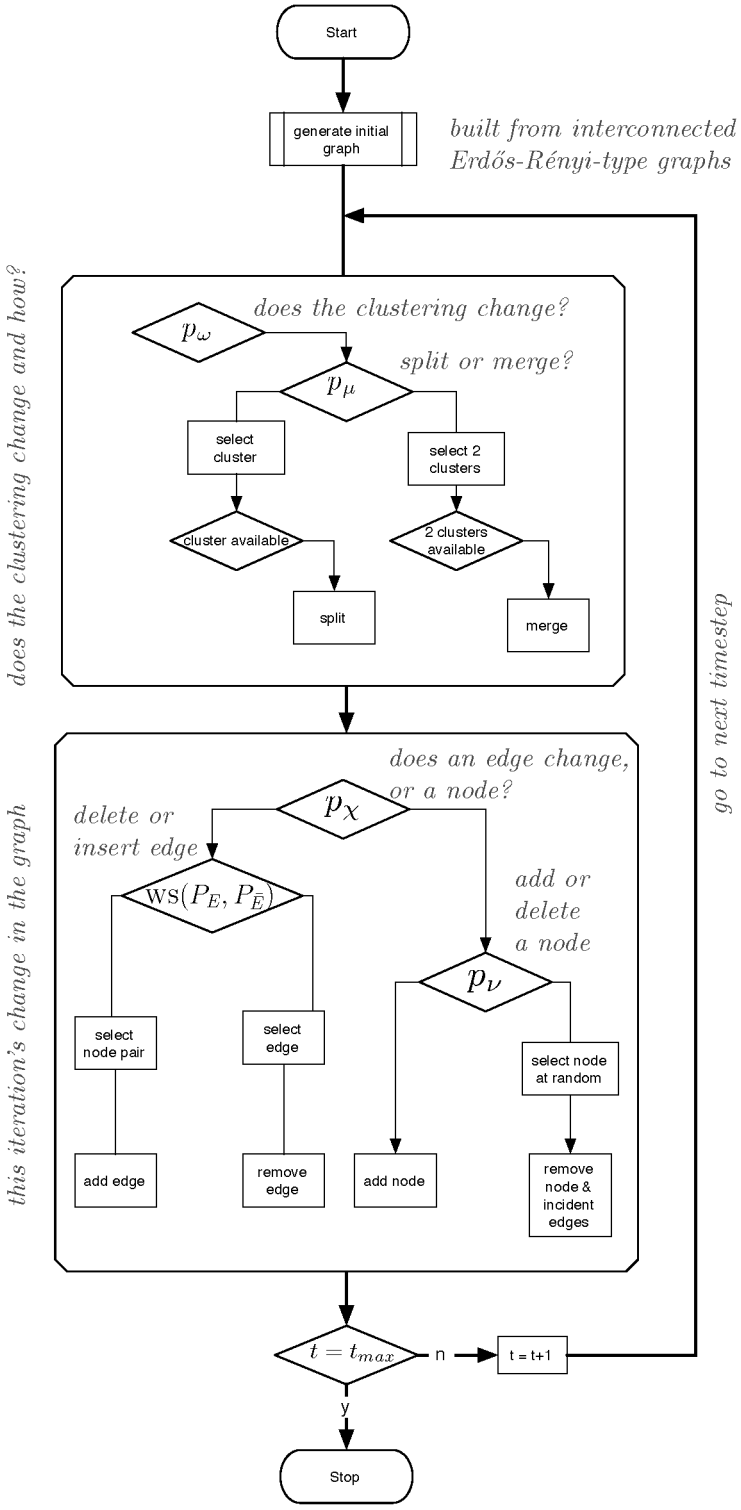
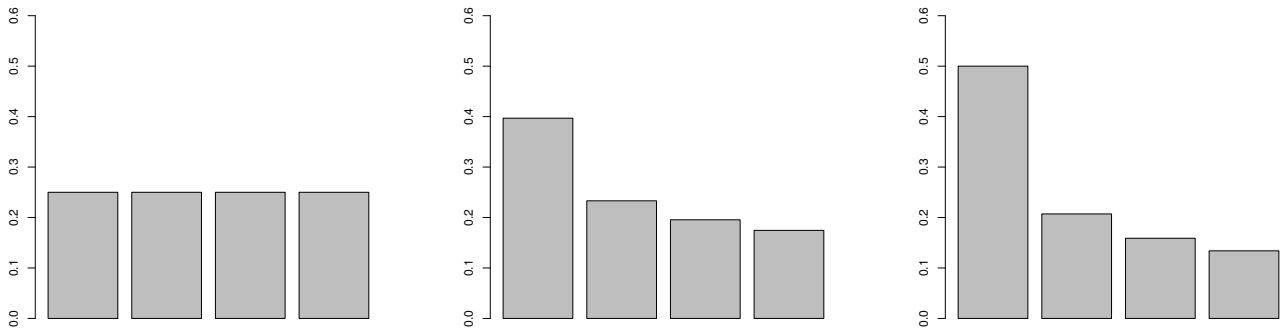
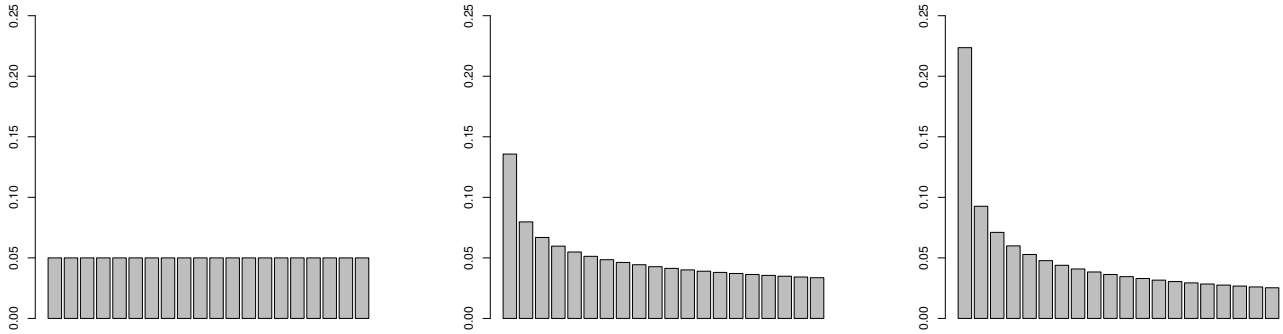


Figure 4.2.2. Schematic decision tree of the generator



(a) $k = 4$, $\beta = 1.0$ **(b)** $k = 4$, $\beta = 1.5$ **(c)** $k = 4$, $\beta = 2.0$

Figure 4.2.3. Expected fractions of $|V|$ in each cluster for $k = 4$, using different values of β for biased selection.



(a) $k = 20$, $\beta = 1.0$ **(b)** $k = 20$, $\beta = 1.5$ **(c)** $k = 20$, $\beta = 2.0$

Figure 4.2.4. Expected fractions of $|V|$ in each cluster for $k = 20$, using different values of β for biased selection.

moreover, as visible in Figures 4.2.3 and 4.2.4, it favors few larger clusters and many smaller clusters of similar size, a setting we frequently observed in real-world data sets, e.g., in the network of Autonomous Systems in Figure 3.2.11. Note that choosing $\beta \leq 1$ yields the opposite effect, amassing probability mass at the upper end of the interval; this yields a different scenario with several larger clusters and only few small ones. It could easily be substituted by any other technique or requirements to cluster sizes; however, keep in mind that the dynamic process of splitting an merging clusters deteriorates any fixed initial distribution of cluster sizes—even though we again use biased selection here (see below). For the splits in particular we plan future methods that try to stay as close as possible to the initial distribution of cluster sizes. For a rough impression of the impact of β, observe the following formula which expresses the expected fraction of nodes in cluster C_i. They directly derive from Equation (4.2.1).

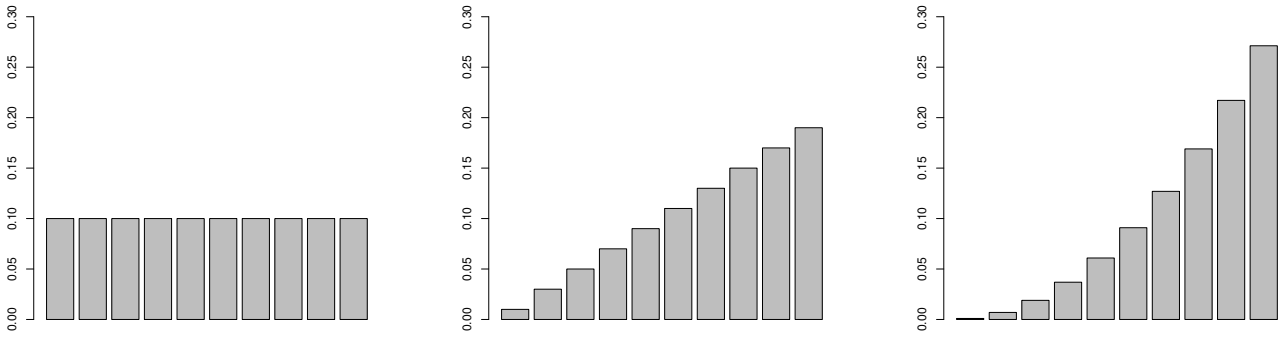
distr. of biased selection

$$\mathbb{E}\left(\frac{|C_i|}{n}\right) = \underbrace{p(x^\beta \leq \frac{i}{k})}_{\substack{p(\text{place node in} \\ \text{Clusters } C_1,\ldots,C_i)}} - \underbrace{p(x^\beta \leq \frac{i-1}{k})}_{\substack{p(\text{place node in} \\ \text{Clusters} C_1,\ldots,C_{i-1})}} = \sqrt[\beta]{\frac{i}{k}} - \sqrt[\beta]{\frac{i-1}{k}} \tag{4.2.2}$$

As an example, the expected fractional sizes of clusters for $k = 4$, $k = 20$ and $k = 10$ and different values of β according to Equation (4.2.2) are displayed in Figures 4.2.3, 4.2.4 and 4.2.5 respectively.

4.2.4.3 Dynamics in the Clustering

Since the main purpose of the generator is to produce test instances for clustering algorithms, it has to maintain a valid clustering which governs edge density and which those clusterings



(a) $k = 10$, $\beta = 1.0$ **(b)** $k = 10$, $\beta = 0.5$ **(c)** $k = 10$, $\beta = 0.33$

Figure 4.2.5. Expected fractions of $|V|$ in each cluster for $k = 10$, using different values of $\beta \leq 1.0$ for biased selection.

current clustering $\zeta(G_t)$

cluster events

reference clustering $\zeta_{\text{ref}}(G_t)$

threshold

found by algorithms can be compared to. On the other hand, as the generator allows the underlying clustering to change dynamically, we maintain two separate clusterings of the graph. We will call the clustering that is used by the generator itself to produce the edge structure the *current clustering* $\zeta(G_t)$, being the *ground truth* which the graph dynamically tries to adapt to (compare to the project groups in Section 4.2.1). We describe below how clusters are split or merged in *cluster events*. The crucial point is that when a cluster operation has just been initiated, the edge density of the graph still corresponds to the previous clustering, which can consequently match or be close to the clustering that a good clustering algorithm will identify in the graph. So in order to evaluate the performance of a clustering algorithm, the generator has to have the previous clustering in store, which we will call the *reference clustering* $\zeta_{\text{ref}}(G_t)$. After several steps in which the edge distribution increasingly incorporates the new *ground truth*, this clustering will become visible in the graph and the former one will vanish. At some point determined by the generator, the *cluster event* is considered completed, and $\zeta_{\text{ref}}(G_t)$ is updated to incorporate the resulting change. We discuss below how we determine this point in time called the *threshold*. Our implementation allows multiple such processes simultaneously (but no cluster is multiply involved), i.e., further *cluster events* can be initiated before the last one has concluded by reaching its threshold.

split

merge

Splitting and Merging Clusters. A cluster C_1 is split by distributing its nodes to two new clusters C_2 and C_3 (formally written as $C_1 \to (C_2, C_3)$). The nodes are distributed using biased selection. The current implementation uses an exponent of 1, so the nodes are distributed equally. Two clusters C_1 and C_2 are merged by combining their nodes to form a new cluster C_3, which we will denote with $(C_1, C_2) \to C_3$. In case p_{in} is universal we are done for both cases; when using cluster-individual p_{in} values, different methods can be imagined for setting the p_{in} of the resulting clusters:

inherit/avg. p_{in}

a) For the split operation $C_1 \to (C_2, C_3)$, C_2 and C_3 inherit their p_{in} from C_1. For the merge operation $(C_1, C_2) \to C_3$, $p_{\text{in}}(C_3)$ is set to the arithmetic mean of $p_{\text{in}}(C_1)$ and $p_{\text{in}}(C_2)$. It might be an undesired effect of this method that the values tend to become more and more uniform in the course of time. Therefore, another method was implemented:

draw new p_{in}

b) The second method tries to estimate a Gaussian distribution from the initially given list $[p_{\text{in}}(C_1), \ldots, p_{\text{in}}(C_k)]$ and generates new p_{in} values randomly according to this distribution. This is done in order to preserve the initial diversity of p_{in} values over the course of time. A new p_{in} is determined via a random variable X with a Gaussian distribution, see Equation (4.2.3), where μ is the arithmetic mean of the list values and σ^2 is the variance of the list values relative to μ, see Equation (4.2.4).

$$X \sim N(\mu, \sigma^2) \tag{4.2.3}$$

$$\sigma^2 = \frac{1}{k}\sum_{i=1}^{k}(p_{\text{in}}(C_i) - \mu)^2 \tag{4.2.4}$$

Then, X can is calculated as in Equation (4.2.5), where $Y \sim N(0,1)$ is generated by the method `java.util.Random.nextGaussian` :

$$X = \sigma Y + \mu \tag{4.2.5}$$

As this might result in values beyond feasibility, if X is not in $[0,1]$, it is recalculated until it can be interpreted as a probability.

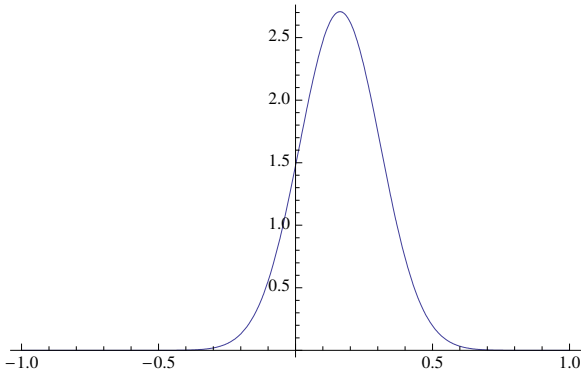


Figure 4.2.6. Estimated distribution for `pIn=[0.1, 0.2, 0.5, 0.25]`

reference catches up with ground truth

Threshold for the Completeness of a Cluster-Event. As mentioned above, the reference (observable) clustering follows the current (*ground-truth*) clustering with some delay. The motivation for it is, that this reference is what the generator deems observable, and since it takes some time for the graph to adapt to a changed *ground-truth* clustering, the latter clustering is almost impossible to guess by an observer. Exactly when the reference clustering is considered to have caught up—at least to some extent—is decided by the threshold value and the edge densities within or between participating clusters. Note that as long as a split or merge operation is in progress, the clusters participating cannot be involved in another operation. The resulting clusters become available again as soon as the operation is "completed" to a sufficient degree. However, other concurrent operations are fine.

concurrent cluster events

Consider a merge operation $(C_1, C_2) \to C_3$ and a split operation $C_3 \to (C_1, C_2)$. We first calculate the expected value for the number of edges between C_1 and C_2 according to p_{in} and p_{out} as follows:

$$a := |C_1| \cdot |C_2| \cdot p_{\text{out}} \quad \text{(split)} \tag{4.2.6}$$

$$b := |C_1| \cdot |C_2| \cdot p_{\text{in}}(C_3) \quad \text{(merge)} \tag{4.2.7}$$

We then count the actual number of edges, $|E(C_1, C_2)|$. For a split operation to be complete, it should be close to a, and for a merge operation close to b. Exactly how close is determined by the input parameter θ, which expresses a tolerance threshold. The generator decides the completeness of a cluster operation according to

θ

$$\mathsf{Completed}(C_3 \to (C_1, C_2)) = \begin{cases} \text{true} & \text{if } |E(C_1, C_2)| \leq \theta \cdot b + (1-\theta) \cdot a \\ \text{false} & \text{if } |E(C_1, C_2)| > \theta \cdot b + (1-\theta) \cdot a \end{cases} \tag{4.2.8}$$

$$\mathsf{Completed}((C_1, C_2) \to C_3) = \begin{cases} \text{true} & \text{if } |E(C_1, C_2)| \geq \theta \cdot a + (1-\theta) \cdot b \\ \text{false} & \text{if } |E(C_1, C_2)| < \theta \cdot a + (1-\theta) \cdot b \end{cases} \tag{4.2.9}$$

For instance, if θ is 0, there is no tolerance and the cluster operation is not completed unless the expected number of edges is reached exactly; a value of $\theta = 1$ let means the operation is instantly considered completed.

4.2.4.4 Deleting and Adding Edges and Nodes

edge or node?

Edge or Node? The generator now chooses whether to manipulate a single edge or a node including its incident edges. By a single random choice an *edge event* is chosen with probability p_χ, otherwise a *node event* takes place as follows.

node ins./del.

preserves $\sim |C|$

Node Dynamics. A *node event* consists of one *atomic node event* and several associated *atomic edge events*. In case a *node event* takes place, with probability p_ν a node is newly inserted into a cluster chosen, again using biased selection (see Section 4.2.4.2 above). Otherwise a node is picked uniformly at random and removed. Note that both cases preserve expected relative cluster sizes. Both operations usually incur *changes* to edges. The removal of node v is thus automatically preceeded by the deletion of its $\deg(v)$ incident edges. In turn, inserting node v automatically entails the insertion of edges as for the initial instance. As described in Section 4.2.4.2 the generator thus connects v to an existing node u with probability $p(v, u)$.

batch updates

η

Batch Updates. A *node event* consists of several *atomic events*, which gives them much more impact than a single *atomic edge event*. However, handling a number of *events* en bloc is very reasonable for a large and quickly changing network. Therefore we additionally introduce parameter η, which enables and scales *batch updates*, i.e., *time steps* which explicitly comprise a number of *atomic edge events* of at least η. This parameter offers another dimension to the generator: *time steps* no longer solely consist of either one *edge event* or one *node event* (alongside its induced edge events), but of a scalable number of such *events*. With a given η, the generator counts *edge events* and issues a *time step* event if at least η such *events* have been performed since the last *time step*. Note that a bulky *node event* might contribute several *edge events* at once, such that more that η *edge events* can occur before a *time step* event is issued. As before, *cluster events* are issued or completed only once per *time step*.

4.2.4.5 Edge Dynamics.

towards ground truth

delete or insert

After the decision to change an edge is made, the generator has to decide whether to add or delete an edge. Ideally, the change should bring the graph closer to the aspired (*ground-truth*) clustering structure, while retaining some randomness. As in a dynamic scenario absent edges are candidates for inclusion in forthcoming states, in the following it is useful to think in terms of a graph and its *complement graph*: $\bar{G} = (V, \bar{E})$ with $\bar{E} = \binom{V}{2} \setminus E$. The first decision for edge dynamics is whether to insert or to delete an edge. As all following decisions, this decision is guided by probability masses. The probability masses for all insertions and deletions are:

$$P_{\bar{E}} := \sum_{\{u,v\} \in \bar{E}} p(u, v) \qquad \text{mass of all insertions} \tag{4.2.10}$$

$$P_E := \sum_{\{u,v\} \in E} (1 - p(u, v)) \qquad \text{mass of all deletions} \tag{4.2.11}$$

Note that these masses are equal,s in the expected case:

$$E\{P_{\bar{E}}\} = \sum_{\{u,v\} \in \binom{V}{2}} p(\{u, v\} \in \bar{E}) \cdot p(u, v) \qquad = \sum_{\{u,v\} \in \binom{V}{2}} (1 - p(u, v)) \cdot p(u, v)$$

$$E\{P_E\} = \sum_{\{u,v\} \in \binom{V}{2}} p(\{u, v\} \in E) \cdot (1 - p(u, v)) = \sum_{\{u,v\} \in \binom{V}{2}} p(u, v) \cdot (1 - p(u, v))$$

weighted selection

The choice between creating and removing an edge is made with probabilities proportional to $P_{\bar{E}}$ and P_E. To such a simple random choice we refer to as *weighted selection* (Algorithm 18)

in the following:

$$\mathsf{operation} \leftarrow \mathsf{weightedSelection}(\{\mathsf{add}, \mathsf{delete}\}, \{P_{\bar{E}}, P_E\}) \tag{4.2.12}$$

As P_E ($P_{\bar{E}}$) monotonically grows (shrinks) with the number of present edges, this mechanism mildly and continuously works towards the expected number of edges. In the following we describe how we proceed similarly for actually choosing a pair of nodes.

the general data structure

A Data Structure for Dynamic Random Choices. We briefly abstract from graphs and now describe our data structure in more general terms. Suppose we are given a set $\mathcal{O}$ of elements, with a weight $\omega(o)$ associated to each element $o \in \mathcal{O}$. Given now a time series $\mathcal{T}$ which in each *time step* with equal probability either inserts a new element o (with some random weight $\omega(o)$) into $\mathcal{O}$ or removes an element $o \in \mathcal{O}$. How can we represent $\mathcal{O}$ such that for each removal, the probability of $o \in \mathcal{O}$ to be removed is proportional to $\omega(o)$?

We can store the elements in the nodes of a complete binary tree T. We define each node of the complete binary tree to be a tuple

$$q_i = (o_i, \omega_i, l_i, r_i) \tag{4.2.13}$$

where o_i is an element, ω_i is the weight $\omega(o_i)$ of this element, $l_i = w_{i+1} + l_{i+1} + r_{i+1}$ is the sum of the weights in the left subtree and $r_i = w_{i+2} + l_{i+2} + r_{i+2}$ is the sum of weights in the right subtree. A leaf node q_ℓ's weights l_ℓ and r_ℓ are simply 0.

maintaining probabilities

Maintaining the property in Equation 4.2.13 is simple for both insertions and removals: Inserting a new element o means adding a new leaf $q = (o, \omega(o), 0, 0)$ to T and then updating all its ancestors by adding $\omega(o)$ to either l_i or r_i, depending on the subtree that includes o. Deleting an element o is done by replacing it by the last (leaf-)node o_ℓ of T and updating the ancestors of o according to the change in weight, and the ancestors of o_ℓ by subtracting $\omega(o_\ell)$. Both operations thus require a logarithmic number of updates. We detail these simple steps in Algorithms 14 and 15.

It remains to show how elements are removed with probability proportional to their weight. The procedure for the selection of an element starts at the root node by drawing a random number x from the interval $[0, w + l + r)$. Now there are three possible ranges for x: if $x \leq w$, the element is returned; if $w < x \leq w + l$, the carryover $x - w$ is sent to the left subtree; and if $w + l < x < w + l + r$, the carryover $x - w - l$ is sent to the right subtree. The procedure continues recursively from there until an element is returned after at most $\log_2 n$ steps (at a leaf). This is sketched out in Algorithm 13. We will show later that this achieves proportionality to ω.

Data Structures for Selecting Pairs of Node. We return to the generator and use the data structure proposed in the last subsection. After deciding whether to insert or remove an edge, the generator has to select an affected pair of nodes. The selection should be done in such a way that an existing edge with low $p(u, v)$ (see Definition 4.1) should have a high chance of being selected for deletion, and that an unconnected pair of nodes with high $p(u, v)$ should have a high chance of being selected for the insertion of a new edge. So a selection process where every pair is weighted according to $p(u, v)$ is desired. In fact we achieve this in a way such that each insertion (deletion) takes place with a probability exactly proportional to $p(u, v)$ $(1 - p(u, v))$.

transfer to edges

The data structure we use for storing the current graph $G(t)$ and complement graph $\bar{G}(t)$ are trees as described above. The selection of a pair of nodes as an edge or a complement edge happens in two stages: First, a source node[6] is selected, then a target node. For the selection of the source node of the edge, two binary trees, the *source trees* $\bar{T}_s$ (for edge additions) and T_s (for edge deletions), are build on $V(t)$. Each element of a tree contains a node u and is weighted with $w(u) = sum_{(u,v)\in\bar{E}(t)} p(u, v)$ and $\sum_{(u,v)\in E(t)}(1 - p(u, v))$ respectively. To each single node $u \in V(t)$, two binary trees $\bar{T}_t(u)$ and $T_t(u)$, called *target trees*, are associated.

first: source tree

then: target tree

[6] For readability we use the terms "source" and "target", albeit we deal with undirected edges.

The nodes of the former are the targets v of outgoing edges (u, v) in $\bar{G}(t)$ weighted by the (addition-) weight $w(v) = p(u, v)$ of that edge; analogously the nodes of the latter are the targets v of outgoing edges (u, v) in $G(t)$ weighted by the (deletion-) weight $w(v) = 1-p(u, v)$ of that edge. Below we give an example of such trees, and illustrate them in Figures 4.2.8 and 4.2.9.

Actually Deleting or Adding Edges. Having decided whether to add or delete an edge (see Section 4.2.4.5) we can now use the above described trees. The source tree (T_s or $\bar{T}_s$) is used to choose the source node of the change via the call to Algorithm 13. Then, using the same algorithm with the appropriate target tree ($T_t(v)$ or $\bar{T}_t(v)$) the target node is chosen.

update in $\Theta(\log(n))$

We devised these data structures to enable quick dynamic maintenance of data structures that let the generator adhere to sound probabilities. Each *edge event* is handled within logarithmic runtime $\Theta(\log(n))$. This approach lets the edge structure converge to the aspired clustering while allowing some randomness. If there are no *cluster events* to be completed, this process yields a clustered graph which is stable apart from minor fluctuations. We now give an example and then prove our claim about proportionality.

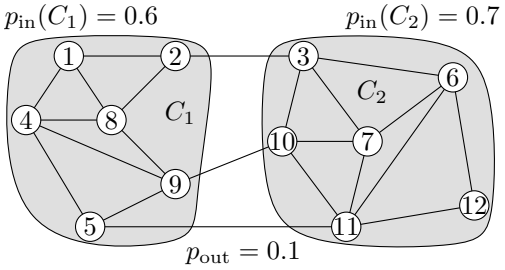


Figure 4.2.7. The graph before the edge modification. The accumulated weights for edge addition is 3.6 from C_1, 3.5 from C_2, and 3.3 from inter-cluster pairs.

Example Process for Edge Modification. We give an illustrating example here of how a specific edge modification is determined. Suppose the graph as shown in Figure 4.2.7 is given at the start of the current *time step*; suppose further that the generator decides to modify the edge set during the current *time step* (according to p_χ, see Table 4.2.1). At first, in accordance with Equations (4.2.10) and (4.2.11) the probability masses for edge deletions and edge additions are $P_{\bar{E}} = 10.4$ and $P_E = 9.3$. Suppose now weightedSelection (see Equation (4.2.12)) draws the random number 0.3 and thus decides in favor of an edge addition operation.

Having decided to insert a new edge, now the tree $\bar{T}_s$ is used to choose a random source node for the new edge. Figure 4.2.8 shows what this tree could look like. Each node carries as a weight the sum of the probabilities of its edges in $\bar{G}$, i.e., the sum of the probabilities of its missing adjacencies in G—these are the blue numbers. The red numbers depict how these weights are propagated upward through the tree. Suppose Algorithm 13 now draws the random number 0.85, yielding

example run

$x = 0.85 \cdot 20.8 = 17.68$ for the initial tree search. At $\bar{T}_s$'s root node 1 we observe that $17.68 > w(1) + l(1)$ and thus the algorithm descends into the right subtree, passing on the new value of $x = 17.68 - 1.8 - 11.8 = 4.08$. At node 3 we observe that $1.9 < 4.08 < 5.9$ and thus the left subtree is chosen, passing on $x = 2.18$. Then at node 6, since $1.3 < 2.18$ the left subtree is chosen, where we finally end up with the leaf node 12, which we thus take as the source node of the new edge.

The target node of the new edge is chosen using the target tree of 12, which is given in Figure 4.2.9. This tree stores all nodes of G which are not adjacent to 12, using the probabilities of the corresponding potential edges as weights of these candidate nodes. Anticipating our discussion below, Figure 4.2.10 shows the subintervals of $[0, 2.7)$ that are equivalent to the target tree depicted in Figure 4.2.9. Randomly drawing 0.44 yields $x = 0.4 \cdot 2.7 = 1.08$. Algorithm 13 chooses in this tree 3 as the target node. Concluding, edge $\{12, 3\}$ is inserted.

Probabilities It is important to note that the way the algorithm chooses its specific edge modification exactly complies with the following probability space: Set the probability of the specific (possible) *event* $\xi_{u,v}$, such as "insert an edge between non-adjacent nodes u and v", to $p(\xi_{u,v}) =$ proportional to $p(u, v)$, thus enabling a fair random choice. It is not hard to see, that the three steps: (i) choose between deletion and insertion, (ii) choose source node

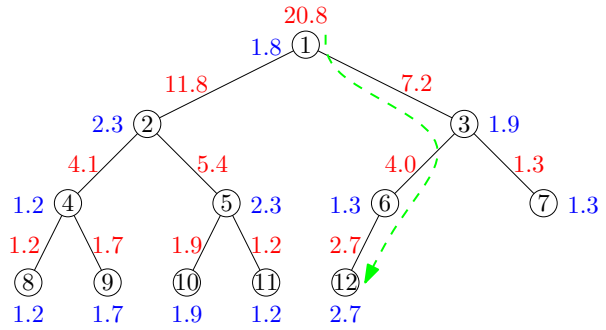


Figure 4.2.8. The source tree $\bar{T}_s$ of the graph in Figure 4.2.7, which is used to determine the source node for an edge insertion. Random number 17.68 in $[0, 20.8)$ guides Algorithm 13 through the tree.

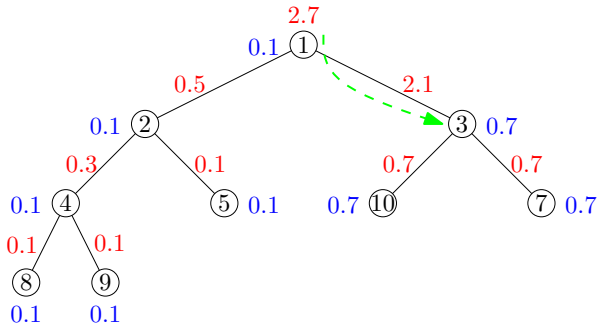


Figure 4.2.9. The target tree $\bar{T}_t(12)$ of node 12 (see Figure 4.2.7). Given the decision to insert an edge starting at node 12, $\bar{T}_t(12)$ is used to determine the target node for the edge. Random number 1.08 in $[0, 2.7)$ guides Algorithm 13 through the tree.

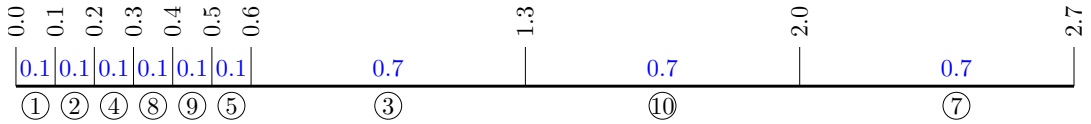


Figure 4.2.10. The target tree in Figure 4.2.9 can equivalently be interpreted as an interval of length 2.7 (total weight at root node), subdivided by the nodes' weights, listed *in-order*. The random choice now simply picks the node associated to the interval containing the random number from $[0, 2.7)$.

and (iii) choose target node, always use correctly normalized and/or combined conditional probabilities, to remain consistent with the above model. The reason for this multi-step procedure is simply an easier-to-handle representation of the different pieces of data. We formulate this observation as a small lemma.

proportional probabilities

Lemma 4.2.1 (Probabilities) *Weighted randomization using binary trees yields probabilities p(insert edge between u and v) = proportional to $p(u, v)$ (or to $1 - p(u, v)$ for deletion) if u and v are non-adjacent (adjacent), and 0 otherwise.*

Proof. We will show that each of the three steps supports this proportionality. We use insertions; deletions are analogous. As a first observation, note that a binary tree for the selection of an element out of a given weighted set as above yields proportional probabilities: The binary search through a tree is equivalent to dividing an array of length equal to the total weight $w + l + r$ of the tree's root into intervals associated to the nodes of the tree with length equal to the nodes' inner weight w, as listed by an *in-order traversal* of the tree, and then picking the interval that contains a random number between 0 and the total root weight.

Let $\xi_{u,v}$ be the *event* that inserts an edge from u to v, furthermore, let ξ_u be the *event* that an edge using u as the source node is inserted, and let ξ_{insert} mean that an edge is inserted. Suppose now u and v are already adjacent, then the target tree $\bar{T}_t(u)$ does not contain the

node v, and thus $p(\xi_{u,v}) = 0$. Otherwise, since trees preserve proportionality:

$$p(\xi_{u,v} \mid \xi_u) = \frac{p(u,v)}{\sum\limits_{\substack{w \in V \\ w \nsim u}} p(u,w)} \tag{4.2.14}$$

$$p(\xi_u \mid \xi_{\text{insert}}) = \frac{\sum\limits_{\substack{w \in V \\ w \nsim u}} p(u,w)}{\sum\limits_{x \in V} \sum\limits_{\substack{w \in V \\ w \nsim x}} p(w,x)} \tag{4.2.15}$$

$$p(\xi_{\text{insert}}) = \frac{\sum\limits_{x \in V} \sum\limits_{\substack{w \in V \\ w \nsim x}} p(w,x)}{\underbrace{\sum\limits_{x \in V} \sum\limits_{\substack{w \in V \\ w \nsim x}} p(w,x)}_{\text{all possible edge insertions}} + \underbrace{\sum\limits_{x \in V} \sum\limits_{\substack{w \in V \\ w \sim x}} (1 - p(w,x))}_{\text{all possible edge deletions}}} \tag{4.2.16}$$

Equation (4.2.16) is not based on the arguments about trees but derives directly from Algorithm 18. Combining Equations (4.2.14)-(4.2.16) we obtain the lemma. □

4.2.5 A Ready-to-Use Java Implementation

a user guide

In the following we detail—in the style of a user guide—how our generator can be downloaded, called, and its output read.

4.2.5.1 Calling the Generator

The generator is launched as a command line tool. It generates a graph and writes it to files as described in the following subsection. For details on the exact nature and effect of parameters, please refer to Section 4.2.4, as we keep descriptions short here for quick reference. The Java main class of the generator is `DCRGenerator`. The `-g` option lets us enter the parameters for a single graph. In order to generate multiple graphs at once, the `-f` option can be used together with a file where each line specifies the parameters of a new graph. We can call for help with the `-h` option. We now explain the syntax of a command line call, the parameters are listed in Table 4.2.1. The syntax specified in Extended Backus Naur Form is as follows:

DCRGenerator

```
argument ::= "-h" | "-g" { keyval } | "-f" file
keyval ::= ikey "=" ival | dkey "=" dval  | hkey "=" hval | fkeyval
ikey ::= "n" | "k" | "t_max" | "eta"
dkey ::= "p_in" | "p_out" | "p_nu" | "p_chi" | "p_omega" | "p_mu" | "theta" | "beta"
bkey ::= "enp" | "binary" | "graphml"
hkey ::= "p_inList" | "D_s"
hval ::= dval | list
list ::= "[" dval { "," dval } "]"
fkeyval = "outDir=" dir | "fileName=" fname
```

A syntactically correct value for `ival` is any string that can be parsed by the `java.lang.Integer.parseInt` method. For `dval`, it is any string that can be parsed by `java.lang.Double.parseDouble`. `file` may be any string from which a java.io.FileReader can be constructed.

Using only `p_in`, a global p_{in} for all clusters is used. The nodes are then distributed over the clusters using the method of biased distribution described in Section 4.2.4.2. Using `p_inList` overrides `p_in` and sets p_{in} individually for each cluster. The length of this list has

CLI key	notatation	domain	default	explanation
`n`	n_0	$\mathbb{N}$	60	initial number of nodes in G_0
`p_in`	p_{in}	$[0,1]$	0.02	edge prob. for node pairs in same cluster
`p_out`	p_{out}	$[0,1]$	0.01	edge prob. for node pairs in different clusters
`k`	k	$\mathbb{N}$	2	initial number of clusters
`t_max`	$t_{\max}$	$\mathbb{N}$	100	total number of *time steps*
`p_nu`	p_ν	$[0,1]$	0.5	given a *node event*, prob. that a node will be added ($1-p_\nu$ for a node deletion)
`p_chi`	p_χ	$[0,1]$	0.5	prob. of an *edge event* ($1-p_\chi$ for a *node event*)
`p_omega`	p_ω	$[0,1]$	0.02	prob. of a *cluster event*
`p_mu`	p_μ	$[0,1]$	0.5	given a *cluster event*, prob. of a *merge event* ($1-p_\mu$ for a *split event*)
`theta`	θ	$[0,1]$	0.25	tolerance threshold to accept new clustering
`beta`	β	$\mathbb{R}$	1.0	exponent of biased selection method
`eta`	η	$\mathbb{N}$	1	lower bound on *edge events* per *time step*
`p_inList`	$[p_{\text{in}}(C_1),\ldots,p_{\text{in}}(C_k)]$	$[0,1]^k$	(not used)	list of individual values of p_{in} for clusters, can be used instead of p_{in}
`D_s`	$[s_1,s_2,\ldots,s_k]$	$\mathbb{R}_+^k$	(not used)	relative size dist. of of cluster sizes in $\mathcal{C}(G_0)$, can be used instead of β
`enp`	gauss. est.	{`true`, `false`}	`false`	new p_{in} gauss. estimate (`true`) or arithm. mean
`outDir`		String	./	file output directory
`fileName`		String	[7]	name of output file
`binary`		{`true`, `false`}	`false`	`true` enables output as binary file (extension .graphj)
`graphml`		{`true`, `false`}	`false`	`true` enables output as GraphML file (extension .graphml)

Table 4.2.1. Command line input parameters

to be equal to the number of initial clusters. Likewise, using `beta` manages the cluster sizes, but stating a value for `D_s` overrides `beta` with an explicit list (again of the same length) of numbers. In this case, weighted selection (Section 4.2.4.5) is used to distribute the nodes. Any required parameter not specified by the user will be set to a default value, which are listed in Table 4.2.1. Thus, calling the generator by

```
> java DCRGenerator -g
```

will produce a graph with only the default values. An example call of the generator could look like this: *example call*

```
> java -jar DCRGenerator -g t_max=1000 n=100 k=5 p_in=0.3 p_out=0.02
  eta=10 p_omega=0.05 binary=true graphml=false
  outDir=/myDynamicGraphsDirectory fileName=mySampleDynamicGraph
```

[7] The default is composed from the current date as `dcrGraph_yyyy-MM-dd-HH-mm-ss`

4.2.5.2 Output Formats

The generator supports two output formats, one of which is XML-based and should have become the main output format of the generator. However, with the *visone* project it is based on slowing down recently, we now recommend the binary format. Nonetheless we describe both in the following.

output: GraphML

Dynamic `GraphML`. For historical reasons our first output format is the XML-based `GraphML`, which can be read, e.g., by a future release of *visone*, tools from the *visone*-library or by a homemade XML-parser.[8] For an introduction to the format we refer the reader to the GraphML Primer[9]. `GraphML` allows for the definition of additional data attributes for nodes and edges which are addressed via a key. These attributes can be static or dynamic, such that dynamic information for our generated graphs is provided by *visone*-specific data tags. At this time a general reference for the dynamic add-ons of visone is [35], a full description of its dynamic add-ons to `GraphML`, however, still does not exist. Therefore we here provide a preliminary technical description of the necessary extensions.

Code Sample 1 shows the definitions used by the generator. The static attribute `dcrGenerator.ID` is the unique node identifier assigned by the generator. A dynamic attribute `visone.EXISTENCE` denotes whether the node or edge is included in the graph at a *time step*. The dynamic attributes `dcrGenerator.CLUSTER` and `dcrGenerator.REFERENCECLUSTER` contain the ids of the cluster and reference cluster assigned to a node by the generator. These are also mapped onto distinct colors for visualization, namely on `visone.BORDERCOLOR` and `visone.COLOR` respectively. Code Sample 2 is an example for the representation of a node - the node exists from *time step* 0 to *step* 55, remaining in cluster 1 and reference cluster 1.

Code Sample 1 Custom `GraphML` attributes used in the generator output

```
<key attr.name="visone.EXISTENCE" attr.type="boolean" dynamic="true" for="node" id="d7"/>
<key attr.name="visone.EXISTENCE" attr.type="boolean" dynamic="true" for="edge" id="d15"/>
<key attr.name="visone.COLOR" attr.type="string" dynamic="true" for="node" id="d4"/>
<key attr.name="visone.BORDERCOLOR" attr.type="string" dynamic="true" for="node" id="d5"/>
<key attr.name="dcrGenerator.CLUSTER" attr.type="int" dynamic="true" for="node" id="d100"/>
<key attr.name="dcrGenerator.REFERENCECLUSTER" attr.type="int" dynamic="true"
for="node" id="d101"/>
<key attr.name="dcrGenerator.ID" attr.type="int" dynamic="false" for="node" id="d102"/>
```

Code Sample 2 `GraphML` representation of a dynamic node

```
<node id="n10">
        <data key="d102">10</data>
        <data key="d7">false</data>
        <data key="d100" time="0">1</data>
        <data key="d5" time="0">#6376b3</data>
        <data key="d101" time="0">1</data>
        <data key="d4" time="0">#6376b3</data>
        <data key="d7" time="0">true</data>
        <data key="d7" time="56">false</data>
</node>
```

output: binary

A Binary Format. In addition to the `GraphML` format, this version provides a custom binary file format which occupies much less memory. A file can be parsed by loading the file into a `java.io.DataInputStream`. After two integers containing the length of the arrays, a byte array for operation codes and an integer array for arguments follow. The dynamic graph and the two associated clusterings can be reconstructed by iterating through the operation codes from the first array and reading the corresponding number of integer arguments from the second array. Node Id's are assigned implicitly through the order in which the nodes are

[8] Since we happily used visone, we do not yet provide a convenient reader for dynamic GraphML.

[9] `http://graphml.graphdrawing.org/primer/graphml-primer.html`

created. Table 4.2.2 shows the semantics of operation codes and arguments and Figure 4.2.11 illustrates the arrangement of data in the file. Code Section 3 is a sample of Java code for reading this, see Section 4.2.5.3 for where to download this code. It reads the dynamic clustered graph into an `ArrayList` of operations as listed in Table 4.2.2

binary format and a reader

Code Sample 3 Example code for parsing the binary `.graphj` file format

```
File file = new File(filePath);
FileInputStream fStream = new FileInputStream(file);
DataInputStream dStream = new DataInputStream(fStream);

int opLength = dStream.readInt();
int argLength = dStream.readInt();

ArrayList<Byte> ops = new ArrayList<Byte>();
ArrayList<Integer> args = new ArrayList<Integer>();

for (int i = 0; i < opLength; ++i) {
        ops.add(dStream.readByte());
}

for (int i = 0; i < argLength; ++i) {
        args.add(dStream.readInt());
}
```

4.2.5.3 Download

Our dynamic generator for dynamic clustered random graphs can freely be downloaded and used. The site that hosts a downloadable jar-file is maintained is `http://i11www.iti.uni-karlsruhe.de/projects/spp1307/dyngen`. Additional information and updates will also be posted there, in particular, this includes any news on an upcoming implementation as a module in an official release of visone.

download

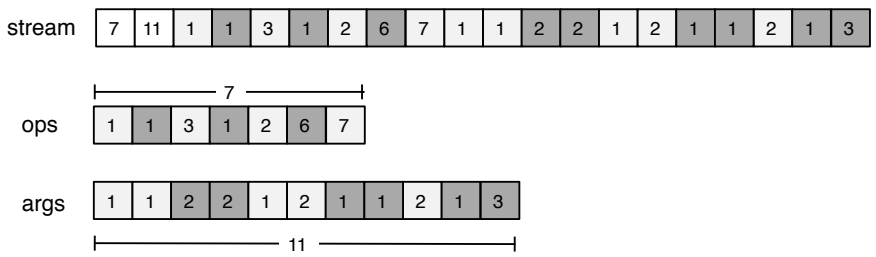


Figure 4.2.11. Arrangement of the data stored in the binary output of the generator.

operation	op-code	arg0	arg1
create node	1	$id(C)$	$id(C_{\text{ref}})$
delete node u	2	$id(u)$	-
create edge $\{u, v\}$	3	$id(u)$	$id(v)$
remove edge $\{u, v\}$	4	$id(u)$	$id(v)$
set cluster of u	5	$id(u)$	$id(C)$
set reference cluster of u	6	$id(u)$	$id(C_{\text{ref}})$
next *time step*	7	-	-

Table 4.2.2. Binary file format

4.2.6 Pseudocode

In this section we list a collection of procedures described in previous sections as pseudocode. We generally assume that the function rand returns a real number drawn uniformly at random from the interval $[0, 1)$, as, e.g., implemented by the function `java.lang.Math.random()` in Java. Moreover we assume that binary trees are stored in an array in the usual way, i.e., such that the left and the right children of a node at index i are stored at index $2i$ and $2i + 1$ respectively.

Algorithm 13: weightedTreeSelect

Input: weighted binary tree T (elements $q_i = (e_i, w_i, l_i, r_i)$, root q_0)
Output: random element q_r with probability that q_r is picked $\sim w_r$

```
1  x ← rand() · (w_0 + l_0 + r_0)
2  i ← 0
3  while true do
4      switch x do                                   // branch at current node
5          case x ≤ w_i
6              return e_i                  // terminate and return current node
7          case w_i < x ≤ (w_i + l_i)
8              x ← x − w_i
9              i ← 2i                          // branch to index of left child
10         case w_i + l_i < x
11             x ← x − w_i − l_i
12             i ← 2i + 1                     // branch to index of right child
```

tree selection

Algorithm 13 describes how a node of a tree used for randomized selection is chosen in logarithmic time in the size of the tree, which is a complete binary tree. Since a change to the dynamic graph is performed after each such choice, we require procedures that keep a tree consistent after nodes are deleted or added. We only give pseudocode for the case of deleting a node from a tree in Algorithm 14; the case for the addition of a tree node is even simpler and omitted. Note that the weight structure of a tree is updated in logarithmic time per tree by the call to Algorithm 15. Four trees in total are affected per edge modification, and all trees need updates if a node is added to or deleted from the graph.

node deletion

tree updates

Algorithm 14: weightedTreeDelete

Input: weighted binary tree T (elements $q_i = (e_i, w_i, l_i, r_i)$, root q_0), $i_{\text{del}} \in \mathbb{N}$
Output: updated tree T with i_{del}th element deleted

```
1  if i_del = i_max then
2      updateWeight(T, i_del, 0)
3      remove q_{i_del}
4  else
5      q_tmp ← q_{i_max}
6      weightedTreeDelete(T, i_max)
7      q_{i_del} ← q_{i_tmp}
8      updateWeight(T, q_{i_del}, w_{i_del})
```

maintains bounds

Algorithm 14, which performs the deletion of elements, retains the tree's properties of being binary and complete; thus, the logarithmic time bounds for searching through the changing tree are maintained. In fact, we observed that for the two source trees a simpler method was consistently quicker in practice: on a deletion, simply set the node's weight to 0.

This method only virtually keeps the tree complete, but saves the effort of restructuring at the cost of gradually letting it grow larger. In the following we list the rough pseudocode of the whole dynamic graph generator and its helper functions. *lazy practice*

Algorithm 15: updateWeight

Input: weighted binary tree T (elements $q_i = (e_i, w_i, l_i, r_i)$, root q_0), $i \in \mathbb{N}$, $w \in \mathbb{R}$
Output: propagates new weight from e_i to e_0, to make T consistent

$w_i \leftarrow w$
while $i > 0$ **do**
  $i_{\text{parent}} \leftarrow \lfloor i/2 \rfloor$ `// compute the index of the parent node`
  if $i \equiv 0 \mod 2$ **then** `// in this case` q_i `is its parent's left child`
    $l_{i_{\text{parent}}} \leftarrow w_i + l_i + r_i$
  else `// otherwise` q_i `is its parent's left child`
    $r_{i_{\text{parent}}} \leftarrow w_i + l_i + r_i$
  $i \leftarrow i_{\text{parent}}$

Algorithm 16: initial DCR graph

Input: $n \in \mathbb{N}$, $k \in \mathbb{N}$, $\beta \in \mathbb{R}$ or $[s_1, s_2, \ldots, s_k] \in \mathbb{R}^k$, $p_{\text{out}} \in [0,1]$, $p_{\text{in}} \in [0,1]$ or $[p_{\text{in}}(C_1), \ldots, p_{\text{in}}(C_k)] \in [0,1]^k$
Output: initial state G of a dynamic graph

$G = (V, E) \leftarrow (\{\}, \{\})$
for $i \leftarrow 0$ **to** n **do**
  $V \leftarrow V +$ new node u
$\zeta = \{C_1, \ldots, C_k\} \leftarrow \{\{\}, \ldots, \{\}\}$
for v **in** V **do**
  $C_i \leftarrow \mathsf{biasedSelect}(\zeta, \beta)$ or $C_i \leftarrow \mathsf{weightedSelect}(\zeta, [s_1, s_2, \ldots, s_k])$
  $C_i \leftarrow C_i + v$
for $\{u, v\}$ **in** $\binom{V}{2}$ **do**
  if *rand()* $\leq p(u, v)$ **then**
    $E \leftarrow E + \{u, v\}$

Please note that for reasons of readability we do not delve into catching pathological cases such as setting $p_{\text{in}} = p_{\text{out}} = 0$. We omit the domains and meaningful names of the input parameters in the following and refer the reader to Table 4.2.1 for more information; however, we naturally stick to the variables used throughout this section.

Algorithm 17: binaryRangeSearch

Input: $x \in \mathbb{R}$, $a \in \mathbb{R}^n$, $h \in \mathbb{N}$, $l \in \mathbb{N}$

```
1  if l > h then
2      return −1  // element not found
3  m ← ⌊(l+h)/2⌋
4  if a[m] ≥ x then
5      if m = 0 or a[m − 1] < x then
6          return m
7      else
8          return binaryRangeSearch(x, a, l, m − 1)
9  else
10     if m = n − 1 or a[m + 1] ≥ x then
11         return m + 1
12     else
13         return binaryRangeSearch(x, a, m + 1, h)
```

Algorithm 18: weightedSelection(A, ω) (ws)

Input: A: set of elements, $\omega : A \to \mathbb{R}^+$: weight function

Output: e: selected element

```
1  a[i] ← e_i ∈ A
2  b[i] ← Σ_{j=0}^{i} ω(a[j])
3  x ← rand() · Σ_j ω(a[j])
4  i ← binaryRangeSearch(x, b, 0, |b|)
5  return a[i]
```

Algorithm 19: DCRGenerator (Standalone)

Input: n, k, β or $[s_1, s_2, \ldots, s_k]$, p_{out}, p_{in} or $[p_{\text{in}}(C_1), \ldots, p_{\text{in}}(C_k)]$, $t_{\max}$, σ, p_ν, p_χ, p_μ, p_ω, θ, gauss. est.

Output: dynamic graph $G(t)$

$G_0 = (V, E) \leftarrow$ initialDCRGraph()
for $t \leftarrow 1$ **to** t_{max} **do**
  for *event* $A \to B$ **in** *ongoing events* **do**
    if *completed*$(A \to B)$ **then** update$(\zeta_{\text{ref}}, A \to B)$
  if *rand()* $\leq p_\omega$ **then**
    if *rand()* $\leq p_\mu$ **then**
      if *2 clusters available* **then**
        $\{C_i, C_j\} \leftarrow$ randomPair(ζ)
        $C_{k+1} \leftarrow C_i \cup C_j$
        $\zeta \leftarrow \zeta \setminus \{C_i, C_j\} \cup C_{k+1}$
        if *use gaussian estimate* **then** $p_{\text{in}}(C_{k+1}) \leftarrow$ gauss() **else** $p_{\text{in}}(C_{k+1}) \leftarrow \frac{p_{\text{in}}(C_i)+p_{\text{in}}(C_j)}{2}$
      else
        if *cluster available* **then**
          $C_i \leftarrow$ randomElement(ζ)
          $C_{k+1} \cup C_{k+2} \leftarrow C_i$
          $\zeta \leftarrow \zeta \setminus \{C_i\} \cup \{C_{k+1}, C_{k+2}\}$
          if *use gaussian estimate* **then** $p_{\text{in}}(C_{k+1}) \leftarrow$ gauss() **else** $p_{\text{in}}(C_{k+1}) \leftarrow p_{\text{in}}(C_i)$
    $i \leftarrow 0$
    while $i < \eta$ **do**
      if *rand*() $\geq p_\chi$ **then**
        if *rand*() $\leq p_\nu$ **then**
          $V \leftarrow V +$ new node u
          $C_i \leftarrow$ randomElement(ζ); $C_i \leftarrow C_i \cup \{u\}$
          for $\{u,v\}$ **in** $\{\{u,v\} : v \in V \setminus \{u\}\}$ **do**
            if *rand*() $\leq p(u, v)$ **then** $E \leftarrow E + \{u, v\}$; $i \leftarrow i + 1$
        else
          $u \leftarrow$ randomElement(V)
          $C(u) \leftarrow C(u) - u$; $V \leftarrow V - u$
          $i \leftarrow i + \deg(v)$; $E \leftarrow E \setminus \{\{u, v\} : v \in V\}$
      else
        if $n \geq 2$ **and not** *(G_t consists of disjoint cliques* **and** *ongoing events* $= \emptyset$*)* **then**
          $op \leftarrow$ weightedSelect$(\{\text{create}, \text{remove}\}, \{P_E, P_{\bar{E}}\})$
          if *op is remove* **then**
            $s \leftarrow$ weightedTreeSelect(T_s), $t \leftarrow$ weightedTreeSelect$(T_t(s))$
            $E \leftarrow E - \{s, t\}$; $i \leftarrow i + 1$
          else
            $s \leftarrow$ weightedTreeSelect$(\bar{T}_s)$, $t \leftarrow$ weightedTreeSelect$(\bar{T}_t(s))$
            $E \leftarrow E + \{s, t\}$; $i \leftarrow i + 1$
      update all trees involved in the changes
return $G(t)$

Section 4.3

Modularity-Driven Clustering of Dynamic Graphs

. . . and I heard their leader's hair was two meters long!

(Ignaz Rutter, speculating about what rumours will be told about our footbag playing in ten years)

MAXIMIZING THE QUALITY INDEX MODULARITY has become one of the primary methods for identifying the clustering structure within a graph in practice, however, in a dynamic context it has not yet been touched. In this section we thus return our focus to the quality function *modularity*, which we thoroughly discussed in Sections 2.2 and 2.3. Recalling what we said in these above sections, *modularity* is the most prominent example of a clustering technique which is heavily used nowadays but almost exclusively maximized in diverse heuristic ways. The general fact that it suffers from local optima adds to the issue that it can behave in a non-local manner (see Section 2.2), such that in a dynamic scenario a clustering can expect rather volatile behavior and even oscillations—a conjecture we shall disprove. Still, in practice it is certainly a reasonable approach to rely on *modularity*-driven clusterings for some changing network, alongside the general postulations for dynamic clusterings as listed in Section 4.1. For this task no work has been conducted so far.

In the following we investigate procedures $\mathcal{A}$ that find a good modularity-based clustering $\mathcal{C}'(G')$ without re-clustering from scratch, but building upon $\mathcal{C}(G)$. We present, analyze and evaluate a number of concepts for efficiently updating *modularity*-driven clusterings. We prove the NP-hardness of dynamic *modularity* optimization and develop heuristic dynamizations of the most widespread [57] and the fastest [38] static algorithms, alongside apt strategies to determine the search space. On a theoretical side, for our fastest procedure, we can even prove a tight bound of $\Theta(\log n)$ on the expected number of operations required. We then evaluate these and a heuristic dynamization of an ILP[10]-algorithm, see Section 2.4. We compare the algorithms with their static counterparts and evaluate them experimentally on random preclustered dynamic graphs and on a large real-world instance. Our results are very favorable for the dynamic approach. They expose that the dynamic maintenance of a clustering yields higher quality than recomputation, guarantees much *smoother* clustering dynamics and much lower runtimes. Additionally they yield strong evidence that small search spaces around the epicenter of the graph *change* work best, and that actual local optimization (via an ILP) around this epicenter is not the best choice.

Frankly speaking, this Section is not only the one I deem most valuable for future research on practicable dynamic graph clustering, but it also contains those results that had to wait so long for the more thorough understanding of *modularity* we eventually attained, and for evidence of its practicality. On top of that, this section motivated my work on a generator for

[10]ILP stands for Integer Linear Program

fully dynamic clustered random graphs in Section 4.2, which bred a ready-to-use platform. Together with the fact that the results the evaluations of the approaches in this section yield, so very clearly advocate a dynamic approach—in terms of speed, *smoothness* and even quality—I felt that after finishing this work, the time is ripe for writing up my thesis. Without the competent assistance of a smart student of mine, Christian Staudt, this study certainly would not be in its current good shape. None of the content herein has yet been published.

Main Results

- The problem of updating a *modularity*-optimal clustering after a *change* in the graph is NP-hard. (Section 4.3.2 and Corollary 4.3.1)
- We develop dynamizations of the currently fastest and of the most widespread heuristics for *modularity*-maximization. For our fastest procedure we prove a tight bound on its asymptotic runtime. (Section 4.3.3 and Theorem 4.3.1)
- We conduct an experimental evaluation of these algorithms, of their static counterparts and of a dynamic partial ILP for local optimality. (Section 4.3.4)
- Our algorithms for dynamic updating (i) save runtime, (ii) yield higher *modularity* and (iii) much *smoother* clustering dynamics than their static versions; the second point is a particularly strong result, as the contrary might be expected. (Section 4.3.4)
- Heuristics perform better than the approach of being locally optimal at this task. (Section 4.3.4.2)
- For update heuristics, surprisingly small search spaces work best, avoid local optima well and adapt quickly and aptly to changes in the *ground-truth* clustering, which strongly argues for the assumption that *changes* in the graph ask for local updates on the clustering. (Section 4.3.4.3)

Future Work. A large variety of data formats exist already for static graphs. Although tools for conversion are ubiquitous (or quickly conceived), things will get worse for dynamic graphs. For this reason it will be a hard job to provide an easily usable tool for dynamic graph clustering. Conversely, with the results of our evaluation at hand such a tool is quite immediately the next step.

4.3.1 Preliminaries

4.3.1.1 Measuring the Smoothness of a Dynamic Clustering

For one of our prime criteria for a good dynamic clustering, its *smoothness*, we can now build upon what we learned earlier, in the context of comparing two static graph clusterings in Section 2.6. By comparing consecutive clusterings, we can quantify how *smooth* an algorithm manages the transition between two steps, an aspect which is crucial to both readability and applicability. As discussed in Section 2.6, an array of measures exist that quantify the (dis)similarity between two partitions of a set. However, our results strongly suggest that most of these widely accepted measures are qualitatively equivalent in all our (non-pathological) instances. An example plot indicating this fact is given in Figure 4.3.2. This observation has already been made in Section 2.6; moreover since the array of objections we there pose towards a number of measures uniformly relies on pathological instances or extremal tendencies of a measure, we omit a quantification of the similarity of these measures in terms of, e.g., a correlation analysis of a series of reasonable dynamic clustered graphs. While this would certainly be an interesting topic on its own right, for our purpose we are happy with our solid observations on this. We thus restrict our view to the *(graph-structural) Rand* index [70], being a well known representative; it maps two clusterings into the interval $[0, 1]$, i.e., from

smoothness

equality to maximum dissimilarity (as a distance): $\mathcal{R}_g(\mathcal{C},\mathcal{C}') := 1 - (|E_{11}| + |E_{00}|)/m$, with $E_{11} = \{\{v,w\} \in E : \mathcal{C}(v) = \mathcal{C}(w) \wedge \mathcal{C}'(v) = \mathcal{C}'(w)\}\}$, and E_{00} the analog for inequality. *Low* distance values correspond to a *smooth* dynamic clustering.

When we compare two clustering $\mathcal{C}(G), \mathcal{C}'(G')$ of different graphs $G = (V,E) \neq G' = (V',E')$, the above measures are not well-defined. A canonical solution is to use the *intersection* of the two graphs, i.e., define $G'' = (V'',E'') = (V \cap V', E \cap E')$, and compare $\mathcal{C}_{|V''}(G'')$ and $\mathcal{C}'_{|V''}(G'')$. In fact any other workaround seems unfair: The intuitions of measures based on either pair-counting, set overlaps or on entropy all do not conform to classifying elements unknown in either G or G' in any particular way—be it well-classified or ill-classified. Simply ignoring "new" elements avoids introducing a bias due to particular dynamics in a graphs such as growth or sparsification.

4.3.1.2 The Quality Index *Modularity*

We here return our focus to *modularity*. For background information and general insights into the nature of this quality index for graph clusterings, the reader should refer to Chapter 2; in the following we just repeat the crucial pieces. *Modularity* can be formulated as:

$$\mathrm{mod}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} - \frac{1}{4m^2} \sum_{C \in \mathcal{C}} \left(\sum_{v \in C} \deg(v) \right)^2 \quad \text{(weighted vers. analogous)} \tag{4.3.1}$$

Recall that, roughly speaking, *modularity* measures the fraction of edges which are covered by a clustering and compares this value to its expected value, given a random rewiring of the edges which, on average, respects node degrees. See Section 2.3 for further details.

4.3.2 The Hardness of DynModOpt

ModOpt, the problem of optimizing *modularity* is NP-hard (see Theorem 2.2.1)[11], but *modularity* can be computed in linear time and lends itself to a number of simple greedy maximization strategies. We now state the hardness of updating an optimal clustering after a graph *change*:

DynModOpt *is NP-hard*

Corollary 4.3.1 (DynModOpt is NP-hard) *Given graph G, a modularity-optimal clustering $\mathcal{C}^{\mathrm{opt}}(G)$ and an atomic event Δ to G, yielding G'. It is NP-hard to find a modularity-optimal clustering $\mathcal{C}^{\mathrm{opt}}(G')$.*

Proof. We reduce an instance G of ModOpt to a linear number of instances of DynModOpt. Given graph G, there is a sequence $\mathcal{G}$ of graphs $(G_0, \ldots, G_\ell = G)$ of linear length such that (i) $\mathcal{G}$ starts with G_0 consisting of one edge e of G and its incident nodes u, v, (ii) $\mathcal{G}$ ends with G, (iii) graph G_{i+1} results from G_i and an atomic event Δ_i. ModOpt can be solved in constant time for G_0 yielding $\mathcal{C}^{\mathrm{opt}}(G_0)$. Subsequently solving DynModOpt for instances $G_i, \mathcal{C}^{\mathrm{opt}}(G_i), \Delta_i$ yielding $\mathcal{C}^{\mathrm{opt}}(G_{i+1})$, we end with $\mathcal{C}^{\mathrm{opt}}(G_\ell) = \mathcal{C}^{\mathrm{opt}}(G)$, the solution to ModOpt. □

Corollary 4.3.1 leaves little hope for solving the update problem efficiently if we set the goal to be *modularity optimization* and require algorithm $\mathcal{A}$ to conform, even if we do take the effort to compute an optimal initial clustering—e.g., via an ILP. Since, furthermore, the static problem is NP-hard and no approximations are known, we resort to heuristic updates.

4.3.3 Dynamic Clustering-Algorithms

Remember from Section 4.1 how we defined graph *changes*: A graph *change* Δ can comprise any number b of atomic events (see Table 4.1.1); the deletion of a node alongside its incident edges is an example of such a compound event. In the view of a dynamic clustering algorithm this is a *batch update*, delimited by a *time step event*, which indicates to an algorithm that a

batch updates

[11] Since this is an optimization problem, we proved the NP-hardness of the corresponding decision problem and the actual problem is NPO-hard. For simplicity we omit this distinction.

readily updated clustering must now be supplied. Between *time steps* it is up to the algorithm how it maintains its intermediate clustering, no measuring takes place then. Please review Figure 4.1.3 for a quick recap.

locality assumption

A natural approach to dynamizing an agglomerative clustering algorithm is to break up those local parts of its previous clustering, which are most likely to require a reassessment after some *changes* to the graph. The half-finished instance is then given to the agglomerative algorithm for completion. A crucial ingredient thus is a *prep strategy* S which decides on the search space which is to be reassessed. We will discuss such strategies later, until then we simply assume that S breaks up a reasonable part of $\mathcal{C}(G_{t-1})$, yielding $\tilde{\mathcal{C}}(G_{t-1})$ (or $\tilde{\mathcal{C}}(G_t)$ if including the *changes* in the graph itself). We call $\tilde{\mathcal{C}}$ the *preclustering* and nodes that are chosen for individual reassessment *free* (can be viewed as singletons).

prep strategy

preclustering

free node

4.3.3.1 Algorithms for Dynamic Updates of Clusterings

The Global Greedy Algorithm. The most prominent algorithm for *modularity* maximization is a global greedy algorithm [57] (see also Sections 2.2 and 2.3), which we call Global (Algorithm 20). Starting with singletons, for each pair of clusters, it determines the increase in *modularity* dQ that can be achieved by merging the pair and performs the most beneficial merge. This is repeated until no more improvement is possible. As the pseudo-dynamic algorithm sGlobal[12], we let this algorithm cluster from scratch ($\mathcal{C}^V$) at each timestep, as a comparison to the dynamic approaches. By passing a *preclustering* $\tilde{\mathcal{C}}(G_t)$ to Global we can define the proper dynamic algorithm dGlobal. Starting from $\tilde{\mathcal{C}}(G_t)$ this algorithm lets Global perform greedy agglomerations of clusters.

Algorithm 20: Global$(G, \mathcal{C})$

```
while ∃C_i, C_j ∈ 𝒞 : dQ(C_i, C_j) ≥ 0 do
    (C_1, C_2) ← arg max_{C_i,C_j∈𝒞} dQ(C_i, C_j)
    merge(C_1, C_2)
```

global greedy

dynamic: build upon $\tilde{\mathcal{C}}$

The Local Greedy Algorithm. In a recent work [38] the simple mechanism of the aforementioned Global has been modified as to rely on local decisions (in terms of graph locality), yielding an extremely fast and efficient maximization. We compared Orca to this method in Section 2.5. Instead of looking globally for the best merge of two clusters, Local repeatedly lets each node consider moving to one of its neighbors' clusters, if this improves *modularity*; this potentially merges clusters, especially when starting with singletons. As soon as no more nodes move, the current clustering is *contracted*, i.e., each cluster is contracted to a single node, and adjacencies and edge weights between them summarized. Then, the process is repeated on the resulting graph which constitutes a higher level of abstraction; in the end, the highest level clustering is decisive about the returned clustering: The operation unfurl assigns each elementary node to a cluster represented by the highest level cluster it is contained in.

local greedy

Algorithm 21: Local$(G^{0\dots h_{\max}}, \mathcal{C}^{0\dots h_{\max}}, P)$

```
1  h ← 0
2  repeat
3      (G, 𝒞) ← (G^h, 𝒞^h)
4      repeat
5          forall free v ∈ V do
6              if max_{v∈N(u)} dQ(u, v) ≥ 0 then
7                  w ← arg max_{v∈N(u)} dQ(u, v)
8                  move(u, 𝒞(w))
9      until no more changes
10     𝒞^h ← 𝒞
11     (G^{h+1}, C̃^{h+1}) ← contract(G^h, 𝒞^h, P)
12     h ← h + 1
13 until no more real contractions
14 𝒞(G^0) ← unfurl(𝒞^{h−1})
```

We again sketch out an algorithm which serves as the core for both a static and a dynamic variant of this approach, as shown in Algorithm 21. As the input, this algorithm takes a hierarchy of graphs and clusterings and a policy P which is decisive about the algorithm's search space. In fact, P

[12] For historical reasons, sGlobal appears in plots as StaticNewman, dGlobal as Newman, sLocal as StaticBlondel and dLocal as Blondel, based on some of the algorithms' authors.

has its part in the graph *contractions*, in that P decides which nodes of the next level graph should be free to move. Note that the input hierarchy can also be flat, i.e., $h_{\max} = 0$, in that case line 11 simply creates all necessary higher levels.

static variant

Again posing as a pseudo-dynamic algorithm, the static variant (as in [38]), sLocal, passes only $(G_t, \tilde{\mathcal{C}}^V)$ to Local, such that it starts with singletons and all nodes freed, instead of a proper *preclustering*. The policy P is set to tell the algorithm to also start from scratch on all higher levels and to not work on previous results in line 11, i.e., in $\tilde{\mathcal{C}}^{h+1}$ again all nodes in the contracted graph are free singletons.

dynamic variant

The dynamic variant dLocal remembers its old results. It passes the changed graph, a current *preclustering* of it and all higher-level contracted structures from its previous run to Local: $(G_t, G_{\text{old}}^{1,\dots,h_{\max}}, \tilde{\mathcal{C}}, \mathcal{C}_{\text{old}}^{1,\dots,h_{\max}}, P)$. In level 0, the preclustering $\tilde{\mathcal{C}}$ defines the set of free nodes. In levels beyond 0, policy P is set to have the contract-procedure free only those nodes of the next level, that have been affected by lower level changes (or their neighbors as well, tunable by policy P). Roughly speaking, dLocal starts by letting all free (elementary) nodes reconsider their cluster. Then it lets all those (super-)nodes on higher levels reconsider their cluster, whose content has changed due to lower level revisions. Thus, a run of Algorithm 21 restores a low-stress state which a run of the static algorithm could have produced, but avoids recomputations in unrelated regions of the graph. In particular there is no risk that ambiguous or near-tie situations are resolved in a complementary fashion without necessity.

dynamic: build upon previous levels

node-distance variables

ILP. While optimality is out of reach, the problem *can* be cast as an ILP; for convenience we repeat some of what has been said in Section 2.2 and 2.4 and build upon it. A *node-distance* relation (or pseudometric) between a set $\tilde{V}$ of nodes (think $\tilde{V} = V$ for now) indicates whether nodes are in the same cluster:

$$\mathcal{X}(\tilde{V}) := \{X_{uv} : \{u,v\} \in \binom{\tilde{V}}{2}\} \quad \text{with} \quad X_{uv} = \begin{cases} 0 & \text{if } \mathcal{C}(u) = \mathcal{C}(v) \\ 1 & \text{otherwise} \end{cases} . \tag{4.3.2}$$

$$\forall \{u,v,w\} \in \binom{\tilde{V}}{3} : \begin{cases} X_{uv} + X_{vw} - X_{uw} \geq 0 \\ X_{uv} + X_{uw} - X_{vw} \geq 0 \\ X_{uw} + X_{vw} - X_{uv} \geq 0 \end{cases} ; \quad X_{uv} \in \{0,1\} \tag{4.3.3}$$

$$\text{minimize } \text{mod}_{\text{ILP}}(G, \mathcal{C}_G) = \sum_{\{u,v\} \in \binom{\tilde{V}}{2}} \left(\omega(u,v) - \frac{\omega(u) \cdot \omega(v)}{2 \cdot \omega(E)} \right) X_{uv} \tag{4.3.4}$$

Note that the definition of X_{uv} (pseudometric) renders this a minimization problem. Since runtimes for the full ILP reach days for more than 200 nodes, a promising idea pioneered in [136] is to solve a *partial ILP* (pILP). Such a program takes a *preclustering*—of much smaller complexity—as the input, and solves this instance, i.e., finishes the clustering, optimally via an ILP; a singleton *preclustering* yields a true ILP ($\tilde{V} = V$). We introduce two variants, (i) the argument noMerge does not merge *pre-clusters*, and only allows free nodes to join clusters or form new ones, and (ii) merge allows existing clusters to merge. For both variants we need to add constraints and terms to Equations 4.3.2-4.3.4. Roughly speaking, for (i), variables Y_{uC} indicating the distance of node u to cluster C are introduced and triplets of constraints similar to Equations 4.3.3 ensure their consistency with the X-variables; for (ii), we additionally need variables $Z_{CC'}$ for the distance between clusters, constrained just as in Equations 4.3.3. In the following we sketch out these formulations.

partial ILP

merge, noMerge

The Full ILP. If we set $\tilde{V} = V$, i.e., all nodes are "free", a full ILP formulation of *modularity*-optimization is already possible with Equations 4.3.2-4.3.4. We merely have to ensure the properties of an equivalence relation, *reflexivity*, *symmetry* and *transitivity*. Equation 4.3.3 represents transitivity, we can omit the other two: Reflexivity, $X_{uu} = 0$, is automatically ensured since a node is always in the same cluster as itself. Symmetry, $X_{uv} = X_{vu}$, is ensured since there is only one such variable.

Elements are Nodes and Preserved Clusters. For the partial ILP we usually preserve some clusters $\tilde{\mathcal{C}}$ and have only some free nodes, so $\tilde{V} \subseteq V$. Nodes are allowed to join other clusters and to form new ones, but preserved clusters can neither split nor merge. To indicate whether a free node joins a cluster, we introduce the set of *node-cluster distance* variables similar to Equations 2.4.7-2.4.9 in Section 2.4.3:

node-cluster distance variables

$$\mathcal{Y}(\tilde{V}, \tilde{\mathcal{C}}) := \{Y_{uC} : \{u, C\} \in \tilde{V} \times \tilde{\mathcal{C}}\} \quad \text{with} \quad Y_{uC} = \begin{cases} 0 & \text{if } \mathcal{C}(u) = C \\ 1 & \text{otherwise} \end{cases}. \tag{4.3.5}$$

We now need to couple these variables with $\mathcal{X}$ to ensure that if two nodes u, v join the same cluster, their variable X_{uv} also reflects that they are clustered together. Moreover a node must only join one cluster, and the objective function must evaluate such joins:

$$\forall \{u, v, w\} \in \binom{\tilde{V}}{2} \times \tilde{\mathcal{C}} : \begin{cases} X_{uv} + Y_{uC} - Y_{vC} \geq 0 \\ X_{uv} + Y_{vC} - Y_{uC} \geq 0 \\ Y_{uC} + Y_{vC} - X_{uv} \geq 0 \end{cases} ; \quad Y_{uC} \in \{0, 1\} \tag{4.3.6}$$

$$\forall u \in \tilde{V} : \sum_{C \in \tilde{\mathcal{C}}} Y_{uC} \geq k - 1 \quad \text{(a node's cluster must be unique)} \tag{4.3.7}$$

$$\begin{aligned} \text{minimize } \text{mod}_{\substack{\text{partialILP} \\ \text{no merge}}}(G, \mathcal{C}) = & \sum_{X_{uv} \in \mathcal{X}(\tilde{V})} \left(\omega(u, v) - \frac{\omega(u) \cdot \omega(v)}{2\omega(E)} \right) X_{uv} \\ & + \sum_{Y_{uC} \in \mathcal{Y}(\tilde{V}, \tilde{\mathcal{C}})} \left(\sum_{w \in C} \left(\omega(u, w) - \frac{\omega(u) \cdot \omega(w)}{2\omega(E)} \right) \right) Y_{uC} \end{aligned} \tag{4.3.8}$$

Preserved Clusters may Merge. Finally, if we also allow pre-clusters to merge, we can handle them just as we handle nodes. We thus additionally introduce *cluster-distance variables* variables, which indicate whether two clusters merge:

cluster-distance

$$\mathcal{Z}(\mathcal{C}) := \{Z_{CD} : \{C, D\} \in \binom{\tilde{\mathcal{C}}}{2}\} \quad \text{with} \quad Z_{CD} = \begin{cases} 0 & \mathsf{merge}(C, D) \\ 1 & - \end{cases} \tag{4.3.9}$$

In order to ensure consistency, we need constraints as in Equations 4.3.3 for $\mathcal{Z}$. Additionally, just as for $\mathcal{X}$ we need to couple $\mathcal{Z}$ with $\mathcal{Y}$, and let the objective function evaluate merging clusters. In turn we must now drop the constraints in Equations 4.3.7, since now a node *can* join more than one cluster—iff these clusters merge.

$$\forall \{C, D, E\} \in \binom{\tilde{\mathcal{C}}}{3} : \begin{cases} Z_{CD} + Z_{DE} - Z_{CE} \geq 0 \\ Z_{CD} + Z_{CE} - Z_{DE} \geq 0 \\ Z_{CE} + Z_{DE} - Z_{CD} \geq 0 \end{cases} \tag{4.3.10}$$

$$\forall \{u, C, D\} \in \tilde{V} \times \binom{\tilde{\mathcal{C}}}{2} : \begin{cases} Z_{CD} + Y_{uD} - Y_{uC} \geq 0 \\ Z_{CD} + Y_{uC} - Y_{uD} \geq 0 \\ Y_{uC} + Y_{uD} - Z_{CD} \geq 0 \end{cases} \tag{4.3.11}$$

$$\begin{aligned} \text{minimize } \text{mod}_{\substack{\text{partialILP} \\ \text{merge}}}(G, \mathcal{C}) = & \sum_{X_{uv} \in \mathcal{X}(\tilde{V})} \left(\omega(u, v) - \frac{\omega(u) \cdot \omega(v)}{2\omega(E)} \right) X_{uv} \\ & + \sum_{Y_{uC} \in \mathcal{Y}(\tilde{V}, \tilde{\mathcal{C}})} \left(\sum_{w \in C} \left(\omega(u, w) - \frac{\omega(u) \cdot \omega(w)}{2\omega(E)} \right) \right) Y_{uC} \\ & + \sum_{Z_{CD} \in \mathcal{Z}(\tilde{\mathcal{C}})} \left(\sum_{x \in C} \sum_{y \in D} \left(\omega(x, y) - \frac{\omega(x) \cdot \omega(y)}{2\omega(E)} \right) \right) Z_{CD} \end{aligned} \tag{4.3.12}$$

Summary of ILP variants. In Table 4.3.1 we summarize which constraints are necessary for which problem formulation. Preliminary experiments using techniques such as breaking symmetry, orbitopal fixing or lazy constraints did not seem promising although a thorough investigation might yield some mild speedup, see Section 2.4.4 for details on this. Note that for the case where merging is allowed we could also have variables as in Equation 4.3.3 for $\mathcal{Z} \cup \mathcal{X}$, and discard $\mathcal{Y}$ altogether. Note further that if in addition to the merging of clusters we also allow splitting, we actually arrive at the full ILP again. The dynamic clustering algorithms which first solicit a *preclustering* and then call ILP are called dILP. Note that they react on any edge event; accumulating events until a timestep occurs can result in prohibitive runtimes.

ILP and dILP

EOO

Elemental Optimizer. The *elemental operations optimizer*, EOO, performs a limited number of operations, trying to increase the quality. Specifically, we allow moving or splitting off nodes and merging clusters, as listed in Table 4.3.2. Although rather limited in its options, EOO is often used as a post-processing tool (see [181] for a discussion). Our algorithm dEOO calls EOO at each *time step*, doing nothing inbetween.

Table 4.3.1. ILP variants

Name	Constraint Set
Full	4.3.2-4.3.3
noMerge	4.3.2-4.3.3, 4.3.5-4.3.7
merge	4.3.2-4.3.3, 4.3.5-4.3.6, 4.3.9-4.3.11

Table 4.3.2. EOO operations, dis-/allowed via parameters

Operation	Effect
merge(u,v)	$\mathcal{C} \leftarrow (\mathcal{C} \setminus \{\mathcal{C}(u), \mathcal{C}(v)\}) \cup \{\mathcal{C}(u) \cup \mathcal{C}(v)\}$
shift(u,v)	$\mathcal{C}(u) \leftarrow \mathcal{C}(u) - u$, $\mathcal{C}(v) \leftarrow \mathcal{C}(v) + u$
split(u)	$\mathcal{C} \leftarrow (\mathcal{C} \setminus \mathcal{C}(u)) \cup \{\{u\}, (\mathcal{C}(u) - u)\}$

Table 4.3.3. Summary of the reactions of the algorithms to graph events. Isolated nodes are made singletons when inserted and simply deleted when removed. With "$\rightarrow S$" we indicate that a *prep strategy* prepares a *preclustering*.

Δ	Algorithms' reactions					
abbrev.	sGlobal	dGlobal	sLocal	dLocal	dILP	dEOO
$E + \{u, v\}$	–	$\rightarrow S$	–	$\rightarrow S$	$\rightarrow S$, pILP(args)	-
$E - \{u, v\}$	–	$\rightarrow S$	–	$\rightarrow S$	$\rightarrow S$, pILP(args)	-
$\omega(u, v) + x$	–	$\rightarrow S$	–	$\rightarrow S$	$\rightarrow S$, pILP(args)	-
$\omega(u, v) - x$	–	$\rightarrow S$	–	$\rightarrow S$	$\rightarrow S$, pILP(args)	-
$t + 1$	Global $(G_t, \mathcal{C}^V)$	Global $(G_t, \tilde{\mathcal{C}})$	Local $(G_t, \mathcal{C}^V$,all)	Local $(G_{t-1}^{0\ldots h_{\max}}, \tilde{\mathcal{C}}, \mathcal{C}_{t-1}^{1\ldots h_{\max}}$, aff/nb)	-	EOO $(G_{t+1}, \mathcal{C}_{t+1}$,args)

4.3.3.2 Strategies for Building the Preclustering

We now describe *prep strategies* which generate a *preclustering* $\tilde{\mathcal{C}}$, i.e., define the search space. We distinguish the *backtrack strategy*, which refines a clustering, and *subset strategies*, which free nodes. The rationale behind the *backtrack strategy* is that selectively backtracking the clustering produced by Global enables it to respect *changes* to the graph. On the other hand, *subset strategies* are based on the assumption that the effect of a *change* on the clustering structure is necessarily local. Both output a half-finished *preclustering*. We detail the two approaches in the following two subsections.

backtrack strategy

subset strategies

subset strategy

Subset Strategies. A *subset strategy* is applicable to all dynamic algorithms. It frees a subset $\tilde{V}$ of individual nodes that need reassessment and extracts them from their clusters. We distinguish three variants which are all based on the hypothesis that local reactions to graph *changes* are appropriate. Consider an edge event involving $\{u, v\}$. The *breakup strategy* (BU) marks the affected clusters $\tilde{V} = \mathcal{C}(u) \cup \mathcal{C}(v)$; the *neighborhood strategy* (N_d) with

BU

N_d

parameter d marks $\tilde{V} = N_d(u) \cup N_d(v)$, where $N_d(w)$ is the d-hop neighborhood of w; the *bounded neighborhood strategy* (BN_s) with parameter s marks the first s nodes found by a breadth-first search simultaneously starting from u and v.

BN_s

The Backtrack Strategy. The *backtrack strategy* (BT) records the merge operations of Global and backtracks them if a graph modification suggests their reconsideration. We detail below what we mean by "suggests", but for brevity we can state that the actions listed for BT provably require very little asymptotic effort and offer global a good chance to find an improvement. Speaking intuitively, the reactions to a *change* in (non-)edge $\{u, v\}$ are as follows (weight *changes* are analogous): For intra-cluster additions we backtrack those merge operations that led to u and v being in the same cluster and allow Global to find a tighter cluster for them, i.e., we separate them. For inter-cluster additions we track back u and v individually, until we isolate them as singletons, such that Global can re-classify and potentially merge them. Inter-cluster deletions are not reacted on. On intra-cluster deletions we again isolate both u and v such that Global may have them find separate clusters. For more details on these operations continue reading. Note that this strategy is only applicable to Global; conferring it to Local is neither straightforward nor promising, as Local is based on node *migrations* in addition to *agglomerations*. Anticipating this strategy's low runtime, we can give a bound on the expected number of backtrack steps for a single call of isolate, being the crucial operation. We leave its formal proof to the more general Theorem 4.3.2 below:

backtrack strategy
BT
clever backtracking
isolate and backtrack

Theorem 4.3.1 *Assume that a backtrack step divides a cluster randomly. Then, for the number I of steps isolate(v) requires, it holds: $E\{I\} \in \Theta(\ln n)$.*

isolate is quick

To motivate the backtrack strategy we first detail some insights on the change in *modularity* if (i) the graph changes and (ii) we decide to move some nodes from one cluster to another, in order to react to the *change*. Please note that all statements generalize trivially to weighted edges. Let $C \in \mathcal{C}$ be a cluster and $D \in \{\mathcal{C} \cup \emptyset\}$ be a cluster or the empty set. Let further $U \subset C$ be a subset of C, and define further the clustering $\mathcal{D}$:

$$\mathcal{D} := (\mathcal{C} \setminus \{C, D\}) \cup \{C \setminus U, D \cup U\} \quad \text{(move } U \text{ from } C \text{ to } D\text{)} \tag{4.3.13}$$

The basis of *modularity* (*Mod*), *coverage* (*Cov*) and the expected value of *coverage* (*ECov*) change when we move from clustering $\mathcal{C}$ to $\mathcal{D}$; we can express these changes and the *change* Δ in *modularity* as follows:

how modularity changes

$$\Delta_{Cov} := Cov(\mathcal{D}) - Cov(\mathcal{C}) \ , \quad \Delta_{ECov} := ECov(\mathcal{D}) - ECov(\mathcal{C}) \ , \tag{4.3.14}$$

$$\Delta := Mod(\mathcal{D}) - Mod(\mathcal{C}) = \Delta_{Cov} - \Delta_{ECov} \tag{4.3.15}$$

Table 4.3.4. Overview of how strategies handle graph events. Changes to edges' weights are analog to creations/removals. Degree-0 nodes are universally made singletons when inserted and ignored when removed.

Event	Reaction			
	BT	BU, $\tilde{V} =$	N, $\tilde{V} =$	BN, $\tilde{V} =$
$E + \{u, v\}$	$\begin{cases} \mathsf{sep}(u, v) & \mathcal{C}(u) = \mathcal{C}(v) \\ \mathsf{iso}(u), \mathsf{iso}(v) & \mathcal{C}(u) \neq \mathcal{C}(v) \end{cases}$	$\mathcal{C}(u) \cup \mathcal{C}(v)$	$N_d(u) \cup N_d(v)$	$\mathrm{BFS}\{u, v\}_{\mid s}$
$E - \{u, v\}$	$\begin{cases} \mathsf{iso}(u), \mathsf{iso}(v) & \mathcal{C}(u) = \mathcal{C}(v) \\ - & \mathcal{C}(u) \neq \mathcal{C}(v) \end{cases}$	$\mathcal{C}(u) \cup \mathcal{C}(v)$	$N_d(u) \cup N_d(v)$	$\mathrm{BFS}\{u, v\}_{\mid s}$

Note that Δ must be non-positive if $\mathcal{C}$ was optimal. If we generalize the definitions in Sec. 4.1.3 from clusters to general sets of nodes, then we can write these as:

$$\Delta_{Cov} := \frac{E(U,D) - E(U, C \setminus U)}{m} \ , \tag{4.3.16}$$

$$\Delta_{ECov} := \frac{1}{4m^2}\left(\sum_{B\in\mathcal{D}} \deg^2(B) - \sum_{B\in\mathcal{C}} \deg^2(B)\right) \tag{4.3.17}$$

$$= \frac{1}{4m^2}\left(\deg^2(D\cup U) + \deg^2(C\setminus U) - \deg^2(D) - \deg^2(C)\right) \tag{4.3.18}$$

$$= \frac{\deg(U)}{2m^2}(\deg(D) - \deg(C\setminus U)) \tag{4.3.19}$$

Given a *change* in the graph we want to know whether moving from $\mathcal{C}$ to $\mathcal{D}$ is beneficial. Thus, in addition to moving from $\mathcal{C}$ to $\mathcal{D}$, we now move from G to G', i.e., we change graph G by, say, adding edge $\{v, w\}$. Analogously to the above we now define Δ'_{Cov}, Δ'_{ECov} and Δ'_{Mod}. We can now establish sufficient and necessary conditions for Δ' to be positive if $\Delta \leq 0$, the following two Tabs. 4.3.5-4.3.6. We distinguish cases whether or not v and w are elements of C, D or U. In both tables, this is done in the first column and the second columns give the appropriate values of Δ'_{Cov} and Δ'_{ECov}. The last columns give tight conditions for Δ' to be strictly positive, i.e., for the case when moving such a set U from C to D increases *modularity* in G'.

how to change the clustering

Summarizing, if we want to adapt a clustering to a *change* in a graph by moving a set U of nodes between clusters, the given ranges for Δ_{ECov} categorize exactly when a given set U (specified by the first column) will increase *modularity* for the new graph G'. However, this does not determine a *specific* set U—we still have to decide on this, but by the size of the range for Δ_{ECov} we can deduce some structure. Since we aim at a dynamization dGlobal of the global agglomerative algorithm, a reasonable approach is as follows: Track back specific merge operations of the static algorithm sGlobal until the most promising (according to Tabs. 4.3.5-4.3.6) operations in terms of moving a set U are available; then let the algorithm finish the clustering for G'. Of course this does not yield optimality by any means, nor does it *identify* the best set U, but it gives Global a fighting chance to find a good improvement with minimum effort, since exclusively the most promising parts of the clustering are broken up.

Table 4.3.5. The different effects on *modularity* if, after the *creation* of edge $\{v, w\}$, we move a specific subset U of nodes from cluster C to cluster D.

preconditions	formulae for Δ'_{Cov} and Δ'_{ECov}	$\Delta \leq 0$ and $\Delta' > 0$ iff
$v, w \notin C \cup D$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov})$
$v \in C\setminus U,\ w \notin D$ or $w \in C\setminus U,\ v \notin D$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov} - \frac{\deg_G(U)}{2(m+1)^2}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov} + \frac{\deg_G(U)}{2m^2})$
$v \in C\setminus U,\ w \in D$ or $w \in C\setminus U,\ v \in D$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov})$
$v \notin C,\ w \in D$ or $w \notin C,\ v \in D$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov} + \frac{\deg_G(U)}{2(m+1)^2}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov} - \frac{\deg_G(U))}{2m^2})$
$v \in U,\ w \notin D$ or $w \in U,\ v \notin D$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov})$
$v \in U,\ w \in D$ or $w \in U,\ v \in D$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov} + \frac{1}{m+1}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov} + \frac{\deg_G(U)}{2(m+1)^2}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov} + \frac{2m+2-\deg_G(U)}{2m^2})$

Table 4.3.6. For the *removal* of edge $\{v, w\}$, this table details effects analogously to Tab. 4.3.5.

preconditions	formulae for Δ'_{Cov} and Δ'_{ECov}	$\Delta \leq 0$ and $\Delta' > 0$ iff
$v, w \notin C \cup D$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov})$
$v, w \in C \backslash U$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov} - \frac{\deg_G(U)}{(m+1)^2}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov} + \frac{\deg_G(U)}{m^2})$
$v, w \in D$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov} + \frac{\deg_G(U)}{(m+1)^2}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov} - \frac{\deg_G(U)}{m^2})$
$v, w \in U$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov} + \frac{\deg_G(D)-\deg_G(C\backslash U)}{(m+1)^2}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov} - \frac{\deg_G(D)-\deg_G(C\backslash U)}{m^2})$
$v \in U,\ w \in C\backslash U$ or $w \in U,\ v \in C\backslash U$	$\Delta'_{Cov} = \frac{m}{m+1}\Delta_{Cov} - \frac{1}{m+1}$ $\Delta'_{ECov} = (\frac{m}{m+1})^2\Delta_{ECov} + \frac{\deg_G(D)-\deg_G(C)-1}{2(m+1)^2}$	$\Delta_{ECov} \in$ $[\Delta_{Cov}, (1+\frac{1}{m})\Delta_{Cov} - \frac{2m+1+\deg_G(D)-\deg_G(C)}{2m^2})$

In the following we detail our update procedures which are motivated by the above discussion. For these we require the helper algorithms given in Algs. 22-24. Algorithm 22, backtrack(v), splits the cluster containing v into those two parts it resulted from. Algorithm 23, isolate(v), iteratively backtracks those merges that involved v, until v is contained in a singleton. Algorithm 24, separate(u, v), backtracks those merges involving the cluster of u and v until u and v are in different clusters.

operations for the backtrack strategy

Algorithm 22: backtrack(v)

(assume $\mathcal{C}(v) =$ merge(A, B) as done by global)

1 $\mathcal{C} \leftarrow (\mathcal{C} - \mathcal{C}(v)) \cup \{A, B\}$

Algorithm 23: isolate(v)

1 **while** $|\mathcal{C}(v)| \neq 1$ **do**
2 backtrack(v)

Algorithm 24: separate(u, v)

1 **while** $\mathcal{C}(u) = \mathcal{C}(v)$ **do**
2 backtrack(u)

Backtracking Inter-Cluster Edge Addition. Since we assume for most graphs that the degree of each cluster does not exceed m, we have the best chances to increase *modularity* if we choose U to contain either v or w and move U to the cluster containing the other vertex (see sixth case in Tab. 4.3.5). Therefore, we define this part of our backtrack strategy as isolate(u), isolate(v).

Backtracking Intra-Cluster Edge Addition. In this case it seems to be the best choice to set U in such a way that it is a subset of C and contains either one of v or w, or both (cases two and four in Tab. 4.3.5; regarding case four, note that D can be empty, which implies $\deg_G(D) = 0$). We thus define this backtrack case as separate(u, v). Since, however, the expected number of backtrack operations is 2 if we assume that in each such split, u and v are separated with probability $1/2$ (see below for details), one might think that this is too little an invested effort. We thus also tested an alternative which performs isolate(u) and isolate(v); however this uniformly did not raise quality but only runtime and distance.

Backtracking Edge Deletions. For the case of edge deletions it is more difficult to find good backtracking strategies. For additions we can try to reasonably reduce the sizes of the affected clusters by splitting them into parts that either form new clusters or merge with existing ones. The strategy in the case of deletions should thus be the inverse: Split off parts of the clusters that are unaffected by the edge deletion and link them with the affected ones. But how do we know which cluster to split? We leave this question unanswered, analytically, and rely on common sense to define the following procedures:

- Inter-cluster edge deletion: *do nothing* ($\tilde{\mathcal{C}} = \mathcal{C}$, but do call global, as usual)
- Intra-cluster edge deletion: isolate(u), isolate(v)

Analysis of the isolate and the separate Operations. It is easy to see that the expected number $E\{S\}$ of backtrack steps S for a single call of separate(u, v) is 2, if we assume that a backtrack step divides a cluster randomly and thus separates u and v with probability $1/2$. Without further a priori knowledge this is a reasonable assumption; however, it is crucial to note that all our findings (The. 4.3.1 in particular) remain valid for any arbitrary but fixed constant probability instead of $1/2$. For simplicity we use $1/2$ in the following. Then, S is distributed according to the geometric distribution with parameter $1/2$ yielding $E\{S\} = 2$.

expect two backtracks from separate

For the proof that the expected number $E\{I\}$ of backtrack steps I for a single call of isolate(v) is in $\Theta(\ln n)$ (see Theorem 4.3.1), we require the following two lemmas for the theorem that proves the bound.

1st helping lemma

Lemma 4.3.1 *Let $(\Omega, \mathcal{A}, P)$ be a probability space, $A_1, \ldots, A_n \in \mathcal{A}$ independent events with $P(A_i) = p, i = 1, \ldots, n$. Then*

$$P\left(\bigcup_{i=1}^{n} A_i\right) = 1 - (1-p)^n \ .$$

(Proof omitted)

2nd helping lemma

Lemma 4.3.2 *Let $i \in \mathbb{N}_0, j \in \mathbb{N}$. Then it holds that*

$$\int_0^\infty \left(\frac{1}{2}\right)^{jx} \left(1 - \left(\frac{1}{2}\right)^x\right)^i dx = \frac{i!(j-1)!}{(j+i-1)!} \cdot \frac{1}{\ln 2} \cdot \frac{1}{i+j} \ .$$

Specifically, for $j = 1$ the following equation holds:

$$\int_0^\infty \left(\frac{1}{2}\right)^{x} \left(1 - \left(\frac{1}{2}\right)^x\right)^i dx = \frac{1}{\ln 2} \cdot \frac{1}{i+1}$$

Proof. The proof uses induction over i, with integration by parts for the induction step, for brevity we just give a proof sketch.

$$\int_0^\infty \underbrace{\left(\frac{1}{2}\right)^{jx}}_{g'} \underbrace{\left(1 - \left(\frac{1}{2}\right)^x\right)^i}_{f} dx \tag{4.3.20}$$

$$\left(\text{such that } g = -\frac{1}{j \ln 2}\left(\frac{1}{2}\right)^{jx} \text{ and } f' = \ln 2 \left(\frac{1}{2}\right)^x i \left(1 - \left(\frac{1}{2}\right)^x\right)^{i-1} \right)$$

$$= \left[-\frac{1}{j \ln 2}\left(\frac{1}{2}\right)^{jx} \cdot \left(1 - \left(\frac{1}{2}\right)^x\right)^i\right]_0^\infty \quad (= 0 \text{ for } i \neq 0, \text{ which holds here})$$

$$- \int_0^\infty -\frac{1}{j \ln 2}\left(\frac{1}{2}\right)^{jx} \cdot \ln 2 \left(\frac{1}{2}\right)^x i \left(1 - \left(\frac{1}{2}\right)^x\right)^{i-1} dx \tag{4.3.21}$$

$$= 0 + \frac{i}{j} \int_0^\infty \left(\frac{1}{2}\right)^{(j+1)x} \left(1 - \left(\frac{1}{2}\right)^x\right)^{i-1} dx \tag{4.3.22}$$

$$\vdots$$

$$= \frac{i \cdot \ldots \cdot 1}{j \cdot \ldots \cdot j + i - 1} \cdot \int_0^\infty \left(\frac{1}{2}\right)^{(i+j)x} dx \tag{4.3.23}$$

$$= \frac{i!(j-1)!}{(j+i-1)!} \cdot \frac{1}{\ln 2} \cdot \frac{1}{i+j} \tag{4.3.24}$$

The integrand in line (4.3.20) is split into functions g' and f. Since f and g are continuously differentiable functions integration by parts yields line (4.3.21). Note that as long as $i \neq 0$ in line (4.3.20), the first (integrated) summand always equals zero. Line (4.3.22) just summarizes terms in order to resemble our starting point in line (4.3.20). We can now repeat these steps i times, such that in each step i decreases by one (reaching 0), j increases by one (reaching $j+i$) and new factors are accumulated. We thus reach line (4.3.23) where in the integrand we now have $f = 1$, such that we can solve the remaining integral. □

Theorem 4.3.2 *Let $n \in \mathbb{N}$ and $X_j^{(i)}, i = 1, \ldots, n,\ j = 1, 2, \ldots$ be i.i.d. random variables that are Bernoulli-distributed with parameter $\frac{1}{2}$. We define*

$$N := \min\{k \in \mathbb{N}_0 : \forall i \in \{2, \ldots, n\}\ \exists j \in \{1, \ldots, k\} : X_j^{(i)} \neq X_j^{(1)}\}\ .$$

Then it follows for the expectance of N:

$$E\{N\} \in \Theta(\ln n)$$

expect $\Theta(\ln n)$ backtracks from isolate

Proof. W.l.o.g. $n \geq 2$.

$$E\{N\} = \sum_{k=0}^{\infty} P(N = k) \cdot k = \sum_{k=0}^{\infty} P(N > k)$$

$$= \sum_{k=0}^{\infty} P(\exists i \in \{2, \ldots, n\}\ \forall j \in \{1, \ldots, k\} : X_j^{(i)} = X_j^{(1)})$$

(set event $A_i : \forall j \in \{1, \ldots, k\} : X_j^{(i)} = X_j^{(1)}$, then it holds $P(A_i) = \left(\frac{1}{2}\right)^k$)

$$= \sum_{k=0}^{\infty} 1 - \left(1 - \left(\frac{1}{2}\right)^k\right)^{n-1} \qquad \text{(Lemma 4.3.1)}$$

$$= \sum_{k=0}^{\infty} \frac{1 - \left(1 - \left(\frac{1}{2}\right)^k\right)^{n-1}}{1 - \left(1 - \left(\frac{1}{2}\right)^k\right)} \left(\frac{1}{2}\right)^k \qquad \text{(rewrite)}$$

$$= \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k \sum_{i=0}^{n-2} \left(1 - \left(\frac{1}{2}\right)^k\right)^i \qquad \text{(geom. series)}$$

$$= \sum_{i=0}^{n-2} \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k \cdot \left(1 - \left(\frac{1}{2}\right)^k\right)^i \qquad \text{(reorder)}$$

$$\in \Theta\left(\sum_{i=0}^{n-2} \int_0^{\infty} \left(\frac{1}{2}\right)^x \cdot \left(1 - \left(\frac{1}{2}\right)^x\right)^i dx\right) \qquad \text{(rect. approx.)}$$

$$\in \Theta\left(\sum_{i=0}^{n-2} \frac{1}{\ln 2} \cdot \frac{1}{i+1}\right) \qquad \text{(Lemma 4.3.2)}$$

$$\in \Theta\left(\frac{1}{\ln 2} \sum_{i=0}^{n-2} \frac{1}{i+1}\right) \in \Theta\left(\frac{1}{\ln 2}\left(\sum_{i=1}^{n} \frac{1}{i} - \frac{1}{n}\right)\right)$$

$$\in \Theta(\ln n) \qquad (n\text{-th harmonic no.})$$

□

We can interpret the random variables $X_j^{(i)} \in \{0, 1\}$ of experiments X_j such that for the j-th

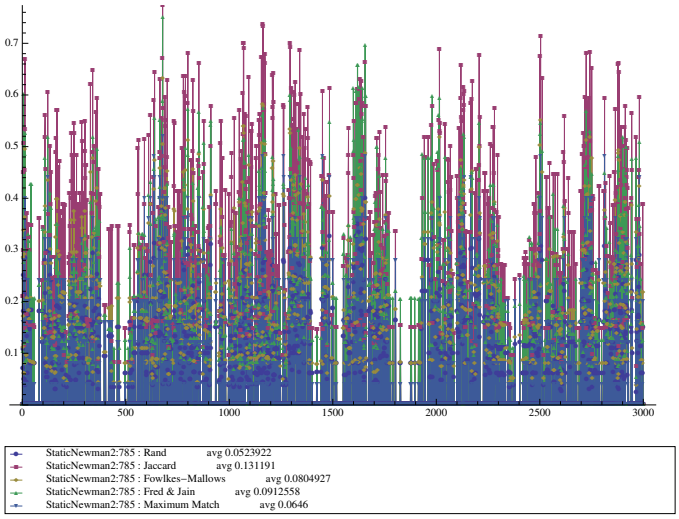


Figure 4.3.1. Raw data for several distance measures

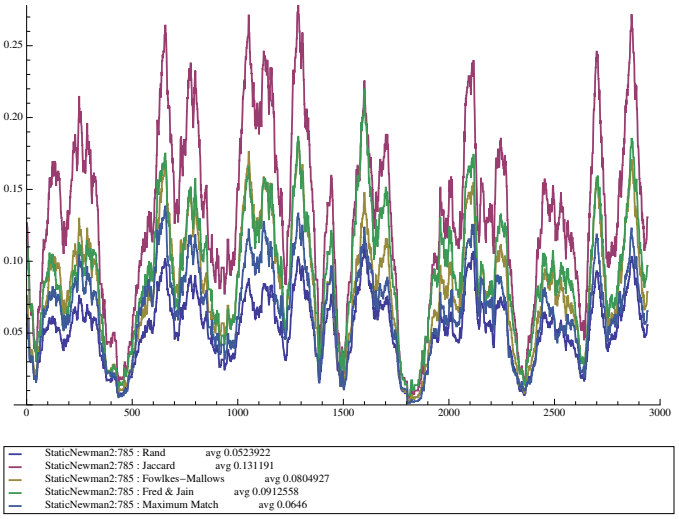


Figure 4.3.2. Smoothed version of Fig. 4.3.1

division (by a backtrack step), $X_j^{(i)} = 0$ if the i-th node is in the left half, and $X_j^{(i)} = 1$ if the i-th node is in the right half. If we assign to node v index 1, then $X_j^{(i)} = X_j^{(1)}$ means, that node i is in the same half as v. Event A_i then means that for all experiments $1, \ldots, k$ node i always ended up on the same half as v. We thus look for the first experiment such that each node other than v has ended up in another half than v at least once (note that multiply separating a node i from v doe not alter the statement). Now its easy to see that Theorem 4.3.2 proves Theorem 4.3.1.

The. 4.3.2 proves The. 4.3.1

4.3.4 Experimental Evaluation of Dynamic Algorithms[13]

For the sake of readability, we use a moving average in plots for distance and quality to smoothen the raw data. An example of this effect is given in Figures 4.3.1 and 4.3.2. These are representative plots for an arbitrary random dynamic graph and an arbitrary dynamic clustering algorithm, others behave similarly in terms of readability. The second observation this example shows is the fact mentioned in Subsection 4.3.1.1 above: different measures for *smoothness* do not differ qualitatively; again, we observed the same for all other graphs and algorithms. We consider the criteria quality (*modularity*), *smoothness* ($\mathcal{R}_g$) and runtime (ms), and additionally $|\mathcal{C}|$. Generally speaking, the x-axis always indicates the current *time step*, and the y-axis gives the measurement as described in the corresponding legend.

4.3.4.1 Instances

We use two kinds of dynamic instances, generated graphs and a real-world instance with practical relevance. We briefly describe both here, but for more details we refer the reader to Sections 4.2 and to 5.1.1, respectively.

Email Graphs. The network of email contacts at the department of computer science at KIT is an ever-changing graph with an inherent clustering: Workgroups and projects cause increased communication. We weigh edges by the number of exchanged emails during the past seven days, thus edges can completely time out; degree-0 nodes are removed from the network. This network, $\mathcal{G}_e$, has between 100 and 1500 nodes depending on the time of year, and about 700K events spanning about 2.5 years.

[13] Supplementary information, in the form of many *Mathematica* notebooks containing further experimental results, is stored at `i11www.iti.uni-karlsruhe.de/projects/spp1307/dyneval`

It features a strong power-law degree distribution. Figure 4.3.3 shows the temporal development of the email graph in terms of n (lower) and m (upper) per 100 events. The first peak stems from a spam attack in late '06, the two large drops from Christmas breaks and the smaller drops from spring and autumn breaks (details on this data set can be found in Section 5.1.1). Unless otherwise mentioned we use $b = 100$ for $\mathcal{G}_e$, yielding 7000 timesteps of 100 events each.

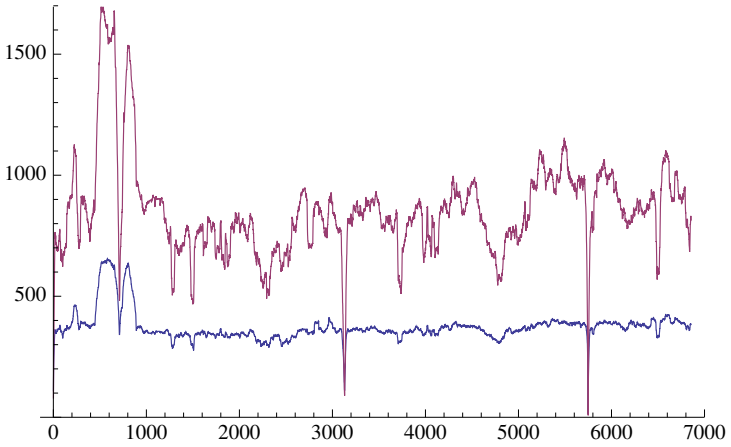
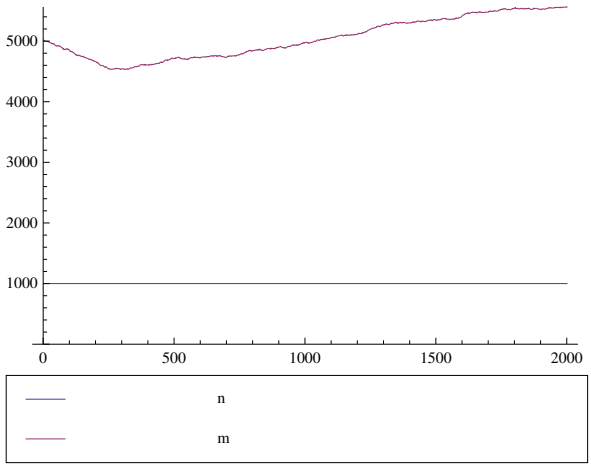
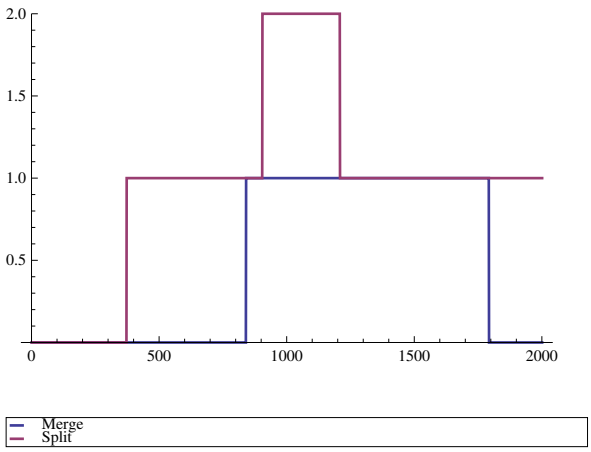


Figure 4.3.3. Nodes (**blue**) and edges (**purple**) of $\mathcal{G}_e$

Random Graphs. Our Erdős-Rényi-type generator builds upon the work of [47] and adds to this dynamicity in all graph elements and in the clustering, i.e., nodes and edges are inserted and removed and *ground-truth* clusters merged and split, always complying with sound probabilities. The visible clustering of the generator is stored as a reference to compare our algorithms to (please refer to Section 4.2 for details). We conducted experiments for a large number of settings, varying size, density, node/edge-volatility, stability of clusters, etc., and in the following only give representative plots, and point out specific peculiarities. The majority of plots uses a representative graph coined $\mathcal{G}_1$, one of our simpler test instances. It is is used in many examples, as behavior on it is largely archetypical; Figure 4.3.4 depicts its rough statistics. As another example, Figure 4.3.5 shows the statistics of a graph which gradually grows in the number of nodes. With an average node degree of 10 and a batch size of 10 an average node insertion or deletion constitutes approximately one batch update.



(a) Numbers of nodes end edges, this instance does not allow node events but only edge events for well controlled experiments.



(b) Numbers and types of changes in the clustering, plateaus indicate that a *ground-truth change* has not yet been reacted to by the reference, i.e., arguably is not yet well visible in the graph.

Figure 4.3.4. Our primary example, $\mathcal{G}_1$, non-default parameters of its generation: (see Table 4.2.1) are $t_{\max} = 2000$, $n_0 = 1000$, $k = 20$, $\eta = 10$, $p_\omega = 0.001$, $p_{\text{in}} = 0.1$, $p_{\text{out}} = 0.005$.

4.3.4.2 Fundamental Results on the Algorithms

Parameters of Local. It has been stated in [38] that the order in which Local considers nodes is irrelevant. In terms of average runtime and quality we can confirm this for sLocal, though with a few exceptions a random order tends to be marginally less *smooth*; for dLocal the same observation holds. Figure 4.3.6 shows representative plots on the *smoothness* for $\mathcal{G}_e$ and $\mathcal{G}_1$ for sLocal and dLocal (different choices for the other parameters yield similar results).

which P for Local

However, since node order *does* influence specific values, a random order can compensate the effects this might have in pathological cases. We will later see more advantages. Remember that Local clusters in several hierarchical contraction levels (see Algorithm 21), such that on the base level a *prep strategy* frees nodes and on higher levels, a policy P decides whether only affected nodes or their neighbors as well are freed. We found that considering only affected nodes or also their neighbors in higher levels, does not affect *any* criterion on average (we omit plots on this). Thus we prefer the affected policy, being the simpler variant.

dEOO alone: bad

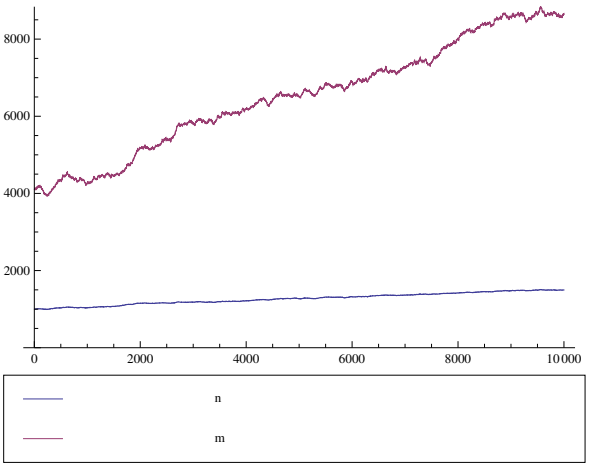
Discarding dEOO. In a first feasibility test, dEOO immediately falls behind all other algorithms in terms of quality (Fig. 4.3.7). This observation is substantiated by the fact that postprocessing such as dEOO work better if related to the underlying algorithm, as found recently in [181]. Moreover, runtimes for dEOO as the sole technique are infeasible for large graphs. We can conclude that dEOO should not be used as a standalone technique.

noMerge better than merge

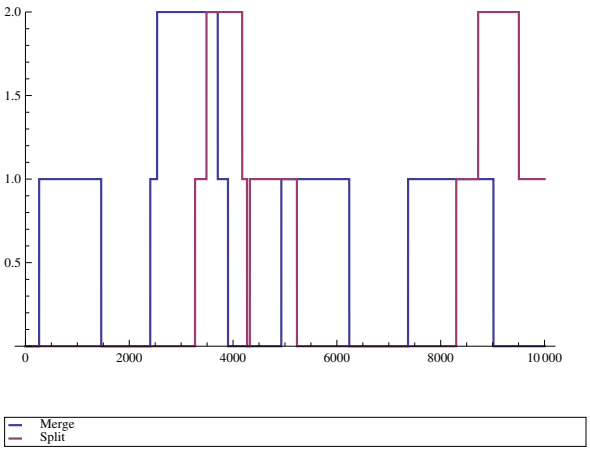
pILP Variants. Allowing the ILP to merge existing clusters takes longer, and clusters coarser—as which is quite intuitive—but also yields a slightly worse *modularity*. We conjecture that the reason for this is that merging invites hazardous local optima. We made this observation on almost all tested instances, and we therefore reject merge for pILP. Nicely visible in Figure 4.3.8b is how in terms of the number of clusters merge and noMerge bound dLocal and dGlobal from below and above, respectively.

pILP overfits

Heuristics vs. pILP. A striking observation we made about the quality of pILP is the fact that it yields worse quality than dLocal and sLocal with identical *prep strategies*, as in Fig. 4.3.8a. Intuitively speaking, pILP solves similar problems in each timestep as the other real heuristics do, but whithin the same restrictions, pILP solves them optimally. We thus clearly expect pILP to yield better quality—but this does not happen. Being locally optimal seems to overfit, a phenomenon that does not weaken over time and persists throughout other instances! Together with its high runtime and only small advantages in *smoothness* pILP seems ill-suited for updates on large graphs.

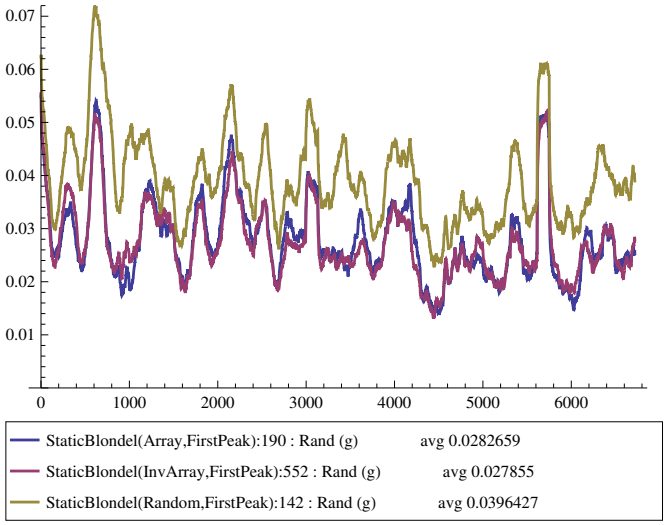


(a) The numbers of nodes end edges both grow.

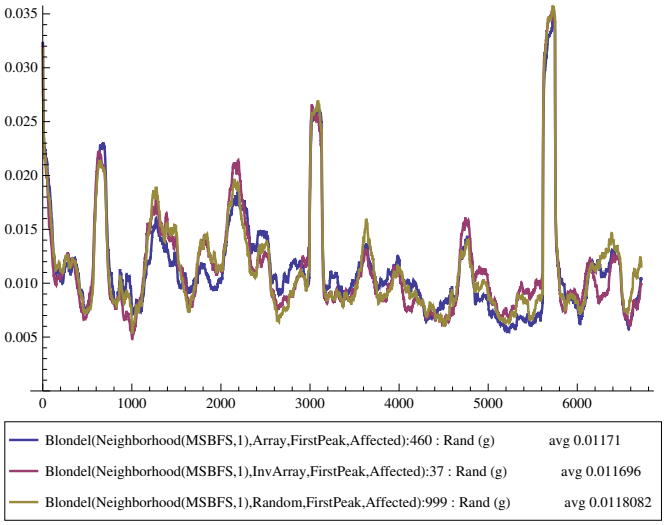


(b) 10000 *time steps* allow more changes in the clustering.

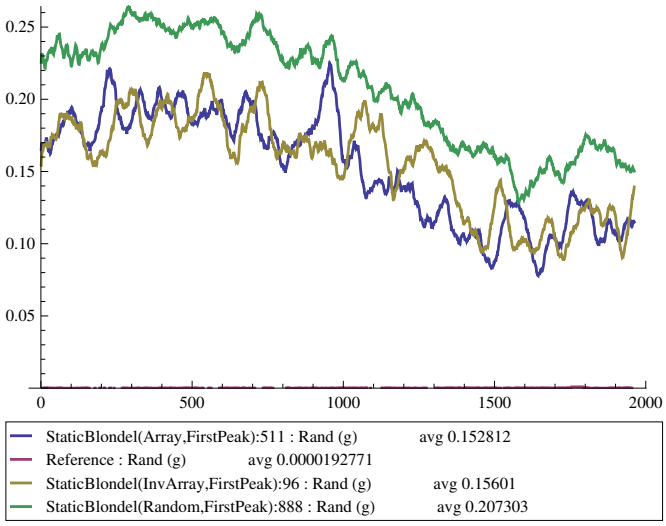
Figure 4.3.5. Here, non-default generation parameters (see Table 4.2.1) are $t_{\max} = 10000$, $n_0 = 1000$, $k = 20$, $\eta = 10$, $p_\omega = 0.001$, $p_{\text{in}} = 0.1$, $p_{\text{out}} = 0.002$, $\chi = 0.9$, $p_\nu = 0.55$.



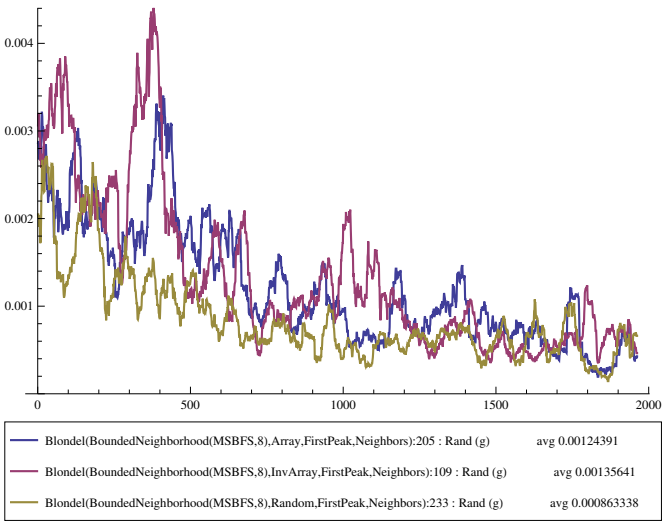
(a) sLocal on $\mathcal{G}_e$



(b) dLocal on $\mathcal{G}_e$



(c) sLocal on $\mathcal{G}_1$



(d) dLocal on $\mathcal{G}_1$

Figure 4.3.6. These plots show effect of different node orders, Random and two fixed orders (Array and InvArray), for Algorithms based on Local, on *smoothness*, in terms of $\mathcal{R}_g$-distance.

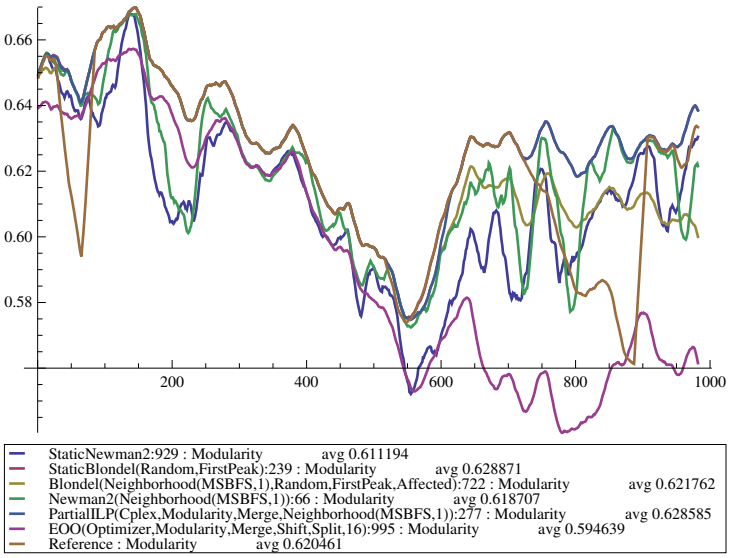
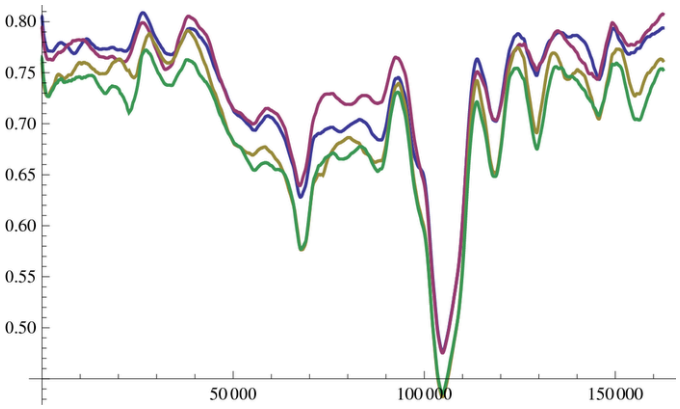
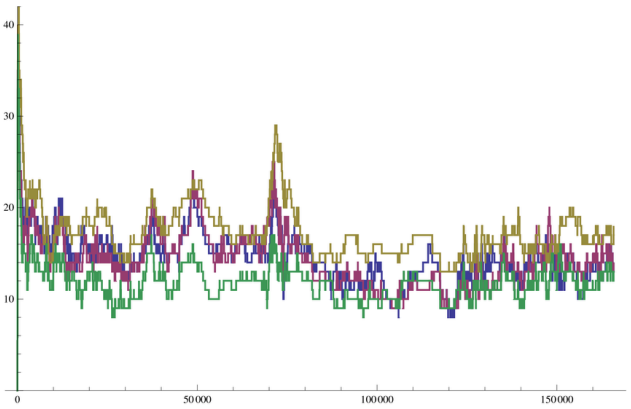


Figure 4.3.7. In terms of *modularity*, dEOO (**purple**) lags behind the other algorithms.



(a) *Modularity*: surprisingly, both pILP-variants perform consistently worse than dLocal and dGlobal.



(b) Number of clusters: merge and noMerge bound the other algorithms from below and above, respectively.

Figure 4.3.8. dLocal (**blue**), dGlobal (**purple**), pILP(noMerge) (**yellow**) and pILP(merge) (**green**) on the first quarter of $\mathcal{G}_e$, batch size 1 for comparability, strategy BN_8; results are representative for most instances.

sLocal vs. sGlobal

Static Algorithms. Briefly comparing sGlobal and sLocal we can state that sLocal consistently yields better quality and a finer (see Figure 4.3.13 at the end) yet less *smooth* clustering. This observation has been made for other (huge) instances in [38] and we can confirm it on all our generated instances; additionally, these results are paralleled by the dynamic counterparts. An exception is instance $\mathcal{G}_e$, as discussed later.

4.3.4.3 Prep Strategies

We now determine the best choice of *prep strategies* and their parameters for dGlobal and dLocal. In particular, we evaluate N_d for $d \in \{0, 1, 2, 3\}$ and BN_s for $s \in \{2, 4, 8, 16, 32\}$, alongside BU and BT. Throughout our experiments $d = 0$ (or $s = 2$) proved insufficient, and is therefore ignored in the following. For dLocal, increasing d has only a marginal effect on quality and *smoothness*, while runtime grows sublinearly, which suggests $d = 1$. Please

review Figure 4.3.9 for these observations. Similar facts hold for other instances and batch sizes. Note that large batch sizes b let a *prep strategy* accumulate many free nodes yielding a larger search space; however, we observed that a small b does *not* benefit from larger search spaces. For dGlobal, N_d risks high runtimes for depths $d > 1$, especially for dense graphs. In terms of quality N_1 is the best choice, higher depths seem to deteriorate quality—a strong indication that large search spaces contain local optima. *Smoothness* approaches the bad values of sGlobal for $d > 2$. We omit plots for dGlobal on this.

large d does not help N_d

For BN, increasing s is essentially equivalent to increasing d, only on a finer scale. Consequently, we can report similar observations. For dLocal, BN_4 proved slightly superior. dGlobal's quality benefits from increasing s in this range, but again at the cost of speed and *smoothness*, so that BN_{16} is a reasonable choice. Figure 4.3.10 illustrates these observations for dGlobal on $\mathcal{G}_1$, we omit plots for dLocal. Again we could confirm these findings on other instances.

moderate s helps BN_s

The strategy which simply breaks up all clusters affected by *changes*, BU, clearly falls behind in terms of all criteria compared to the other strategies, and often mimics the static algorithms. As expected we can discard this strategy and rather consider it as a "control". Note that this is a very basic confirmation of the assumption that local updates are a good idea.

BU is bad

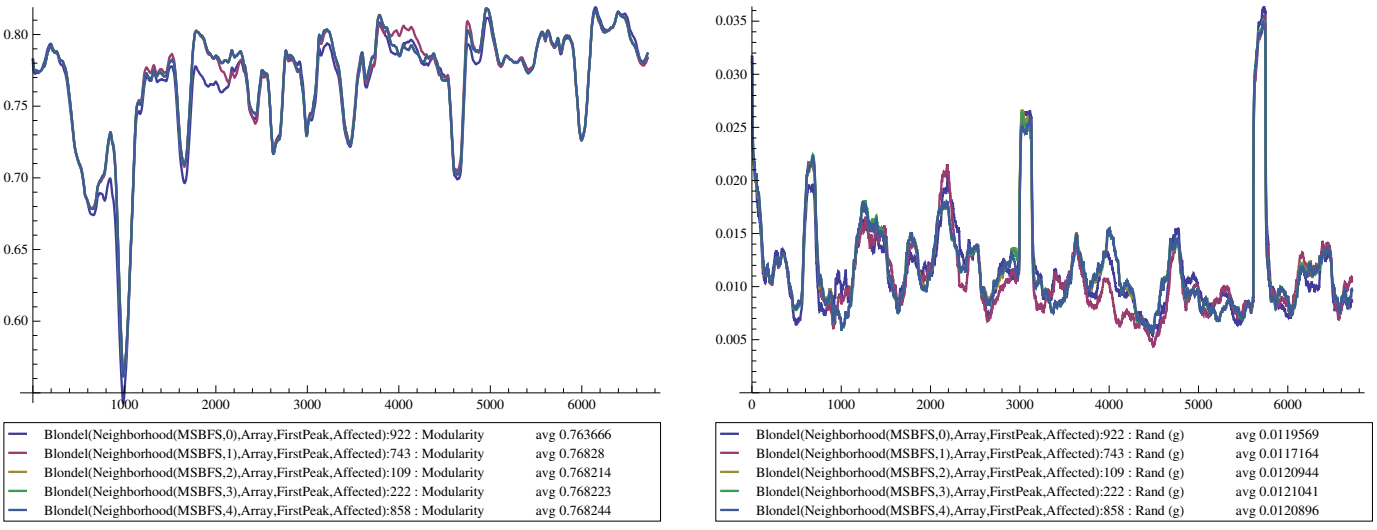
dGlobal using BT is by far the fastest algorithm, confirming our theoretical predictions from Sec. 4.3.3.2, but still produces competitive quality. However, it often yields a *smoothness* in the range of sGlobal. Summarizing, our best dynamic candidates are dGlobal@BT and $\mathsf{dGlobal@BN}_{16}$ (achieving a speedup over sGlobal of up to 1k and 20 at 1k nodes, respectively) and $\mathsf{dLocal@BN}_4$(with a speedup of 5 over sLocal).

BT is fast but non-smooth

4.3.4.4 Comparison of the Best and their Static Counterparts

We now take a focused look at those dynamic clustering algorithms and *prep strategies*, which we observed to be the most promising and compare them with their static counterpart. As a general observation, as depicted in Figure 4.3.11a, each dynamic candidate beats its static counterpart in terms of *modularity*. On the generated graphs, dLocal is superior to dGlobal, and faster, this is not the case for the email network—here both Global algorithms beat each Local algorithm. In terms of *smoothness* (Figure 4.3.11b), dynamics (except for dGlobal@BT)

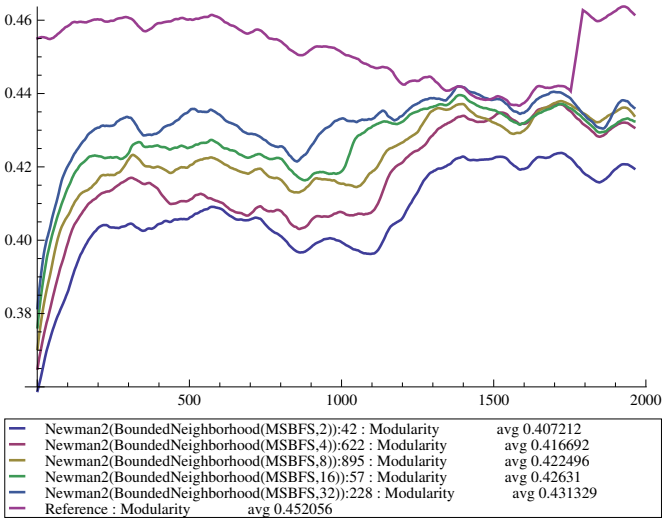
dynamic beats static



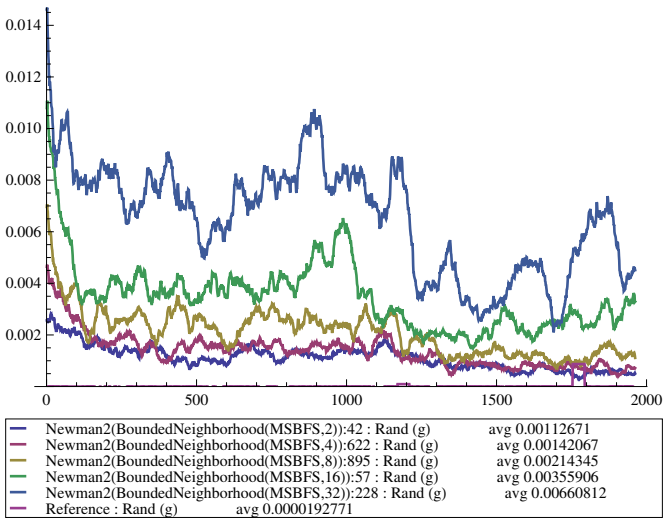
(a) *Modularity* does not improve with growing d.

(b) Smoothness worsens with growing d.

Figure 4.3.9. These plots show the effect of d for strategy N_d on the behavior of dLocal for $\mathcal{G}_e$.

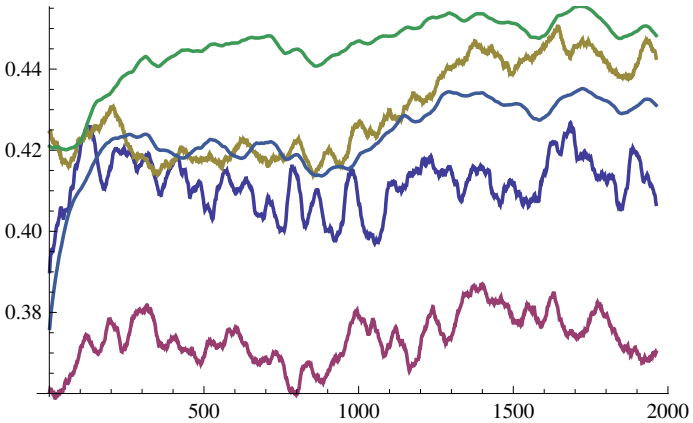


(a) *Modularity* slightly improves with growing s.

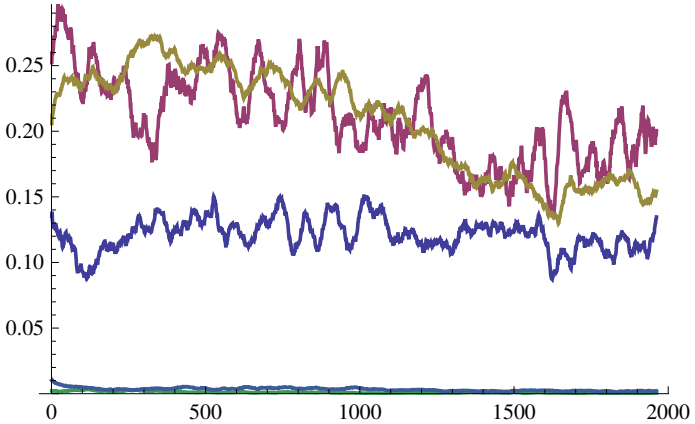


(b) Smoothness worsens with growing s.

Figure 4.3.10. The effect of s for strategy BN_s on the behavior of dGlobal for $\mathcal{G}_1$;we can clearly observe a gradual convergence on both plots; note further how *modularity* for the reference jumps due to a finished cluster event (see Section 4.2).



(a) *Modularity*: a rather clear ranking is possible for quality: Locals surpass Globals and dynamics surpass statics.



(b) $\mathcal{R}_g$: only dGlobal@BT is within the order of magnitude of the static algorithms, the other dynamics excel in smoothness.

Figure 4.3.11. With respect to quality and to *smoothness* dGlobal@BT (**dark blue**) and $\mathsf{dGlobal@BN}_{16}$ (**light blue**) beat sGlobal (**purple**), and $\mathsf{dLocal@BN}_4$ (**green**) beats sLocal (**yellow**) on $\mathcal{G}_1$.

are superior to statics by a factor of ca. 100, and even dGlobal@BT beats them.

4.3.4.5 Dynamic Algorithms React Quickly to Changing Clusterings

quick reactivity

Throughout our experiments we observed that the dynamic algorithms exhibit the ability to react quickly and aptly to changes in the *ground-truth* clustering. Figure 4.3.12 shows an example where our best dynamic algorithms quickly cope with rapid changes to the clustering—in contrast to the reference clustering, with its rather clumsy, stepwise adaption. The changes in the *ground-truth* clustering are visible by the drops in the reference quality. At each such change, after brief depressions, the *modularity* values of all algorithms rise to their old levels.

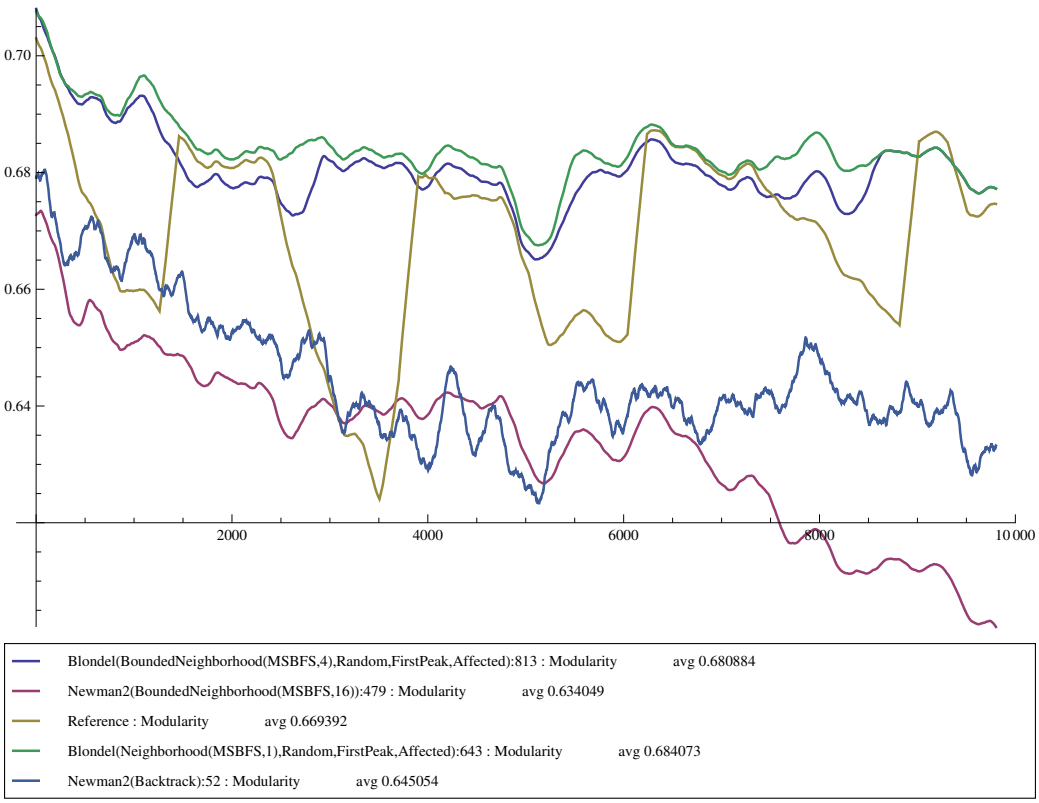


Figure 4.3.12. Dynamic algorithms adapt quickly to changes in the *ground-truth* clustering, such changes are visible by drops in the reference quality (in turn, jumps occur when the reference clustering reacts).

The quick reactivity of a dynamic algorithm is of particular importance as, clearly, static counterpart algorithms are not subject to such issues, since they "forget " their previous work. Only $\mathsf{dGlobal@BN}_{16}$ seems to need some more time to adapt to the last clustering event. This instance is a growing network with 10K *changes* of batch size 10, its few changes in the clustering are rapidly realized by a decent frequency of node insertions in ways consistent with the coming clustering. It is thus a more "difficult" instance for an algorithm to prove its reactivity; on other instances we observed even better results.

4.3.4.6 Implementation Notes.

We conducted our experiments on up to eight cores, 1 per experiment, of a dual *Intel Xeon* 5430 running *SUSE Linux* 11.1. The machine is clocked at 2.6GHz, has 32GB of RAM and 2×1MB of L2 cache. Our algorithms and measures are implemented in *Java* 1.6.0_13, partially using the *yFiles* graph library[14], and run on a 64-Bit Server VM. Evaluations, plots and the setups of experiments were conducted via a frontend programmed in *Mathematica* (version 7.0.1.0). As priority queue we use a `java.util.PriorityQueue`. As a data structure which supports *backtrack*, instead of using a rather involved fully dynamic *union-find* structure, we maintain a similar structure, a binary forest with actual nodes as leaves and the merge operations as internal tree-nodes.

[14]Licensed from *yWorks*, for more information, see `www.yworks.com`.

4.3.5 Summary of Insights

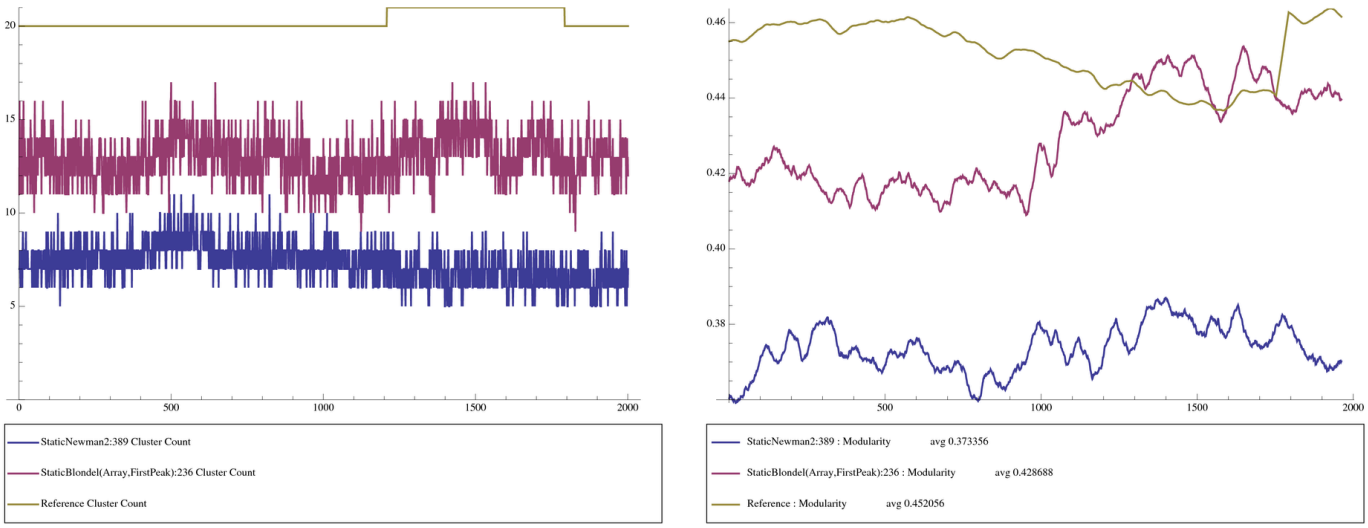
executive summary

Since the above results discuss a confusing array of degrees of freedom, we here summarize our findings. The outcomes of our evaluation are very favorable for the dynamic approach in terms of all three criteria. They are quicker, *smoother* and yield higher quality clusterings, and in addition, they are by no means sluggish, but adapt their results to *ground-truth* changes quickly without major dents in quality.

We observed that dLocal is less susceptible to an increase of the search space than dGlobal. However, our results argue strongly for the locality assumption in both cases—an increase in the search space beyond a very limited range is not justified when trading off runtime against quality. On the contrary, quality and *smoothness* may even suffer for dLocal. Consequently, N and BN strategies with a limited range are capable of producing high-quality clusterings while excelling at *smoothness*. The BT strategy for dGlobal yields competitive quality at unrivaled speed, but at the expense of *smoothness*.

For dLocal a gradual improvement of quality and *smoothness* over time is observable, which can be interpreted as an effect reminiscent of *simulated annealing*, a technique that has been shown to work well for *modularity* maximization [125]. In fact, our findings on the quality that pILP yields—and algorithm that largely impedes the escape from a local maximum—corroborate this: the combination of a *prep strategy* and a maximization heuristic surpassed pILP. In some instances we even observed a behavior that resembles an asymptotic convergence towards a "consolidated" result, e.g., in Figure 4.3.10a for quality in $\mathcal{G}_1$.

Despite the fact that the overwhelming majority of our findings can be claimed to be very general with respect to the different instances we tested, our data indicates that the best choice for an algorithm in terms of quality may still depend on the nature of the target graph. In particular we point out that while dLocal surpasses dGlobal on almost all generated graphs, dGlobal is superior on our real-world instance $\mathcal{G}_e$—independent of the batch size. We speculate that this is due to $\mathcal{G}_e$ featuring a power law degree distribution in contrast to the Erdős-Rényi-type generated instances. Note that this behavior has not been observed for the static counterparts ([38]).



(a) Cluster count: sLocal (red) yields a finer clustering than sGlobal (blue), a similar observation holds for the dynamic counterparts.

(b) Quality: sLocal (red) surpasses sGlobal (blue) on this generated graph.

Figure 4.3.13. Serving as a baseline, sLocal (**dark blue**) yields a finer clustering, with a number of clusters closer to that of the reference, and a higher *modularity* than sGlobal (**purple**).

Section 4.4

Dynamic Min-Cut Tree Clustering

Dam it!

(Beaver, Alberta, Canada)

Quality guarantees or optimal results in general are something, algorithms or objective functions for graph clustering rarely admit. In terms of those techniques which appear to be used the most in practice, this is even more true, as discussed earlier. However, the literature is very rich on rigorous properties of *cuts* in graphs, which are intextricably related to clustering, as exemplified by the measure *inter-cluster conductance* (see Section 1.2.2). Another such direct involvement of cuts motivates this section. Inspired by the work of Kannan et al. [145], Flake et al. [87] recently presented a clustering algorithm which relies on cuts. Their elegant approach employs *minimum-cut trees*, first constructed by Gomory and Hu [113], and is capable of finding a hierarchy of clusterings by virtue of an input parameter. The striking feature of graph clusterings computed by this method is that they are guaranteed to yield a certain bottleneck/cut measure—related to *conductance*—within and between clusters, tunable by the input parameter α. The authors have shown how to use the algorithm in practice, in particular in the context of citation graphs and web graphs, and that it identifies clusterings of practical relevance. An obvious "disadvantage" of their algorithm is the fact that it can neither be implemented nor understood as quickly by non-experts as, e.g., some greedy agglomerative maximization. However, this does not belittle the capabilities of the algorithm and the sheer beauty of the approach from a theorist's point of view. Moreover, in contrast to the preceeding section on quick *modularity*-driven updates of graph clusterings, we here enjoy stalwart guarantees. The question whether it is possible to dynamically update a graph clustering that obeys these guarantees, motivates this section.

In this section we give an affirmative answer to this question. We develop the first correct algorithm that efficiently and dynamically maintains a clustering for a changing graph as found by the method of Flake et al. [87], allowing arbitrary atomic changes in the graph, and keeping consecutive clusterings similar, i.e., enforcing (temporal) *smoothness* (a secondary criterion). Our algorithms build upon partially updating an *intermediate minimum-cut tree* of a graph in the spirit of Gusfield's [127] simplification of the Gomory-Hu algorithm [113]. It turns out that, with only slight modifications, our techniques can update entire *min-cut trees*. Lining the path to these results are many elegant insights into the structure of minimum-s-t-cuts in changing graphs. A small experimental evaluation of the performance of our procedures compared to the static algorithm on a real-world dynamic graph corroborates our theoretical results with a promising practical speedup factor of 10. Although theoretical runtimes do not admit an overall asymptotic speedup, due to particularly degenerate cases, our results strongly indicate that in most practical cases the complexity of our algorithms scale with $|\mathcal{C}|$ and not with n. A second question we briefly address in the back of this section asks how and whether clustering with *min-cut trees* can be sped up by means of an approximate *min-cut tree*. In fact we show how a relative approximation factor for a *min-cut tree* carries over to the quality of a clustering computed by the above method.

When I decided to turn to dynamic clustering, initially I was displeased with the idea to heuristically dynamize the work of a static heuristic. Although I later realized that there is much potential in that, as discussed in Section 4.3, I tried to find a way to cling to concrete guarantees, as a tangible criterion to adhere to during an update. Thus, the work in [87] posed a feasible starting point. However, I found that there has already been an attempt to dynamize this algorithm, by Saha and Mitra [193, 192]. It quickly became obvious to me that there was a fatal flaw in that work. With no immediate remedy suggesting itself and a simple counterexample found by Pascal Maillard, a smart student of mine at that time, I did not find the time to tend to this problem statement for months. Until, together with another smart student, Tanja Hartmann, we profoundly investigated the issues of that previous attempt and finally managed to devise a correct algorithm for a dynamic update. Major parts of this section have recently been published in [118, 117], based on joint work with Tanja Hartmann and Dorothea Wagner.

Main Results

1. We develop the first correct algorithm that efficiently maintains a dynamic clustering of a dynamic graph, which guarantees specific properties on the size of bottlenecks inside the clustering. The lemmata that constitute our results yield several new insights into the structure of the set of min-cuts of a graph. (Section 4.4.2 and 4.4.3)

2. Our algorithms allow to combine the goals of effort saving and enforcing *smoothness* between *time steps*, such that guarantees on *smoothness* can actually be stated. Although no overall improvement in the asymptotic worst-case running time can generally be stated, most cases do allow for an asymptotic speed-up. (Section 4.4.4.1)

3. The approach for this task as given by Saha and Mitra [193, 192] is wrong and simple counterexamples can be given. (Section 4.4.8)

4. Our algorithms for dynamically maintaining the clustering can be extended into algorithms which dynamically maintain full min-cut trees. (Section 4.4.3.3)

5. Suppose we have an approximate *min-cut tree*, then the approximation factor of the tree carries over reasonably to the quality guarantees. (Section 4.4.9)

Future Work. Four directions for further work on dynamics are at hand: First, properties of minimum-cuts can further be explored and exploited to lower asymptotic runtimes. In order to tighten the analysis or to get it closer to what happens in practice, an average case analysis will certainly yield better bounds, but such analyses in the context of dynamic graphs tend to be rather tenacious. Furthermore, a method to dynamically adapt the parameter α will be necessary if the nature of the graph gradually changes, e.g., its density. Finally, the more fundamental problem of dynamically updating whole *min-cut trees* probably holds more secrets than outlined in this section—we merely use it as a tool, and usually do not fully compute those trees. Going back to the static problem, an interesting question asked for in Section 4.4.9 is whether there is a good and quick method to find approximate *min-cut trees*; this would widen the range of networks this algorithm can be applied to.

minimum cut

Related Work on Cuts. Finding a *minimum cut* in a graph G is arguably one of the most fundamental problems in graph theory, and countless applications build upon it. Such a cut of minimum cardinality (or minimum weight) can efficiently be found for unweighted or non-negatively weighted graphs by means of, e.g., the algorithm of Stoer and Wagner [205] in time $O(nm + n^2 \log n)$. Finding a minimum cut in graphs with real weights is NP-hard [106].

s-t-cut

A closely related notion, an *s-t*-cut is a cut with minimum cardinality (or weight) among all cuts that separate the nodes s and t with $s, t \in G$. The famous theorem of Ford and

s-t-flow

Fulkerson [88] proved that this problem is equivalent to finding a maximum *s-t*-flow through

G, a bewildering array of algorithms have been proposed to solve this problem, one example is the *Push-Relabel* algorithm [112] which can be implemented as to run in time $O(n^2\sqrt{m})$. It is folklore that no more than $n-1$ such cuts are required to have at hand a minimum cut for all $\binom{n}{2}$ pairs of nodes in a graph. Less well known is a smart representation of these cuts in a *minimum-cut tree*. Gomory and Hu pioneered this concept [113] and showed how to compute such a representation by means of $n-1$ max-flow computations. Gusfield [127] later simplified their algorithm such that although asymptotic running times were not decreased, the new algorithm is much simpler to implement and faster in practice. The *cactus*[79] of a graph is strongly related to this idea, as it is a compact representation of all *global* minimum cuts.

cactus

4.4.1 Preliminaries and Notation

In this section we strictly reserve the term *node* (or *super-node*) for compound vertices/nodes of abstracted graphs, which may contain several basic vertices; however, we identify singleton nodes with the contained vertex without further notice. Furthermore, in this section, dynamic *changes* of G will solely concern edges; the reason for this is, that vertex insertions and deletions are trivial as long as the changed vertex is disconnected. Thus, a *changes* Δ of G always involves edge $\{b,d\}$, with $c(b,d)=\varrho$, yielding $G_\oplus$ if $\{b,d\}$ is newly inserted into G, and $G_\ominus$ if it is deleted from G. For simplicity we will not handle *changes* to the weight of an edge, since this can be done almost exactly as deletions and additions. Bridge edges in G require special treatment when deleted or inserted. However, since they are both simple to detect and to deal with, we ignore them by assuming the dynamic graph to stay connected at all times.

super-node

δ^c or δ^d

Δ involves $\{b,d\}$

$c(b,d)=\varrho$

$G_\oplus$ and $G_\ominus$

G connected

The *minimum-cut tree* $T(G)=(V,E_T,c_T)$ of G is a tree on V, such that the cheapest edge on the unique path between u and v in $T(G)$ induces a minimum-u-v-cut $\theta_{u,v}$ in G. The weight of this cheapest edge is equal to the weight of $\theta_{u,v}$. Remember that neither this edge, nor $T(G)$, need to be unique. In the following we stick to the convention that for the pair of nodes $b,d\in V$ we always call this path γ (as a set of edges). Edge $e_T=\{u,v\}$ of T induces the cut $\theta_{u,v}$ in G, sometimes denoted θ_v if the context identifies u. We sometimes identify e_T with the cut it induces in G. For details on *min-cut trees*, see the pioneering work by Gomory and Hu [113] or the simplifications by Gusfield [127].

min-cut tree T

path γ

Recall that a *contraction* of G by $N\subseteq V$ means replacing set N by a single super-node η, and leaving η adjacent to all former adjacencies u of vertices of N, with edge weight equal to the sum of all former edges between N and u. Analogously we can *contract* by a set $M\subseteq E$. We start by giving some fundamental insights, which we will rely on in the following, leaving their rather basic proofs to the reader.

contraction of G

Lemma 4.4.1 *Let $e=\{u,v\}\in E_T$ be an edge in $T(G)$.*

Consider $G_\oplus$: If $e\notin\gamma$ then e is still a min-u-v-cut with weight $c(\theta_e)$. If $e\in\gamma$ then its cut-weight is $c(\theta_e)+\varrho$, it stays a min-u-v-cut iff $\forall u$-v-cuts θ' in G that do not separate *b,d: $c(\theta')\geq c(\theta_e)+\varrho$.*

Consider $G_\ominus$: If $e\in\gamma$ then e remains a min-u-v-cut, with weight $c(\theta_e)-\varrho$. If $e\notin\gamma$ then it retains weight $c(\theta_e)$, it stays a min-u-v-cut iff $\forall u$-v-cuts θ' in G that separate *b,d: $c(\theta')\geq c(\theta_e)+\varrho$.*

when do cuts remain?

4.4.2 Theory

4.4.2.1 The Static Algorithm

Finding communities in the world wide web or in citation networks are but example applications of graph clustering techniques. In [87] Flake et al. propose and evaluate an algorithm which clusters such instances in a way that yields a certain guarantee on the quality of the clusters. The authors base their quality measure on the *expansion* of a cut $(S,\bar{S})$ due to

expansion of a cut

Kannan et al. [145]:

$$\Psi = \frac{\sum_{u \in S, v \in \bar{S}} w(u,v)}{\min\{|S|, |\bar{S}|\}} \qquad (\textit{expansion of cut } (S, \bar{S})) \tag{4.4.1}$$

The *expansion* of a graph is the minimum *expansion* over all cuts in the graph. For a clustering $\mathcal{C}$, *expansion* measures both the quality of a single cluster C, quantifying the clearest bottleneck within C, and the goodness of bottlenecks defined by cuts $(C, V \setminus C)$. Inspired by a bicriterial approach for good clusterings by Kannan et al. [145], which bases on the related measure *conductance*[15], Flake et al. [87] design a graph clustering algorithm that, given parameter α, asserts the following:[16]

quality guarantee

$$\underbrace{\frac{c(C, V \setminus C)}{|V \setminus C|}}_{\text{inter-cluster cuts}} \leq \alpha \leq \underbrace{\frac{c(P,Q)}{\min\{|P|,|Q|\}}}_{\text{intra-cluster cuts}} \quad \forall C \in \mathcal{C} \quad \forall P, Q \neq \emptyset \quad P \dot\cup Q = C \tag{4.4.2}$$

Algorithm 25: Cut-Clustering

Input: Graph $G = (V, E, c)$, α

1 $V_\alpha := V \cup \{t\}$ `// add` t
2 $E_\alpha := E \cup \{\{t, v\} \mid v \in V\}$ `// star-connect` t
3 $c_\alpha|_E := c$, $c_\alpha|_{E_\alpha \setminus E} := \alpha$ `// new edges weigh` α
4 $G_\alpha := (V_\alpha, E_\alpha, c_\alpha)$
5 $T(G_\alpha) :=$ *min-cut tree* of G_α
6 $T(G_\alpha) \leftarrow T(G_\alpha) - t$
7 $\mathcal{C}(G) \leftarrow$ components of $T(G_\alpha)$

These quality guarantees—simply called *quality* in the following—are due to special properties of *min-cut trees*, which are used by the clustering algorithm, as given in Algorithm 25 (comp. [87]). It performs the following steps: Add an artificial node t to G, and connect t to all other vertices by weight α. Then, compute a *min-cut tree* $T(G_\alpha)$ of this augmented graph. Finally, remove t and let the resulting connected components of T define the clustering. In the following, we will call the fact that a clustering can be computed by this procedure the *invariant*. For the proof that Cut-Clustering yields a clustering that obeys Equation (4.4.2), we refer the reader to [87]. Flake at al. further show how nesting properties of min cuts [105] can be used to avoid computing the whole *min-cut tree* T and try to only identify those edges of T incident to t. Their recommendation for finding these edges quickly, is to start with separating high degree nodes from t. Furthermore they show that this property yields a whole clustering hierarchy, if α is scaled. In the following we will use the definition of $G_\alpha = (V_\alpha, E_\alpha, c_\alpha)$, denoting by $G_\alpha^\ominus$ and $G_\alpha^\oplus$ the corresponding augmented *and* modified graphs. For now, however, general $G_{\oplus(\ominus)}$ are considered.

invariant

the static algorithm

A Failed Dynamic Attempt. Saha and Mitra [193] published an algorithm that aims at the same goal as our work. Unfortunately, we discovered a methodical error in this work. Roughly speaking, it seems as if the authors implicitly assume an equivalence between *quality* and the *invariant*. A full description of issues is beyond the scope of this work, but we briefly point out errors in the authors' procedures and give counter-examples in the final Subsection 4.4.8.

Saha, Mitra [193]

4.4.2.2 *Minimum-Cut Trees* and the Gomory-Hu Algorithm

We briefly describe the construction of a *min-cut tree* as proposed by Gomory and Hu [113] and simplified by Gusfield [127]. Although we will adopt ideas of the latter work, we first give Gomory and Hu's algorithm (Algorithm 26) as the foundation.

Gomory, Hu [113]
Gusfield [127]

The algorithm builds the *min-cut tree* of a graph by iteratively finding min-u-v-cuts for vertices that have not yet been separated by a previous min-cut. The *intermediate min-cut tree* $T_*(G) = (V_*, E_*, c_*)$ (or simply T_* if the context is clear) is initialized as an isolated,

$T_*(G)$

[15] *conductance* is similar to *expansion* but normalizes cuts by total incident edge weight instead of the number of vertices in a cut set.

[16] The disjoint union $A \cup B$ with $A \cap B = \emptyset$ is denoted by $A \dot\cup B$.

Algorithm 26: Gomory-Hu (Minimum-Cut Tree)

Input: Graph $G = (V, E, c)$
Output: *Min-cut tree* of G

1 Initialize $V_* := \{V\}, E_* := \emptyset$ and c_* empty and tree $T_*(G) := (V_*, E_*, c_*)$
2 **while** $\exists S \in V_*$ *with* $|S| > 1$ **do** // unfold all super-nodes
3 $\quad \{u, v\} \leftarrow$ arbitrary pair from $\binom{S}{2}$
4 $\quad$ **forall** $S_j \sim S$ *in* $T_*(G)$ **do** $N_j \leftarrow$ subtree of S with $S_j \in N_j$
5 $\quad G_S = (V_S, E_S, c_S) :=$ in G *contract* each subtree N_j to node η_j // subtree contraction
6 $\quad (U, V_S \setminus U) \leftarrow$ min-u-v-cut in G_S, weight δ, $u \in U$
7 $\quad S_u \leftarrow S \cap U$, and $S_v \leftarrow S \cap (V_S \setminus U)$ // split $S = S_u \cup S_v$
8 $\quad V_* \leftarrow (V_* \setminus \{S\}) \cup \{S_u, S_v\}$, $E_* \leftarrow E_* \cup \{\{S_u, S_v\}\}$, $c_*(S_u, S_v) \leftarrow \delta$ // do the split in $T_*(G)$
9 $\quad$ **forall** *former edges* $e_j = \{S, S_j\} \in E_*$ **do**
10 $\quad\quad$ **if** $\eta_j \in U$ **then** $e_j \leftarrow \{S_u, S_j\}$; // either reconnect S_j to S_u
11 $\quad\quad$ **else** $e_j \leftarrow \{S_v, S_j\}$; // or reconnect S_j to S_v
12 **return** $T_*(G)$

edgeless super-node containing all original nodes (line 1). Then, until no node S of T_* contains more than one vertex, a node S is *split*. To this end, nodes $S_i \neq S$ are dealt with by contracting in G whole subtrees N_j of S in T_*, connected to S via edges $\{S, S_j\}$, to single nodes η_j (line 5) before cutting, which yields G_S—a notation we will continue using in the following. The split of S into (S_u, S_v) is then defined by a min-u-v-cut in G_S (line 6). Afterwards, N_j is reconnected, again by S_j, to either S_u or S_v depending on which side of the cut η_j, containing S_j, ended up. It is crucial to note, that this cut in G_S can be proven to induce a min-u-v-cut in G.

N_j and η_j
G_S
(S_u, S_v)

An *execution* $\mathrm{GH} = (G, F, K)$ of Gomory-Hu is characterized by graph G, sequence F of $n-1$ *step pairs* (compare to line 3) of nodes and sequence K of *split cuts* (compare to line 6). Pair $\{u, v\} \subseteq V$ is a *cut pair* of edge e of cut-tree T if θ_e is a min-u-v-cut in G.

GH

Theorem 4.4.1 *Consider a set $M \subseteq E_T$ and let $T_\circ(G) = (V_\circ, M, c_\circ)$ be $T(G)$ with $E_T \setminus M$ contracted. Let f and f' be sequences of the elements of M and $E_T \setminus M$, respectively, and k and k' the corresponding sequences of edge-induced cuts of G. The Gomory-Hu execution $\mathrm{GH} = (G, f' \circ f, k' \circ k)$[17] has $T_\circ(G)$ as intermediate min-cut tree (namely after f).*

For any $T_\circ$ there is some GH

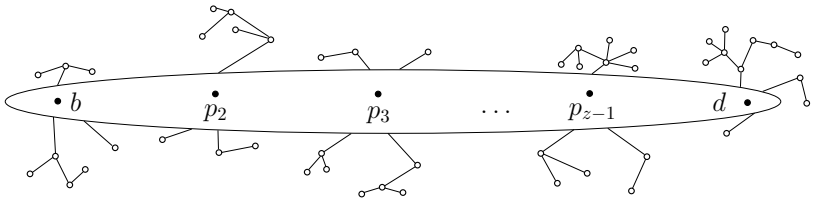
In the following we will denote by $T_\circ$ an *intermediate min-cut tree* which serves as a starting point, and by T_* a working version. We prove Theorem 4.4.1 by induction on the edges in $f' \circ f$, however, for the sake of brevity we move the full proof to Subsection 4.4.6. This theorem states that if for some reason we can only be sure about a subset of the edges of a *min-cut tree*, we can *contract* all other edges to super-nodes and consider the resulting tree $T_\circ$ as the correct *intermediate* result of some GH, which can then be continued. One such reason could be a dynamic *change* in G, such as the insertion or the deletion of an edge, which by Lemma 4.4.1 maintains a subset of the old min-cuts. Thus we could already design an effort-saving algorithm for dynamically updating *min-cut trees*: *contract* all those edges of $T(G)$ which might not be valid any more, yielding $T_\circ(G_{\oplus(\ominus)})$, as depicted in Figure 4.4.1, and start a run of Gomory-Hu with this *intermediate min-cut tree*.

a first algorithm

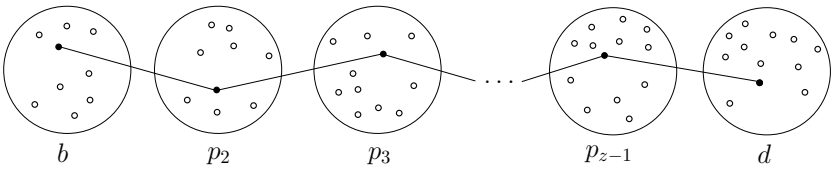
4.4.2.3 Using Arbitrary Minimum Cuts in G

Gusfield [127] presented an algorithm for finding *min-cut trees* which avoids complicated *contraction* operations. In essence he provided rules for adjusting iteratively found min-u-v-cuts in G (instead of in G_S) that potentially cross, such that they are consistent with

[17] The term $b \circ a$ denotes the concatenation of sequences b and a, i.e., a happens first.



(a) $T_\circ$ by contracting all edges of γ in $T(G)$



(b) $T_\circ$ by contracting all edges of $E_T \setminus \gamma$

Figure 4.4.1. Sketches of *intermediate min-cut trees* $T_\circ$; for $G_\oplus$ (a) we *contract* γ to a node, and for $G_\ominus$ (b) we *contract* each connected component induced by $E_T \setminus \gamma$, yielding a path of nodes.

the Gomory-Hu procedure and thus non-crossing, but still minimal. We need to review and generalize some of these ideas as to fit our setting. The following lemma essentially tells us, that at any time in Gomory-Hu, for any edge e of $T_\circ$ there exists a *cut pair* of e in the two nodes incident to e.

cut pairs stay cut pairs

new cut pairs

Lemma 4.4.2 (Gus. [127], Lemma 4[18]) *Let S be cut into S_x and S_y, with $\{x, y\}$ being a cut pair (not necessarily the step pair). Let now $\{u, v\} \subseteq S_x$ split S_x into S_{xu} and S_{xv}, wlog. with $S_y \sim S_{xu}$ in T_*. Then, $\{x, y\}$ remains a cut pair of edge $\{S_y, S_{xu}\}$ (we say edge $\{S_x, S_y\}$ gets* reconnected*). If $x \in S_{xv}$, i.e., the min-u-v-cut separates x and y, then $\{u, y\}$ is also a cut pair of $\{S_{xu}, S_y\}$.*

hidden shadowed

In the latter case of Lemma 4.4.2, we say that pair $\{x, y\}$ gets *hidden*, and, in the view of vertex y, its former counterpart x gets *shadowed* by u (or by S_u). It is not hard to see that during Gomory-Hu, *step pairs* remain *cut pairs*, but *cut pairs* need not stem from *step pairs*. However, each edge in T has at least one *cut pair* in the incident nodes. We define the *nearest cut pair* of an edge in T_* as follows: As long as a *step pair* $\{x, y\}$ is in adjacent nodes S_x, S_y, it is the *nearest cut pair* of edge $\{S_x, S_y\}$; if a *nearest cut pair* gets hidden in T_* by a step of Gomory-Hu, as described in Lemma 4.4.2 if $x \in S_{xv}$, the *nearest cut pair* of the reconnected edge $\{S_y, S_{xu}\}$ becomes $\{u, y\}$ (which are in the adjacent nodes S_y, S_{xu}). The following theorem basically states how to iteratively find min-cuts as Gomory-Hu, without the necessity to operate on a *contracted* graph.

nearest cut pair

we can *avoid contraction*

Theorem 4.4.2 (Gus. [127], Theorem 2[5]) *Let $\{u, v\}$ denote the current step pair in node S during some* GH*. If $(U, V \setminus U)$, $(u \in U)$ is a min-u-v-cut in G, then there exists a min-u-v-cut $(U_S, V_S \setminus U_S)$ of equal weight in G_S such that $S \cap U = S \cap U_S$ and $S \cap (V \setminus U) = S \cap (V_S \setminus U_S)$, $(u \in U_S)$.*

Being an ingredient to the original proof of Theorem 4.4.2, the following Lemma 4.4.3 gives a constructive assertion, that tells us how to arrive at a cut described in the theorem by inductively adjusting a given min-u-v-cut in G. Thus, it is the key to avoiding *contraction* and using cuts in G by rendering min-u-v-cuts non-crossing with other given cuts.

how to *avoid contraction*

Lemma 4.4.3 (Gus. [127], Lemma 1[5]) *Let $(Y, V \setminus Y)$ be a min-x-y-cut in G $(y \in Y)$. Let $(H, V \setminus H)$ be a min-u-v-cut, with $u, v \in V \setminus Y$ and $y \in H$. Then the cut $(Y \cup H, (V \setminus Y) \cap (V \setminus H))$ is also a min-u-v-cut.*

Given a cut as by Theorem 4.4.2, Gomory and Hu state a simple mechanism which reconnects a former neighboring subtree N_j of a node S to either of its two split parts (lines 9-11 in Algorithm 26), by the cut side on which the *contraction* η_j of N_j ends up. In contrast, to establish reconnection when avoiding *contraction*, this criterion is not available, as N_j is not handled en-block. For this purpose, Gusfield iteratively defines *representatives* $r(S_i) \in V$ of nodes S_i of T_*. Starting with an arbitrary vertex as $r(\{V\})$, *step pairs* in S_i must then always include $r(S_i)$, with the second vertex becoming the representative of the newly split off node S_j. For a suchlike run of Gomory-Hu, Gusfield shows (using Lemma 4.4.2 iteratively) that for two adjacent nodes S_u, S_v in any $T_\circ$, $r(S_u), r(S_v)$ is a *cut pair* of edge $\{S_u, S_v\}$, and, most importantly his Theorem 3: For $u, v \in S$ let *any* min-u-v-cut $(U, V \setminus U)$, $u \in U$, in G split node S into $S_u \ni u$ and $S_v \ni v$ and let $(U_S, V \setminus U_S)$ be this cut adjusted via Lemma 4.4.3 and Theorem 4.4.2; then a neighboring subtree N_j of S, formerly connected by edge $\{S, S_j\}$, lies in U_S iff $r(S_j) \in U$. Since we intend to work with arbitrary *intermediate min-cut trees* as in Theorem 4.4.1, we do not have representatives and thus need to adapt Gusfield's Theorem 3, namely using *nearest cut pairs* as representatives, in order to finally enable a simplified construction of *min-cut trees*. The proof of the following theorem can be found in Subsection 4.4.6.

representative

Theorem 4.4.3 (comp. Gus. [127], Theorem 3[5]) *In any T_* of a* GH*, suppose $\{u, v\} \subseteq S$ is the next step pair, with subtrees N_j of S connected by $\{S, S_j\}$ and nearest cut pairs $\{x_j, y_j\}$, $y_j \in S_j$. Let $(U, V \setminus U)$ be a min-u-v-cut in G, and $(U_S, V \setminus U_S)$ its adjustion. Then $\eta_j \in U_S$ iff $y_j \in U$.*

how to reconnect subtrees

4.4.2.4 Finding and Shaping Minimum Cuts in the Dynamic Scenario

In this section we let graph G *change*, i.e., we consider the addition of an edge $\{b, d\}$ or its deletion, yielding $G_\oplus$ or $G_\ominus$. First of all we define valid representatives of the nodes on $T_\circ$. By Lemma 4.4.1 and Theorem 4.4.1, given an edge addition, $T_\circ$ consists of a single super-node and many singletons, and given edge deletion, $T_\circ$ consists of a path of super-nodes; for examples see Figure 4.4.1.

Definition 4.2 (Representatives in $T_\circ$)
(i) Edge addition: *Set singletons to be representatives of themselves; for the only super-node S choose an arbitrary $r(S) \in S$.*
(ii) Edge deletion: *For each node S_i, set $r(S_i)$ to be the unique vertex in S_i which lies on γ in $T(G)$.*
(iii) New nodes during algorithm, and the choice of *step pairs*: *On a split of node S during the algorithm, require the step pair to be $\{r(S), v\}$ with an arbitrary $v \in S, v \neq r(S)$. Let the split be $S = S_{r(S)} \cup S_v, v \in S_v$, then define $r(S_{r(S)}) := r(S)$ and $r(S_v) := v$.*

how to define representatives

Consider edge additions; singletons in $T_\circ$ trivially are their own representatives. Since no singleton gets split, the single super-node S gets split first, and thus only needs representatives for its parts thereafter, which are defined by the *step pair*, see below. With edge deletions, according to Lemma 4.4.1 each node of $T_\circ$ contains a vertex that lies on γ in the old $T(G)$, with the edges connecting these vertices being correct min-cuts in $G_\ominus$(see Figure 4.4.1b), they thus are *nearest cut pairs*. By Lemma 4.4.2 the representatives of new nodes as defined above always define *nearest cut pairs*. Thus, in the case of edge additions, choosing an arbitrary *step pair* in S at the start is feasible.

Following Theorem 4.4.1, we define the set M of "good" edges of the old tree $T(G)$, i.e., edges that stay valid due to Lemma 4.4.1, as $M := E_T \setminus \gamma$ for the insertion of $\{b, d\}$ and to $M := \gamma$ for the deletion. Let the *intermediate* cut-tree $T_\circ(G_{\oplus(\ominus)})$ be $T(G)$ *contracted* by M. As above, let f be any sequence of the edges in M and k the corresponding cuts in G.

Lemma 4.4.4 *Given an edge addition (deletion) in G. The Gomory-Hu execution* $\mathrm{GH}_{\oplus(\ominus)} = (G_{\oplus(\ominus)}, f_{\oplus(\ominus)} \circ f, k_{\oplus(\ominus)} \circ k)$ *is feasible for $G_{\oplus(\ominus)}$ yielding $T_\circ(G)$ as the intermediate min-cut*

$\mathrm{GH}_{\oplus(\ominus)}$

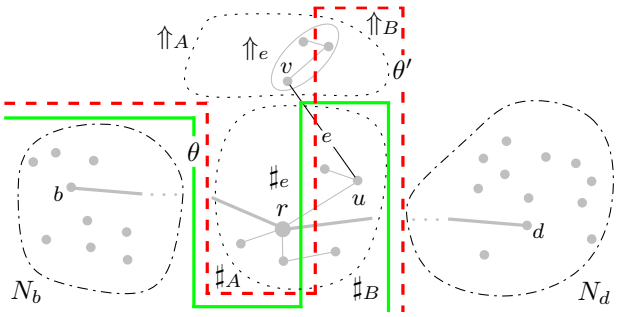


Figure 4.4.2. Special parts of $G_\ominus$: γ (fat) connects b and d, with r on it; wood $\sharp_e$ and treetop $\Uparrow_e$ (dotted) of edge e, both cut by θ' (dashed), adjusted to θ (solid) by Lemma 4.4.9. Both $\sharp_e$ and $\Uparrow_e$ are part of some node S, with representative r, outside subtrees of r are N_b and N_d (dash-dotted). Compare to Figure 4.4.1b.

tree after sequence f, if $f_{\oplus(\ominus)}$ and $k_{\oplus(\ominus)}$ are feasible sequences of step pairs and cuts on $T_\circ(G_{\oplus(\ominus)})$.

As Lemma 4.4.4 describes a specific variant of the setting in Theorem 4.4.1, it also relies on induction on the *split cuts* in k, see Subsection 4.4.6 for its proof. It is the basis of our updating algorithms, founded on $T_\circ$'s as in Figure 4.4.1, using arbitrary cuts in $G_{\oplus(\ominus)}$ instead of actual *contractions*. Still, the non-crossing nature of min-u-v-cuts allows for more effort-saving and temporal *smoothness*.

Treetop and Wood $\Uparrow_e$ and $\sharp_e$

Definition 4.3 (Treetop and Wood) *Consider edge $e = \{u, v\}$ off γ, and cut $\theta = (U, V \setminus U)$ in G induced by e in $T(G)$ with γ contained in U. In the contracted graph $G_\ominus(S)$, $S \cap (V \setminus U)$ is called the* treetop $\Uparrow_e$*, and $S \cap U$ the* wood $\sharp_e$ *of e. The subtrees of S are N_b and N_d, containing b and d, respectively (see Figure 4.4.2 for an example).*

Cuts That Can Stay. There are several circumstances which imply that a previous cut is still valid after a graph modification, making its recomputation unnecessary. The following three lemmas all give such assertions. Their proofs mostly rely on properties of Gomory-Hu-executions and on Lemma 4.4.1, they can be found in Subsection 4.4.6.

$e_{\min}$ on γ still minimal in $G_\oplus$

Lemma 4.4.5 *Suppose $e_{\min}$ is the cheapest edge on γ. In $G_\oplus$, $e_{\min}$ still induces a min-b-d-cut.*

entire treetops can be reconfirmed

Lemma 4.4.6 *In $G_\ominus$, let $(U, V \setminus U)$ be a min-u-v-cut not separating $\{b, d\}$, with γ in $V \setminus U$. Then, a cut induced by an edge $\{g, h\}$ of the old $T(G)$, with $g, h \in U$, remains a min separating cut for all its previous cut pairs within U in $G_\ominus$, and a min-g-h-cut in particular.*

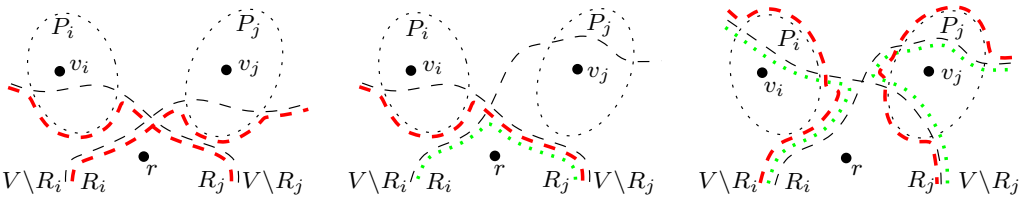
cheap treetop-edges remain in $G_\ominus$

Lemma 4.4.7 *Assume $g \in V$ on γ and $\{y_b, g\}, \{y_d, g\} \in \gamma$, and let wlog. $c(\{y_b, g\}) \leq c(\{y_d, g\})$. Let further $\{u, v\}$ be an edge within $\Uparrow_{\{g,h\}}$ (or $\{g, h\}$ itself) in $T(G)$. If $c_T(\{u, v\}) \leq c_T(\{y_b, g\}) - \varrho$ in the old tree, then, in $G_\ominus$, $\{u, v\}$ also induces a min-u-v-cut.*

As a corollary from Lemma 4.4.6 we get that in $T(G_\ominus)$ the entire treetops of reconfirmed edges of $T(G)$ are also reconfirmed. Cuts that can be retained save effort and encourage *smoothness*; however new cuts can also be urged to behave well, as follows.

The Shape of New Cuts. In contrast to the above lemmas, during a Gomory-Hu execution for $G_\ominus$, we might find an edge $\{u, v\}$ of the old $T(G)$ that is *not* reconfirmed by a computation in $G_\ominus$, but a new, cheaper min-u-v-cut $\theta' = (U, V(S) \setminus U)$ is found. For such a new cut we can still make some guarantees on its shape to resemble its "predecessor": Lemmas 4.4.8 and 4.4.9 tell us, that for any such min-u-v-cut θ', there is a min-u-v-cut $\theta = (U \setminus \Uparrow_e, (V(S) \setminus U) \cup \Uparrow_e)$ in

how to largely preserve old cuts



(a) θ'_i separates v_j, r, and θ'_j separates v_i, r

(b) θ'_i does not separate v_j, r, but θ'_j separates v_i, r

(c) neither does θ'_i separate v_j, r, nor θ'_j v_i, r

Figure 4.4.3. Three different cases concerning the positions of θ'_i and θ'_j (black, dashed), and their adjustments.

$G_\ominus$ that (a) does not split $\Uparrow_e$, (b) but splits $V \setminus \Uparrow_e$ exactly as θ' does. Figure 4.4.2 illustrates such cuts θ (solid) and θ' (dashed).

Lemma 4.4.8 *Given $e = \{u, v\}$ within S (off γ) in $G_\ominus(S)$. Let $(\Uparrow_A, \Uparrow_B)$ be a cut of $\Uparrow_e$ with $v \in \Uparrow_A$. Then $c_\ominus(N_b \cup \Uparrow_e, N_d \cup \#_e) \leq c_\ominus(N_b \cup \Uparrow_A, N_d \cup \#_e \cup \Uparrow_B)$. Exchanging N_b and N_d is analogous.* *don't cut treetops!*

Lemma 4.4.9 *Lemma 4.4.8 can be generalized in that both considered cuts also cut the wood $\#_e$ in some arbitrary but fixed way.* *wood not affected*

The proof of the above lemmas is rather technical, but conceptually it relies on the fact that if a cut which splits the treetop were cheaper, then this treetop cannot have been valid in the previous tree. While theses lemmas can be applied in order to retain treetops, even if new cuts are found, in the following, we take a look at how new, cheap cuts can affect the treetops of *other* edges. In fact a similar treetop-conserving result can be stated.

Let G' denote an undirected, weighted graph and $\{r, v_1, \ldots, v_z\}$ a set of designated vertices in G'. Let $\Pi := \{P_1, \ldots, P_z\}$ be a partition of $V \setminus r$ such that $v_j \in P_j$. We now assume the following *partition-property* to hold: For each v_j it holds that for any v_j-r-cut $\theta'_j := (R_j, V \setminus R_j)$ (with $r \in R_j$), the cut $\theta_j := (R_j \setminus P_j, (V \setminus R_j) \cup P_j)$ is of at most the same weight. The crucial observation is, that Lemma 4.4.9 implies this *partition-property* for $r(S)$ and its neighbors in $T(G)$ that lie inside S of $T_\circ$ in $G_\ominus$. Treetops thus are the sets P_j. However, we keep things general for now. *partition-property*

Consider a min-v_i-r-cut $\theta'_i := (R_i, V \setminus R_i)$, with $r \in R_i$, that does not split P_i and an analog min-v_j-r-cut θ'_j (by the *partition-property* they exist). We distinguish three cases, given in Figure 4.4.3, which yield the following possibilities of reshaping min-cuts: *reshaping min-cuts*

Case (a): As cut θ'_i separates v_j and r, and as v_j satisfies the *partition-property*, the cut $\theta_i := (R_i \setminus P_j, (V \setminus R_i) \cup P_j)$ (red dashed) has weight $c(\theta_i) \leq c(\theta'_i)$ and is thus a min-v_i-r-cut, which does not split $P_i \cup P_j$. For θ'_j an analogy holds.

Case (b): For θ'_j Case (a) applies. Furthermore, by Lemma 4.4.3 the cut $\theta_{\text{new}(j)} := (R_i \cap R_j, (V \setminus R_i) \cup (V \setminus R_j))$ (green dotted) is a min-v_j-r-cut, which does not split $P_i \cup P_j$. By Lemma 4.4.2 the previous *split cut* θ'_i turns out to be also a min-v_i-v_j-cut, as $\theta_{\text{new}(j)}$ separates v_i and r.

Case (c): As in case (b), by Lemma 4.4.3 the cut $\theta_{\text{new}(i)} := ((V \setminus R_j) \cup R_i, (V \setminus R_i) \cap R_j)$ (green dotted) is a min-v_i-r-cut, and the cut $\theta_{\text{new}(j)} := ((V \setminus R_i) \cup R_j, (V \setminus R_j) \cap R_i)$ (green dotted) is a min-v_j-r-cut. These cuts do not cross. So as v_i and v_j both satisfy the *partition-property*, cut $\theta_i := (((V \setminus R_j) \cup R_i) \setminus P_i, ((V \setminus R_i) \cap R_j) \cup P_i)$ and $\theta_j := (((V \setminus R_i) \cup R_j) \setminus P_j, ((V \setminus R_j) \cap R_i) \cup P_j)$ (both red dashed) are non-crossing min separating cuts, which neither split P_i nor P_j.

To summarize the cases discussed above, we make the following observation.

Observation 4.4.1 *During a* GH *starting from $T_\circ$ for $G_\ominus$, whenever we discover a new, cheaper min-v_i-$r(S)$-cut θ' ($v_i \sim r(S)$ in node S) we can iteratively reshape θ' into a min-* *reshape: summary*

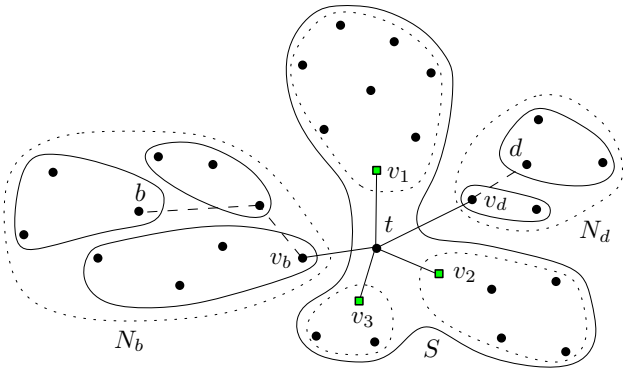


Figure 4.4.4. $T_\circ(G_\alpha^\ominus)$ for an inter-cluster deletion, t's neighbors off γ need inspection. The cuts of v_b and v_d are correct, but they might get *shadowed*.

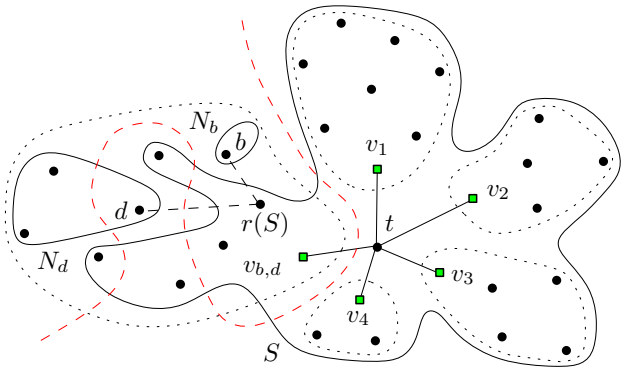


Figure 4.4.5. $T_\circ(G_\alpha^\ominus)$ for an intra-cluster deletion, edge $\{v_{b,d}, t\}$ defines a treetop (t's side). The dashed cut could be added to Θ by Algorithm 29 (line 9).

v_i-$r(S)$-cut θ which neither cuts $\Uparrow_i$ nor any other treetop $\Uparrow_j$ ($v_i \sim r(S)$ in S), by means of Cases (a,b,c).

4.4.3 Update Algorithms for Dynamic Clusterings

we only need to isolate t

In this section we put the results of the previous sections to good use and give algorithms for updating a *min-cut tree* clustering, such that the *invariant* is maintained and thus also the *quality*. It is important to see that it is not necessary to maintain a full *min-cut tree* to determine the induced clustering. By concept, we merely need to know all vertices of $T(G)$ adjacent to t; we call this set $W = \{v_1, \ldots, v_z\} \cup \{v_b, v_d\}$, with $\{v_b, v_d\}$ being the particular vertex/vertices on the path from t to b and d, respectively. We call the corresponding set of non-crossing min-v_i-t-cuts that isolate t, Θ. We will thus focus on dynamically maintaining only this information, and sketch out how to unfold the rest of the *min-cut tree*. From Lemma 4.4.4, for a given edge insertion or deletion, we know $T_\circ$, and we know in which node of $T_\circ$ to find t, this is the node we need to examine. We now give algorithms for the deletion and the insertion of an edge running inside or between clusters.

Algorithm 27: Inter-Cluster Edge Deletion

Input: $W(G)$, $\Theta(G)$ $G_\alpha^\ominus = (V_\alpha, E_\alpha \setminus \{\{b,d\}\}, c_\alpha^\ominus)$, edge $\{b,d\}$ with weight ϱ
Output: $W(G_\ominus)$, $\Theta(G_\ominus)$

```
1 L(t) ← ∅, l(t) ← ∅
2 for i = 1, ..., z do
3 |   Add v_i to L(t)                          // old cut-vertices
4 |   D(v_i) ← ∅                                // shadows
5 Θ(G_⊖) ← {θ_b, θ_d} , W(G_⊖) ← {v_b, v_d}
6 return Check Cut-Vertices ( W(G), Θ(G), W(G_⊖), Θ(G_⊖), G_α^⊖, {b,d}, D, L(t) )
```

4.4.3.1 Edge Deletion

inter-cluster del.

Inter-Cluster Edge-Deletion. Our first algorithm handles inter-cluster deletion (Algorithm 27). Just like its three counterparts, it takes as an input the old graph G and its sets $W(G)$ and $\Theta(G)$ (not the entire *min-cut tree* $T(G_\alpha)$), furthermore it takes the changed graph, augmented by t, $G_\alpha^\ominus$, the deleted edge $\{b,d\}$ and its weight ϱ. Recall that an inter-cluster

Algorithm 28: Check Cut-Vertices

Input: $W(G), \Theta(G), W(G_\ominus), \Theta(G_\ominus), G_\alpha^\ominus, \{b, d\}, D, L(t)$
Output: $W(G_\ominus), \Theta(G_\ominus)$

```
1  while L(t) has next element v_i do
2      θ_i ← first min-v_i-t-cut given by FlowAlgo(v_i, t)        // small side for v_i
3      if c_α^⊖(θ_i) = c_α(θ_i^old) then                          // retain old cuts of the same weight
4          Add θ_i^old to l(t)                                    // pointed at by v_i
5      else                                                       // new cheaper cuts
6          Add θ_i to l(t)                                        // pointed at by v_i
7          while L(t) has next element v_j ≠ v_i do               // test vs. other new cuts
8              if θ_i separates v_j and t then                    // v_j shadowed by Lemma 4.4.3
9                  Move v_j from L(t) to D(v_i)
10                 if l(t) ∋ θ_j, pointed at by v_j then Delete θ_j from l(t)
11 while L(t) has next element v_i do                             // make new cuts cluster-preserving
12     set (R, V_α \ R) := θ_i with t ∈ R for θ_i ∈ l(t) pointed at by v_i   // just nomenclature
13     θ_i ← (R \ C_i, (V_α \ R) ∪ C_i)                           // by partition-property (Lemma 4.4.9)
14     forall v_j ∈ D(v_i) do                                     // handle shadowed cuts ...
15         θ_i ← (R \ C_j, (V_α \ R) ∪ C_j)                       // ...with Cases (a) and (b)
16     forall v_j ≠ v_i in L(t) do                                // handle other cuts ...
17         θ_i ← (R ∪ C_j, (V_α \ R) \ C_j)                       // ...with Case (c)
18 Add all vertices in L(t) to W(G_⊖), and their cuts from l(t) to Θ(G_⊖)
19 return W(G_⊖), Θ(G_⊖)
```

deletion yields t on γ, and thus, $T_\circ(G_\alpha)$ contains edges $\{v_b, t\}$ and $\{v_d, t\}$ cutting off the subtrees N_b and N_d of t by cuts θ_b, θ_d, as shown in Figure 4.4.4. All clusters contained in node $S \ni t$ need to be changed or reconfirmed. To this end Algorithm 27 lists all cut vertices in S, $v_1, \ldots, v_z$, into $L(t)$, and initializes their *shadows* $D(v_i) = \emptyset$. The known cuts θ_b, θ_d are already added to the final list, as are v_b, v_d (line 5). Then the core algorithm, Check Cut-Vertices is called, which—roughly speaking—performs those GH-steps that are necessary to isolate t, of course, using (most of) the lemmas derived above.

the workhorse: Check Cut-Vertices

First of all, note that if $|\mathcal{C}| = 2$ ($\mathcal{C} = \{N_b, N_d\}$ and $S = \{t\}$) then $L(t) = \emptyset$ and Algorithm 27 lets Check Cut-Vertices (Algorithm 28) simply return the input cuts and terminates. Otherwise, it iterates the set of former cut-vertices $L(t)$ once, thereby possibly shortening it. We start by computing a new min-v_i-t-cut for v_i. We do this with a max-v_i-t-flow computation, which is known to yield *all* min-v_i-t-cuts [186], taking the *first* cut found by a breadth-first search from v_i (lines 2). This way we find a cut which minimally interferes with other treetops, thus encouraging temporal *smoothness*. If the new cut is non-cheaper, we use the old one instead, and add it to the tentative list of cuts $l(t)$ (lines 3-4). Otherwise we store the new, cheaper cut θ_i, and examine it for later adjustment. For any candidate v_j still in $L(t)$ that is separated from t by θ_i, Case (a) or (b) applies (line 8). Thus, v_j will be in the *shadow* of v_i, and not a cut-vertex (line 9). In case v_j has already been processed, its cut is removed from $l(t)$.

the first cut

collecting shadows

Once all cut-vertex candidates are processed, each one either induces the same cut as before, is new and *shadows* other former cut-vertices or is itself *shadowed* by another cut-vertex. Now that we have collected these relations, we actually apply Cases (a,b,c) and Lemma 4.4.9 in lines 11-17. Note that for retained, old cuts, no adjustment is actually performed here. Finally, all non-*shadowed* cut-vertices alongside their adjusted cuts are added to the final lists, and those returned.

apply Cases by shadows

Algorithm 29: Intra-Cluster Edge Deletion

Input: $W(G), \Theta(G), G_\alpha^\ominus = (V_\alpha, E_\alpha \setminus \{\{b,d\}\}, c_\alpha^\ominus)$, edge $\{b,d\}$ with weight ϱ
Output: $W(G_\ominus)$, $\Theta(G_\ominus)$ regarding $G_\ominus$

1 $\theta_{b,d} \leftarrow$ first min-t-$v_{b,d}$-cut given by FlowAlgo$(t, v_{b,d})$ // small side for t
2 **if** $c_\alpha^\ominus(\theta_{b,d}) = c_\alpha(\theta_{b,d}^{old})$ **then** // no cheaper cut found
3 **return** $W(G), \Theta(G)$ // retain clustering
4 **else** // a new cut should retain treetops
5 set $(R, V_\alpha \setminus R) := \theta_{b,d}$ with $t \in R$ // just nomenclature
6 **forall** $C_i \neq C_{b,d}$ **do** // by Lemma 4.4.9
7 $\theta_{b,d} := (R \cup C_i, (V_\alpha \setminus R) \setminus C_i)$
8 $L(t) \leftarrow \emptyset, l(t) \leftarrow \emptyset$
9 $\Theta(G_\ominus) \leftarrow \{\theta_{b,d}\}, W(G_\ominus) \leftarrow \{v_{b,d}\}$
10 **for** $i = 1, \ldots, z$ **do** // not including $v_{b,d}$
11 Add v_i to $L(t)$
12 $D(v_i) \leftarrow \emptyset$
13 $W(G_\ominus), \Theta(G_\ominus) \leftarrow$ Check Cut-Vertices $(W(G), \Theta(G), W(G_\ominus), \Theta(G_\ominus), G_\alpha^\ominus, \{b,d\}, D, L(t))$
14 $W(G_\ominus) \leftarrow W(G_\ominus) \cup v_{b,d}$, $\Theta \leftarrow \Theta \cup \{\theta_{b,d}\}$
15 Resolve all crossings in $\Theta(G_\ominus)$ by Lemma 4.4.3
16 Isolate the sink t from all remaining unclustered vertices
17 **return** $W(G_\ominus), \Theta(G_\ominus)$

intra-cluster del.

Intra-Cluster Edge-Deletion. Next we look at *intra*-cluster edge deletion. Looking at our starting point $T_\circ$, the safe path γ lies within some cluster $C_{b,d}$, which does not help much. In this case, t lies off γ, and thus there is an edge $\{v_{b,d}, t\}$, with $v_{b,d} \in C_{b,d}$, which defines a treetop containing all other former clusters and t, see Figure 4.4.5. Algorithm 29 has the same in- and output as Algorithm 27, and starts by finding a new *first* min-t-$v_{b,d}$-cut. If this yields that no new, cheaper t-$v_{b,d}$-cut exists, then, by Lemma 4.4.6, we are done (line 2). Otherwise, we can at least adjust $\theta_{b,d}$ such that it does not interfere with any former cluster C_i by Lemma 4.4.9, as C_i is part of a treetop (lines 5-7); note that $C_{b,d}$ can not necessarily be preserved. Then we prepare the sets $L(t), l(t), \Theta(G_\ominus), W(G_\ominus)$ in lines 8-12. Check Cut-Vertices now performs the same tasks as for Inter-Cluster Edge Deletion: it separates all cut-vertex candidates from t in a non-intrusive manner; note that this excludes $v_{b,d}$ (line 10), as $C_{b,d}$ is no treetop, and thus defies the adjustments. After line 13 we have one min-$v_{b,d}$-t-cut that leaves its treetop untouched, but might cut $C_{b,d}$, and a new set $\Theta(G_\ominus)$ of non-crossing min-v_i-t-cuts (with some former $v_j \in W(G)$ possibly having become *shadowed*), which might, however, also cut through $C_{b,d}$. Putting all these cuts and cut-certices into $\Theta(G_\ominus)$ and $W(G_\ominus)$, we can now apply Lemma 4.4.3 (using t as "x"), to make all cuts non-crossing. Note that this can also result in *shadowing* $v_{b,d}$ as in Case (b) (dotted cut). Finally, some vertices from the former cluster $C_{b,d}$ might then still remain unclustered, i.e., not separated from t by any $\theta \in \Theta(G_\ominus)$. For clustering these vertices v we cannot do better than proceeding as usual: compute their set of min-v-t-cuts and render them non-crossing by Lemma 4.4.3, possibly *shadowing* one another or some previous cut θ. We refrain from detailing the latter steps.

4.4.3.2 Edge Addition

intra-cluster add.

The good news for handling $G_\oplus$ is, that an algorithm Intra-Cluster Edge Addition need not do anything, but return the old clustering: By Lemma 4.4.1 and Theorem 4.4.1, in $T_\circ$, only path γ is *contracted*. But since γ lies within a cluster, the cuts in G_α, defining the old clustering, all remain valid in $G_\alpha^\oplus$, as depicted in Figure 4.4.7 with dotted clusters and affected node S.

inter-cluster add.

By contrast, adding an edge between clusters is more demanding. Again, γ is *contracted*, see region S in Figure 4.4.6; however, t lies on γ in this case. A sketch of what needs to

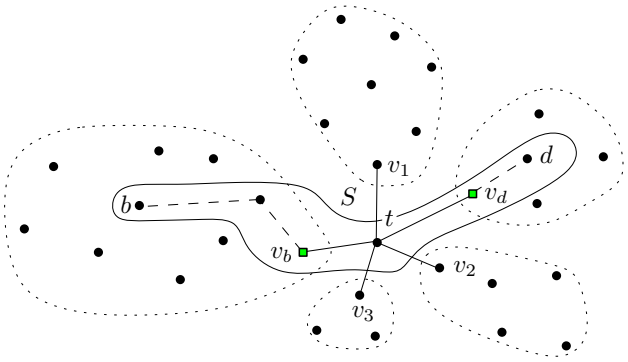


Figure 4.4.6. $T_\circ(G_\alpha^\oplus)$ for an inter-cluster addition. At least v_b and v_d need inspection.

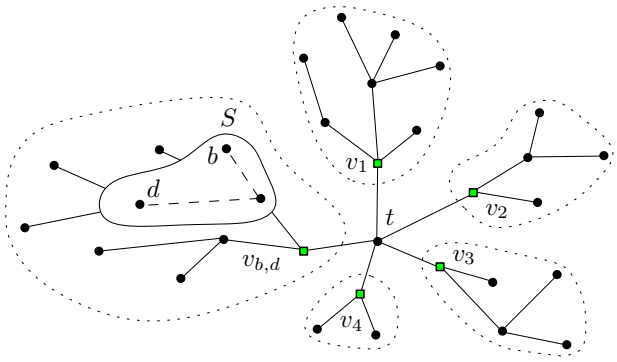


Figure 4.4.7. $T_\circ(G_\alpha^\oplus)$ for an intra-cluster addition. All relevant min-v-t-cuts persist.

be done resembles the above algorithms: We compute new min-v_b-t- and min-v_d-t-cuts (or possibly only one, if it immediately *shadows* the other in line 12, in Algorithm 30), and keep the old v_i-t-cuts. Then—proceeding as usual—we note which cuts *shadow* which others and reconnect nodes by Theorem 4.4.3. Similar to Algorithm 29, the two new cuts may leave a "wild" set of vertices from the previous subtrees N_b, N_d, where crossings still have to be removed (via Lemma 4.4.3) in the end, and leftover vertices must be separated from t from scratch. We leave the pseudo-code to Subsection 4.4.7.

4.4.3.3 Updating Entire *Min-Cut Trees*

outlook: towards whole min-cut trees

An interesting topic on its own right and more fundamental than clustering, is the dynamic maintenance of *min-cut trees*. In fact the above clustering algorithms are surprisingly close to methods that update *min-cut trees*. Since all the results from Section 4.4.2 still apply, we only need to unfold whatever treetops or subtrees of t—which we gladly accept as super-nodes for the purpose of clustering—and take care to correctly reconnect subtrees. This includes, that merely examining the neighbors of t does not suffice, we must iterate through all nodes S_i of $T_\circ$. For the sake of brevity we must omit further details on such algorithms and refer the interested reader to [129].

4.4.4 Performance of the Algorithm

4.4.4.1 Temporal Smoothness

no optimal shadowing?

Our secondary criterion—which we left unformalized—to preserve as much of the previous clustering as possible, in parts synergizes with effort-saving, an observation foremost reflected in the usage of $T_\circ$. Lemmas 4.4.6 and 4.4.9, using *first cuts* and Observation 4.4.1 nicely enforce temporal *smoothness*. However, in some cases we must cut back on this issue, e.g., when we examine which other cut-vertex candidates are *shadowed* by another one, as in line 8 of Algorithm 28. Here it entails many more cut-computations and a combinatorially non-trivial problem to find an ordering of $L(t)$ to optimally preserve old clusters. Still we can state the following lemma:

valid clusterings are often maintained

Lemma 4.4.10 *Let $\mathcal{C}(G)$ fulfill the invariant for $G_\ominus$, i.e., let the old clustering be valid for $G_\ominus$. In the case of an inter-cluster deletion, Alg 27 returns $\mathcal{C}(G)$. For an intra-cluster deletion Algorithm 29 returns a clustering $\mathcal{C}(G_\ominus) \supseteq \mathcal{C}(G) \setminus C_{b,d}$, i.e., only $C_{b,d}$ might become fragmented.*

The proof for both cases relies on the fact that any output clustering differing in cluster C_i requires at least one min-v_i-t-cut ($v_i \in C_i$) to separate b, d, invalidating $\mathcal{C}(G)$. Both proofs

can be found in Subsection 4.4.6. Considering the remaining cases, intra-cluster addition obviously retains a valid previous clustering; however, for inter-cluster addition no strong assertion can be made.

4.4.4.2 Running Times

We universally express running times of our algorithms in terms of the number of necessary max-flow computations, leaving open how these are done. A summary of tight bounds is given in Tab. 4.4.1. The columns *lower bound*/*upper bound* denote bounds for the—possibly rather common—case that the old clustering is still valid after some graph update. As discussed in the last subsection, the last column (*guaran. smooth*) states whether our algorithms *always* return the previous clustering, in case its valid; the numbers in brackets denote a tight lower bound on the running time, in case our algorithms do find that previous clustering.

	worst case	old clustering still valid		
		lower bound	upper bound	guaran. smooth
Inter-Del	$\lvert\mathcal{C}(G)\rvert - 2$	$\lvert\mathcal{C}(G)\rvert - 2$	$\lvert\mathcal{C}(G)\rvert - 2$	Yes
Intra-Del	$\lvert\mathcal{C}(G)\rvert + \lvert C_{b,d}\rvert - 1$	1	$\lvert\mathcal{C}(G)\rvert + \lvert C_{b,d}\rvert - 1$	No (1)
Inter-Add	$\lvert C_b\rvert + \lvert C_d\rvert$	1	$\lvert C_b\rvert + \lvert C_d\rvert$	No (2)
Intra-Add	0	0	0	Yes

Table 4.4.1. Bounds on the number of max-flow calculations

For *Inter-Del* (Algorithm 27) we require at most $|\mathcal{C}(G)| - 2$ cuts, separating t from *all* (no *shadowing*) neighbors, except v_b and v_d (comp. Figure 4.4.4). Since this is exactly what happens in case the old clustering remains valid, the other bounds are equal and we know we will find the old clustering. Algorithm 29 (*Intra-Del*) needs to examine all clusters within t's treetop (being treetops themselves), and potentially all vertices in $C_{b,d}$—even if the previous clustering is retained, e.g., with every vertex *shadowing* the one cut off right before, and pair $v_{b,d}, t$ getting hidden. Obviously, we attain the lower bound if we cut away $v_{b,d}$ from t, directly preserving $C_{b,d}$ and the entire treetop of t. For *Inter-Add* (Algorithm 30), we potentially end up separating every single vertex in $C_b \cup C_d$ from t, one by one, even if the previous clustering is valid, as, e.g., v_b might become *shadowed* by some other $v \in C_b \cup C_d$, which ultimately yields the upper bound. In case the previous clustering is valid, however, we might get away with simply cutting off v_b and v_d at once, alongside their former clusters. This means, there is no guarantee that we return the previous clustering; still, with two cuts (v_b-t and v_d-t), we are quite likely to do so. Row *Intra-Add* is obvious. Note that a computation from scratch (static algorithm) entails a tight upper bound of $|V|$ max-flow computations for all four cases, in the worst case.

4.4.4.3 Further Speed-Up

For the sake of brevity we omit a few ideas for effort-saving in the pseudo-code. Apart from the minor Lemmas 4.4.5 and 4.4.7, one heuristic is to decreasingly order vertices in the list $L(t)$, e.g., in line 11 of Algorithm 29 or in line 3 of Algorithm 27; for their static algorithm Flake et al. [87] found that this effectively reduces the number of cuts necessary to compute before t is isolated.

speed-up via dynamic min-u-v-flows

Since individual min-u-v-cuts are constantly required, another dimension of effort-saving lies in dynamically maintaining max-u-v-flows. In fact there are techniques for doing this, two of which we briefly mention here, but leave to read up in [129] and references therein, for readers interested in a detailed description, since that is beyond the scope of this work. Given an initial max-u-v-flow and a graph modification, Kohli and Torr [150] present a method for dynamically maintaining max-u-v-flows that first adjusts the residual graph in a special way,

such that the flow is still valid, and then use any augmenting-path flow algorithm on this residual graph. Another approach is to build up a topologically ordered DAG on vertex subsets of G, directed from u to v. The nodes of this DAG consist of the strongly connected components in the residual graph of a max-u-v-flow, as described by Picard and Queyranne [186]. This DAG can be used to manage all min-u-v-cuts, and can efficiently be updated. Actual effort-saving by these methods depends on the dynamics, in particular hidden *step pairs* and *shadowing* prevents strong assertions.

4.4.5 Experiments

In this brief section, we very roughly describe some experiments we made with an implementation of the update algorithms described above, just for a first proof of concept. The instance we use is a network of e-mail communications within the Fakultät für Informatik at KIT. Vertices represent members and edges correspond to e-mail contacts, weighted by the number of e-mails sent between two individuals during the last 72 hours. We process a queue of 12 560 elementary modifications, 9 000 of which are actual edge modifications, on the initial graph G shown in Figure 4.4.8 ($|V| = 310, |E| = 450$). This queue represents about one week, starting on Saturday (21.10.06); a spam-attack lets the graph slightly grow/densify over the course. We delete zero-weight edges and isolated nodes. Following the recommendations of Flake et al. [87] we choose $\alpha = 0.15$ for the initial graph, yielding 45 clusters, see Figure 4.4.8 for an illustration. We compare their static algorithm (see Section 4.4.2.1) and our dynamic algorithm in terms of the number of max-flow computations necessary to maintain the clustering. For the 9 000 proper steps, static computation needed 2 080 897 max-flows, and our dynamic update needed 198 790, saving more than 90% max-flows, such that in 96% of all modifications, the dynamic algorithm was quicker. Surprisingly, inter-cluster additions have the greatest impact on effort-saving, followed by the trivial intra-cluster additions. By contrast, both deletion operations only mildly outperform the static algorithm. Out of the 9 000 total operations, 49 of the inter-cluster, and 222 of the intra-cluster deletions are the only ones, where the static algorithm happens to be quicker.

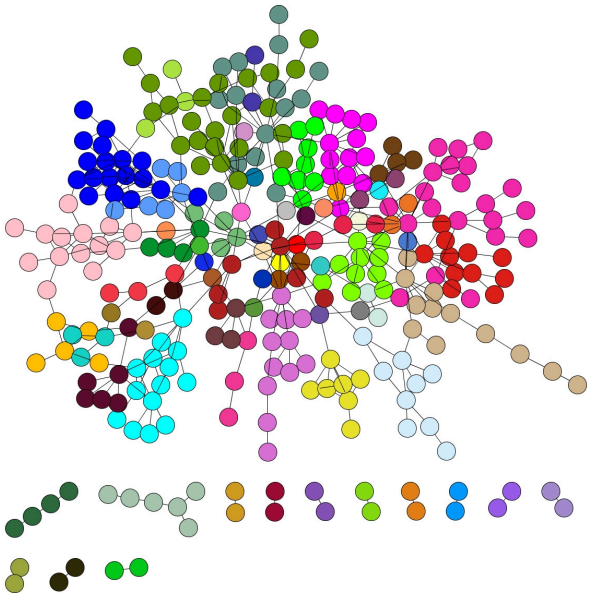
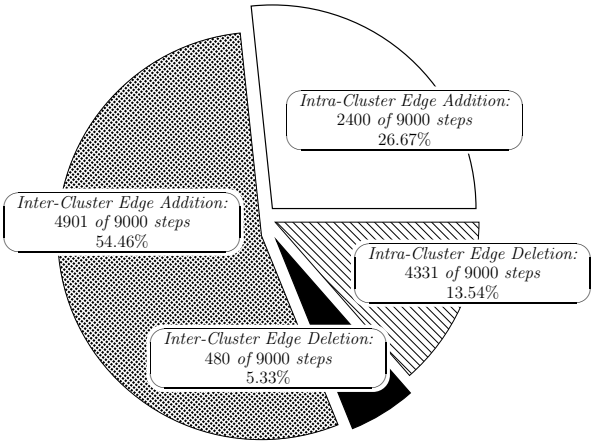


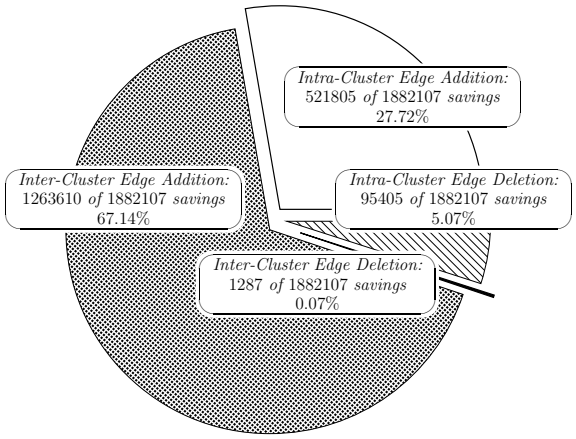
Figure 4.4.8. Initial real world e-mail graph, the clustering is indicated by colors.

Note that updating the clustering after increasing the weight of an edge is done by one of the new algorithms regarding edge additions. The addition of an edge is considered a special case of increasing the weight of an edge. Weight decreases are handled analogously. Thus, in the following we simply talk about intra-cluster and inter-cluster edge additions and edge deletions as the four elementary modifications. Figure 4.4.9a shows the proportions of the elementary modifications regarding the total number of 9 000 modifying steps. The case occurring most often is, with 54.46%, the addition of an edge between two different clusters. The inter-cluster edge deletion, by contrast, only occurs 480 times which corresponds to 5.33%. During the whole experiment the cut-clustering heuristic of Flake et al. [87] calculates 2 080 897 maximum flows. Our updating algorithms, however, only need 198 790 max-flow calculations. This yields a saving of 1 882 107 max-flow calculations which constitutes 90.45% of effort saving. Figure 4.4.9b shows the proportions of the elementary modifications regarding the total number of 1 882 107 savings. We see that the ratio of the percentaged savings provided by edge additions to the proportion of the edge additions regarding the number of total steps is greater than one, while the proportion of edge deletions in Figure4.4.9a provides a

factor 10 speed-up



(a) Total number of steps decomposed into elementary modifications



(b) Total savings of max-flow calculations decomposed into elementary modifications

Figure 4.4.9. Total number of steps and savings of max-flow calculations

smaller proportion of the total savings in Figure 4.4.9b. More precisely, the inter-cluster edge additions are the most efficient modifications, as 54.46% of the total number of steps provide 67.14% of the saved max-flow computations. So each unit of the inter-cluster edge addition proportion on average causes 1.23% of all savings. The least efficient modifications are the inter-cluster edge deletions with 5.33% of all steps gaining only 0.07% of all savings. This corresponds to 0.01% of all savings on average per unit of the inter-cluster deletion proportion.

4.4.6 Omitted Proofs

Theorem 4.4.1

Proof. [of Theorem 4.4.1] The proof uses induction on the $n-1$ edges in $f' \circ f$. The edges are regarded as *step pairs* in a Gomory-Hu execution GH. The set $M \subseteq E_T$ denotes the *step pairs* already applied in the execution, and $T_*(G) = (V_*, E_*, c_*)$ denotes the current working version of the *intermediate min-cut tree*.

Induction base case: Gomory-Hu starts with a single node S containing V, such that $V_* = \{V\}$ and $E_* = \emptyset$. The *contracted* graph G_S thus equals G as nothing is *contracted* yet with $M = \emptyset$. Therefore, T_* corresponds to a $T_\circ$ that is formed by contracting $E_T \setminus M = E_T$ in $T(G)$.

the first step-pair

Now take the first pair $\{u,v\}_1$ of $f' \circ f$ as a *step pair* for the algorithm. Since the current split node is $S = \{V\}$, $\{u,v\}_1$ is a valid *step pair* in S. At the same time $\{u,v\}_1$ represents an edge in $T(G)$ and therefore induces a min-u-v-cut $(U, V \setminus U)$ in $G = G_S$ as a valid *split cut*, with $u \in U$. By splitting and replacing $S = V$ by $S_u = U$ and $S_v = V \setminus U$ connected with a new edge, we get an *intermediate min-cut tree* T_* with $V_* = \{S_u, S_v\} = \{U, (V \setminus U)\}$ and $E_* = \{\{S_u, S_v\}\}$. The only edge in T_* created by the *step pair* $\{u,v\}_1$, has weight $c_T(\{u,v\}_1) = c(U, V \setminus U)$. So after one iteration the *intermediate min-cut tree* T_* exactly corresponds to $T_\circ$ formed by contracting all edges of $E_T \setminus M$ in $T(G)$, with $M = \{\{u,v\}_1\}$, fulfilling our claim. Note, that the *step pair* $\{u,v\}_1$ is not hidden until now.

Induction hypothesis: We now assume the following: The first w pairs $\{u,v\}_1, \ldots, \{u,v\}_w$ in $f' \circ f$ are valid *step pairs* regarding the various split nodes S, and the related edge-induced cuts in G are valid *split cuts* regarding the various *contracted* graphs G_S. The current *intermediate min-cut tree* $T_*(G)$ after these w iterations exactly corresponds to $T_\circ(G)$ formed by contracting all edges of $E_T \setminus M'$, with $M' = \{u,v\}_1, \ldots, \{u,v\}_w$ being the set of the first w *step pairs* in $f' \circ f$. Furthermore we assume that none of the *step pairs* in M' is hidden yet.

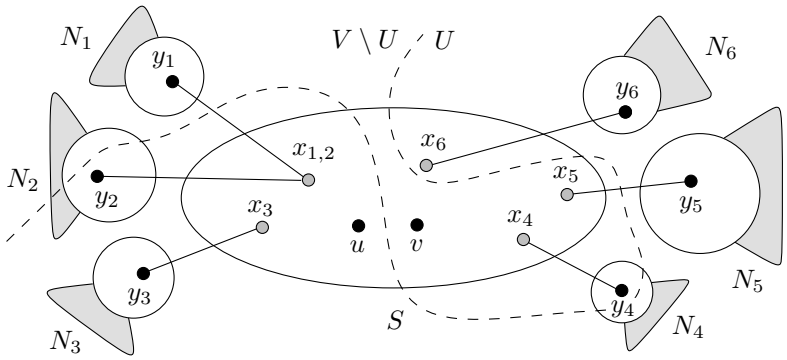


Figure 4.4.10. Intermediate *min-cut tree* $T_\star(G)$ with subtrees $N_1, \ldots, N_6$ and *nearest cut pairs* $\{x_2, y_1\}, \ldots, \{x_6, y_6\}$.

Induction step: Let nodes u, v constitute the next *step pair* $\{u, v\}_{w+1}$ in $f' \circ f$ with split node S. The related cut $(U, V \setminus U)$ in G induced by the edge $\{u, v\}_{w+1}$ in T_*, with $u \in U$, serves as the current *split cut*. We first need to show, that this cut is also a min-u-v-cut in the current *contracted* graph G_S. Let $N(j)$ denote the set of vertices in a subtree N_j of the current split node S. Then the current *contracted* graph G_S results from G by contracting the set $N(j)$ in G for all subtrees of S. The cut $(U, V \setminus U)$ induced by the edge $\{u, v\}_{w+1}$ is a min-u-v-cut in G. Moreover, it does not separate any two vertices g and h lying in the same set Nj, as otherwise the edge $\{u, v\}_{w+1}$ would lie on the unique path $\gamma_{g,h}$ from g to h in $T(G)$, contradicting the assumption that g and h belong to the same subtree of S.Thus the cut $(U, V \setminus U)$ is also a min-u-v-cut in the *contracted* graph G_S and hence is a valid *split cut* for the $(w+1)$-th iteration.

next min-u-v-cut in G_S

Now we can prove that after splitting and replacing the current split node S and after reconnecting the subtrees of S the resulting *intermediate min-cut tree* $T_*(G)$, i.e., the *intermediate min-cut tree* after $w+1$ iterations, again corresponds to $T_\circ(G)$ formed by contracting $T(G)$ by the edges $E_T \setminus M'$, with $M' = \{\{u, v\}_1, \ldots, \{u, v\}_{w+1}\}$ being the set of the first $w+1$ *step pairs* in $f' \circ f$. To this end we show that none of the *step pairs* $\{\{u, v\}_1, \ldots, \{u, v\}_w\}$, which created the edges of the previous *intermediate min-cut tree*, gets hidden by the splitting of S. However, since these *step pairs* directly correspond to edges in $T(G)$ it immediately follows that they never get separated. As the new edge $\{S_u, S_v\}$ in $T_*(G)$ is created by the *step pair* $\{u, v\}_{w+1}$, which represents an edge in $T(G)$, and as all other *step pairs* in $M' = \{\{u, v\}_1, \ldots, \{u, v\}_w\}$ also represent edges in $T(G)$ as well as in $T_*(G)$ (by the induction hypothesis), none of the *step pairs* $\{\{u, v\}_1, \ldots, \{u, v\}_w\}$ gets separated by the *split cut* related to $\{u, v\}_{w+1}$. Therefore, after $w+1$ iterations, the new *intermediate min-cut tree* $T_*(G)$ exactly corresponds to $T_\circ(G)$ formed by contracting all edges of $E_T \setminus M'$ in $T(G)$, with $M' = \{\{u, v\}_1, \ldots, \{u, v\}_{w+1}\}$ being the set of the first $w+1$ *step pairs* in $f' \circ f$. □

no step-pair ever gets hidden

Proof. [of Theorem 4.4.3] This proof uses induction on the subtrees of a split node S in an *intermediate min-cut tree* $T_*(G)$ and shows constructively that there always exists a *split cut* $(U_S, V_S \setminus U_S)$ in G_S as described in Theorem 4.4.2, which is by the way also a min-u-v-cut in G and does not split any subtree of S. Furthermore, the proof shows that the two sides of this *split cut* pick the subtrees as described. For each subtree N_j of S the connecting edge $e_j = \{S, S_j\}$ induces the min-y_j-x_j-cut $\theta_j := (N(j), V \setminus N_j)$ in G, with $y_j \in N(j)$. As it holds that $S \subset V \setminus N(j)$, for each subtree N_j the *step pair* $\{u, v\}$ lies on the $V \setminus N(j)$-side of the minimum y_j-x_j-cut θ_j induced by the connection edge e_j (see Figure 4.4.10). Now let $(U, V \setminus U)$ denote an arbitrary minimum u-v-cut in G, with $u \in U$.

Theorem 4.4.3

Induction base case: We apply Lemma 4.4.3 to θ_1 and $(U, V \setminus U)$ and get a minimum u-v-cut $(U_1, V \setminus U_1)$, with $u \in U_1$, that does not separate any vertices in $N(1)$ and splits

$V \setminus N(1)$ the same way as $(U, V \setminus U)$ does. So, as it holds that $S \subseteq V \setminus N(1)$, also S gets split the same way, and we get

$$\begin{aligned} S \cap U_1 &= S \cap U \\ \text{and} \quad S \cap V \setminus U_1 &= S \cap V \setminus U. \end{aligned}$$

the first reconnection works

With $y_1 \in N(1)$, by Lemma 4.4.3, we further get

$$\begin{aligned} N(1) \cup U &= U_1 && \text{if } y_1 \in U \text{ and} \\ N(1) \cup (V \setminus U) &= V \setminus U_1 && \text{otherwise, i.e., if } y_1 \in V \setminus U, \end{aligned}$$

and therefore, it holds that $N(1) \subseteq U_1$ if and only if $y_1 \in U$. Thus this induces that the related sides of $(U_1, V \setminus U_1)$ and $(U, V \setminus U)$ only differ in $N(1)$, i.e., $U_1 \setminus N(1) = U \setminus N(1)$ and $(V \setminus U_1) \setminus N(1) = (V \setminus U) \setminus N(1)$.

Induction hypothesis: We now assume the cut $(U_z, V \setminus U_z)$ to be a minimum u-v-cut in G, with $u \in U_z$, that does not separate any vertices in any subtree N_j, $j = 1, \ldots, z$, and splits V the same way as $(U, V \setminus U)$ does. More precisely, we assume that it holds that

$$\begin{aligned} S \cap U_z &= S \cap U \\ \text{and} \quad S \cap V \setminus U_z &= S \cap V \setminus U. \end{aligned}$$

and that $N(j) \subseteq U_z$ if and only if $y_j \in U$ for $j = 1, \ldots, z$, while the related sides of $(U_z, V \setminus U_z)$ and $(U, V \setminus U)$ only differ in the sets $N(j)$, $j = 1, \ldots, z$. More formally, this is,

$$\begin{aligned} U_z \setminus \{N(j) | j = 1, \ldots, z\} &= U \setminus \{N(j) | j = 1, \ldots, z\} \text{ and} \\ (V \setminus U_1) \setminus \{N(j) | j = 1, \ldots, z\} &= (V \setminus U) \setminus \{N(j) | j = 1, \ldots, z\}. \end{aligned}$$

Induction step: We apply Lemma 4.4.3 to cut $\theta_{z+1} = (N(z+1), V \setminus N(z+1))$, which is induces by the connection edge $e_{z+1} = \{S, S_{z+1}\}$ of subtree N_{z+1}, and cut $(U_z, V \setminus U_z)$. So we get a minimum u-v-cut $(U_{z+1}, V \setminus U_{z+1})$, with $u \in U_{z+1}$, that does not separate any vertices in $N(z+1)$ and splits $V \setminus N(z+1)$ the same way as $(U_z, V \setminus U_z)$ does. So, as it holds that $S \subseteq V \setminus N(z+1)$, also S gets split the same way, and we get

$$\begin{aligned} S \cap U_{z+1} &= S \cap U_z \underset{\text{induction hypothesis}}{=} S \cap U \\ \text{and} \quad S \cap V \setminus U_{z+1} &= S \cap V \setminus U_z \underset{\text{induction hypothesis}}{=} S \cap V \setminus U. \end{aligned} \tag{4.4.3}$$

With $y_{z+1} \in N(z+1)$, by Lemma 4.4.3, we further get

$$\begin{aligned} N(z+1) \cup U_z &= U_{z+1} && \text{if } y_{z+1} \in U_z \text{ and} \\ N(z+1) \cup (V \setminus U_z) &= V \setminus U_{z+1} && \text{otherwise, i.e., if } y_{z+1} \in V \setminus U_z, \end{aligned}$$

and therefore, it holds that $N(z+1) \subseteq U_{z+1}$ if and only if $y_{z+1} \in U_z$. As, by induction hypothesis, the related sides of $(U_z, V \setminus U_z)$ and $(U, V \setminus U)$ *do not* differ in $N(z+1)$, if follows that $y_{z+1} \in U_z$ if and only if $y_{z+1} \in U$, and therefore, it holds that

$$N(z+1) \subseteq U_{z+1} \quad \text{if and only if} \quad y_{z+1} \in U. \tag{4.4.4}$$

all reconnections work

Furthermore, as a consequence of Lemma 4.4.3 it holds that the related sides of $(U_{z+1}, V \setminus U_{z+1})$ and $(U_z, V \setminus U_z)$ only differ in $N(z+1)$, i.e., $U_{z+1} \setminus N(z+1) = U \setminus N(z+1)$ and $(V \setminus U_{z+1}) \setminus N(z+1) = (V \setminus U_z) \setminus N(z+1)$. So for all sets $N(j)$, $j = 1, \ldots, z$, it follows that $N(j) \subseteq U_{z+1}$ if and only if $N(j) \subseteq U_z$. By induction hypothesis and (4.4.4) we finally get for $j = 1, \ldots, z+1$

$$N(j) \subseteq U_{z+1} \quad \text{if and only if} \quad y_j \in U. \tag{4.4.5}$$

So with Assertion (4.4.3) and Assertion (4.4.5) we finally proved the existence of a minimum u-v-cut in G that splits S the same way as $(U, V \setminus U)$ does, and that does not separate any vertices of any subtree of S. It is easy o see that such a minimum u-v-cut is also a minimum u-v-cut in graph $G(S)$, which results from G by contracting all subtrees of S. So Theorem 4.4.2 and Theorem 4.4.3 are both proven true. □

Lemma 4.4.4

Proof. [of Lemma 4.4.4] The Gomory-Hu execution $\mathrm{GH}_{\oplus(\ominus)}$, by definition, uses the same sequence k of *split cuts* as execution GH does, which considers the graph G and reaches $T_\circ(G)$ as *intermediate min-cut tree* after the application of k. Therefore, execution $\mathrm{GH}_{\oplus(\ominus)}$ also has $T_\circ(G)$ as *intermediate min-cut tree* on condition that k represents a feasible sequence of *split cuts* concerning the modified graph $G_{\oplus(\ominus)}$. This then implies f to be a feasible sequence of *step pairs*. Similar to the proof of Theorem 4.4.1, this proof uses induction on the *split cuts* in k.

induction on split cuts yields: $\mathrm{GH}_{\oplus(\ominus)}$ works

Induction base case: The execution $\mathrm{GH}_{\oplus(\ominus)}$ starts with the *first split cut* induced by the first edge $\{u, v\}_1$ in f. As the *first split cut* is applied to the *contracted* graph $G_S^{\oplus(\ominus)} = G_{\oplus(\ominus)}$, and $\{u, v\}_1 \in M$ induces a minimum u-v-cut in $G_{\oplus(\ominus)}$ (by the choice of M and Corollary 4.4.1), the *first split cut* is feasible.

Induction hypothesis: We now assume the *split cuts* induced by the edges $\{u, v\}_2, \ldots, \{u, v\}_z$ in f to be feasible regarding the various *contracted* graphs $G_S^{\oplus(\ominus)}$ in $z - 1$ further iterations.

Induction step: Consider the next *split cut* induced by the edge $\{u, v\}_{z+1}$ in f, which constitutes the *step pair* in the current split node S. For the following argumentation we need to distinguish the cases of edge addition and edge deletion.

Edge addition ($M = E_T \setminus \gamma$): If it holds for the modified vertices b and d that $\{b, d\} \not\subseteq S$, it follows that $G_S^{\oplus(\ominus)} = G_S$ in this iteration, as the modified edge $\{b, d\}$ then is *contracted* (Note that b and d never lie in different subtrees of S, as the edges on γ, which correspond to the cuts that separate b and d, are not included in M, and f respectively). With $G_S^{\oplus(\ominus)} = G_S$ the current *split cut* is feasible.

If it holds that $\{b, d\} \subseteq S$, the *contracted* graph $G_S^{\oplus(\ominus)}$ results from G_S by the addition of the edge $e_\oplus = \{b, d\}$ and, as the edge $\{u, v\}_{z+1}$ cannot lie on the path γ, the current *split cut* does not separate b and d. So, as the current *split cut* is a minimum u-v-cut in G_S, by Lemma 4.4.1 the current *split cut* also represents a minimum u-v-cut in $G_S^\oplus$ and hence is feasible.

Edge deletion ($M = \gamma$): As all *split cuts* considered so far separate the modified vertices b and d, the current *intermediate min-cut tree* is a path of nodes with b included in the first and d included in the last node. So if the current split node S includes b (the case when it includes d is symmetric), then S has only one subtree, which includes d. If S includes neither b nor d, then S has exactly two subtrees, with b and d in different subtrees. In both cases the graph $G_S^\ominus$ results from G_S by the deletion of the edge $e_\ominus = \{b, d\}$. Furthermore, the current *split cut* must separate b and d, as the edge $\{u, v\}_{z+1}$ lies on path γ. So, as the current *split cut* is a minimum u-v-cut in G_S, by Lemma 4.4.1 the current *split cut* also represents a minimum u-v-cut in $G_S^\ominus$ and hence is feasible.

As the remaining *step pairs* and *split cuts* in $f_{\oplus(\ominus)}$ and $k_{\oplus(\ominus)}$ are defined as arbitrary valid sequences, and as such sequences always exist, the assertion of the lemma is proven. □

Lemma 4.4.5

Proof. [of Lemma 4.4.5] Let θ be the cut induced by $e_{\min}$; then in $G_\oplus$ it has weight $c_\oplus(\theta) = c(\theta) + \varrho$. Suppose now θ' is b-d-cut with $c_\oplus(\theta') < c_\oplus(\theta)$. Since θ' must cut edge $\{b, d\}$ in $G_\oplus$, its weight in G is $c(\theta') \leq c_\oplus(\theta') - \varrho$. This yields $c(\theta') < c(\theta)$, a contradiction to $e_{\min}$'s minimality for G. □

Lemma 4.4.6 *Proof.* [of Lemma 4.4.6] Consider the min-u-v-cut $(U, V \setminus U)$ in $G_\ominus$ to be the *first split cut* of GH, with *step pair* $\{u, v\}$. As the cut does not separate $\{b, d\}$, wlog. let $b, d \in V \setminus U$. Let $\{U\}$ be the next split node of GH, such that b and d are *contracted* into $\eta_{V\setminus U}$ in $G_U^\ominus$. Since for any *step pair* within U, $\{b, d\}$ are not separated, by the correctness of Gomory-Hu and Lemma 4.4.1, any previous min-g-h-cut is still valid in $G_\ominus$. Furthermore, Lemma 4.4.2 asserts that previous *cut pairs* within U also stay valid. □

Lemma 4.4.7 *Proof.* [of Lemma 4.4.7] By Lemma 4.4.1 $\{y_b, g\}, \{y_d, g\}$ stay valid min-cuts in $G_\ominus$. A GH starting with *step pairs* $\{y_b, g\}, \{y_d, g\}$ yields a path of nodes N_b, S_g, N_d as an *intermediate* cut tree, with $u, v \in S_g$. Suppose there is a cheaper u-v-cut θ' than that of $\{u, v\}$, then by Lemma 4.4.1 θ' must separate b and d and thus cut $\{y_b, g\}$ or $\{y_d, g\}$. But then θ' is cheaper than $c(\{y_b, g\}) - \varrho$ (and than $c(\{y_d, g\}) - \varrho$) and either violates that $(N_b, V \setminus N_b)$ remains a min-y_b-g-cut or that $(N_d, V \setminus N_d)$ remains a min-y_d-g-cut; a contradiction. □

Lemma 4.4.8 *Proof.* [of Lemma 4.4.8] We prove this lemma regarding the subtree N_b by contradiction. The proof regarding the subtree N_d is symmetric. We show that the cut $\theta := (\Uparrow_A, N(b) \cup N(d) \cup \sharp \cup \Uparrow_B)$, which differs from θ'_b in the set $N(b)$, would be cheaper in G than the edge-induced minimum u-v-cut $\theta_{\min} := (\Uparrow, N(b) \cup N(d) \cup \sharp)$ in G, which differs from θ_b in the set $N(b)$, if θ'_b was cheaper than θ_b.

So we assume that $c_\ominus(\theta_b) > c_\ominus(\theta'_b)$. As the cuts θ and $\theta_{\min}$ both do not separate the modified vertices b and d, each of them is of the same weight in $G_\ominus(S)$, $G_\ominus$ and G, by Lemma 4.4.1. Here we consider the weights in $G_\ominus$ and get

$$\begin{aligned} c_\ominus(\theta_{\min}) &= c_\ominus(\theta_b) - c_\ominus(N(b), N(d) \cup \sharp) + c_\ominus(N(b), \Uparrow) \quad \text{and} \\ c_\ominus(\theta) &= c_\ominus(\theta'_b) - c_\ominus(N(b), N(d) \cup \sharp \cup \Uparrow_B) + c_\ominus(N(b), \Uparrow_A) \end{aligned}$$

With $(N(d) \cup \sharp) \subseteq (N(d) \cup \sharp \cup \Uparrow_B)$ and $\Uparrow_A \subseteq \Uparrow$ it holds that

$$\begin{aligned} c_\ominus(N(b), N(d) \cup \sharp) &\leq c_\ominus(N(b), N(d) \cup \sharp \cup \Uparrow_B) \quad \text{and} \\ c_\ominus(N(b), \Uparrow) &\geq c_\ominus(N(b), \Uparrow_A) \end{aligned}$$

So with the assumption that $c_\ominus(\theta_b) > c_\ominus(\theta'_b)$ we finally get

$$\begin{aligned} c_\ominus(\theta_{\min}) - c_\ominus(\theta) &= [c_\ominus(\theta_b) - c_\ominus(\theta'_b)] \\ &- [c_\ominus(N(b), N(d) \cup \sharp) - c_\ominus(N(b), N(d) \cup \sharp \cup \Uparrow_B)] \\ &+ [c_\ominus(N(b), \Uparrow) - c_\ominus(N(b), \Uparrow_A)] > 0 \end{aligned}$$

This contradicts the fact that the edge-induced u-v-cut $\theta_{\min}$ is a min-u-v-cut in G. □

Lemma 4.4.9 *Proof.* [of Lemma 4.4.9] Again, we prove this lemma regarding the subtree N_b. The proof regarding the subtree N_d is symmetric. The assertion of this lemma follows by Lemma 4.4.8. We express the cuts θ_{bb} and θ'_{bb} with the aid of the cuts θ_b and θ'_b considered in Lemma 4.4.8, which just differ in the set $\sharp_A$. So we get

$$\begin{aligned} c_\ominus(\theta_{bb}) &= c_\ominus(\theta_b) - c_\ominus(\sharp_A, N(b) \cup \Uparrow) + c_\ominus(\sharp_A, N(d) \cup \sharp_B) \quad \text{and} \\ c_\ominus(\theta'_{bb}) &= c_\ominus(\theta'_b) - c_\ominus(\sharp_A, N(b) \cup \Uparrow_A) + c_\ominus(\sharp_A, N(d) \cup \sharp_B \cup \Uparrow_B) \end{aligned}$$

So with $c_\ominus(\theta_b) \leq c_\ominus(\theta'_b)$, by Lemma 4.4.8, we finally get

$$\begin{aligned} c_\ominus(\theta'_{bb}) - c_\ominus(\theta_{bb}) &= [c_\ominus(\theta'_b) - c_\ominus(\theta_b)] \\ &- [c_\ominus(\sharp_A, N(b) \cup \Uparrow_A) - c_\ominus(\sharp_A, N(b) \cup \Uparrow)] \\ &+ [c_\ominus(\sharp_A, N(d) \cup \sharp_B \cup \Uparrow_B) - c_\ominus(\sharp_A, N(d) \cup \sharp_B)] \geq 0 \end{aligned}$$

□

Proof. [of Lemma 4.4.10] Consider inter-cluster deletion (Algorithm 27) first. To return a new clustering $\mathcal{C}(G_{\ominus})$ different from $\mathcal{C}(G)$ the algorithm needs to find a new cheaper min-v_i-t-cut for at least one cut-vertex $v_i \in \{v_1, \ldots, v_z\}$. As the previous clustering is supposed to be also valid for $G_{\ominus}$, there must exist another vertex $u \in C_i$ that serves as a witness that the cut θ_i (defining C_i) still constitutes a min-u-t-cut in the modified graph $G_{\alpha}^{\ominus}$. Then there must exists a *min-cut tree* $T(G_{\alpha}^{\ominus})$ such that the edge-induced minimum v_i-t-cut represented in this new *min-cut tree* gets *shadowed* and must not separate the modified vertices b, d. This contradicts Lemma 4.4.1, which says that each new minimum v_i-t-cut in $G_{\alpha}^{\ominus}$ which is cheaper than the previous one in graph G_{α} needs to separate the modified vertices b, d. *Lemma 4.4.10*

Considering intra-cluster deletion (Algorithm 29), all the above arguments apply to the clusters $\mathcal{C}(G) \setminus \{C_{b,d}\}$. Thus these clusters are again found; however $C_{b,d}$ might be fragmented in an almost arbitrary manner. □

4.4.7 Omitted Algorithms

Algorithm 30 gives the pseudo-code for the handling edge additions between clusters. Since its description is almost analogous to the above algorithms, the only detail we point out is the following. In line 5 the so called *best* min-v_b-t-cut (or min-v_d-t-cut) is used. Consider the situation sketched out in Figure 4.4.6, and let us choose among all possible min-v_b-t-cuts $U, V \setminus U$, $v_b \in U$ (given by some max-flow). To maximize both the progress in terms of clustering and temporal *smoothness*, we chooses a cut that puts as many vertices of the former cluster C_b as possible into U while cutting away from t as few other cut-vertices as possible.

4.4.8 Problems in the work of Saha and Mitra

This section gives a brief overview of the errors we found in the work of B. Saha and P. Mitra [193]. A preliminary version of this work is [192]. The authors describe four procedures for updating a clustering and a data structure for the deletion and the addition of intra-cluster and inter-cluster edges. We briefly point out the errors in the authors' procedure that deals with the addition of intra-cluster edges. For a thorough discussion we refer the reader to Hartmann [129]. Algorithm 31 sketches the approach given in [193] for handling edge additions between clusters. Summarizing we found that *Case 1* does maintain *quality* but not the *invariant*. *Case 2* maintains both *quality* and the *invariant* if and only if the input fulfills the *invariant*, however it can be shown that this case is of purely theoretical interest and extremely improbable. Finally, *Case 3* neither maintains *quality* nor the *invariant*. The following subsections illustrate these shortcomings with examples.

4.4.8.1 A Counter-Example for *Case 1* and *Case 2*

We now give an example instance which the algorithm given in [193] fails to cluster correctly. The two upper figures (Figure 4.4.11a,4.4.11b) show the input instance, as computed by algorithm Cut-Clustering. In Figure 4.4.11c, a first edge addition then triggers *Case 1*, and thus the clustering is kept unchanged. Note that here, *quality* is still maintained. Then in Figure 4.4.11d a second edge is added and handled by *Case 2*, since inter-cluster quality is violated ($c(C_1, C_2) = 4\alpha > 3 = \alpha \cdot \min\{|C_1|, |C_2|\}$), and the condition for *Case 2* in Line 3 of the algorithm is fulfilled ($2 \cdot 4\alpha/6 > \alpha$). Thus the two clusters are merged. In this result the dashed cut in Fig 4.4.11d shows an intra-cluster cut with value $c(\text{dashed}) = 2.75 \cdot \alpha < 3 \cdot \alpha$, which violates intra-cluster quality, as claimed in Equation (4.4.2).

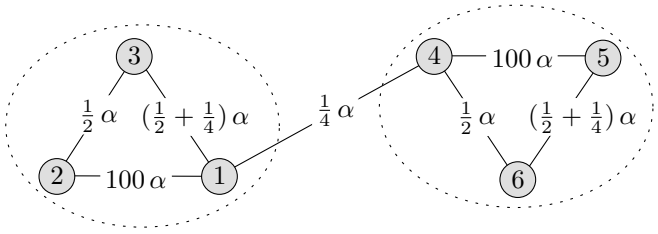
Algorithm 30: Inter-Cluster Edge Addition

Input: $W(G), \Theta(G), G_\alpha^\oplus = (V, E \cup \{\{b,d\}\}), c_\alpha^\oplus$, edge $\{b,d\}$ with weight ϱ
Output: $W(G_\ominus), \Theta(G_\ominus)$

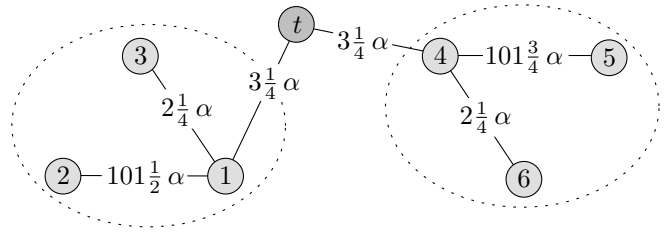
$L(t) \leftarrow \{v_b, v_d\}$, $l(t) \leftarrow \emptyset$
$D(v_b) \leftarrow \emptyset$, $D(v_d) \leftarrow \emptyset$
$W(G_\oplus) \leftarrow \{v_1, \ldots, v_z\}$, $\Theta(G_\oplus) \leftarrow \{\theta_1, \ldots, \theta_z\}$
while $L(t)$ *has next element* u_i **do**
  $\theta \leftarrow$ "best cut" given by FlowAlgo(u_i, t) `// see text`
  if $c_\alpha^\oplus(\theta_i) = c_\alpha(\theta_i^{old}) + \varrho$ **then**
    Move u_i from $L(t)$ to $W(G_\oplus)$
    Add θ_i^{old} to $\Theta(G_\oplus)$
  else
    Add θ_i to $l(t)$ `// pointed at by` u_i
    while $L(t)$ *has next element* $u_j \neq u_i$ **do**
      if θ_i *separates* u_j *and* t **then**
        Delete u_j from $L(t)$
        if $l(t)$ *already contains a cut* θ_j *pointed at by* u_j **then**
          Delete θ_j from $l(t)$
    while $W(G_\oplus)$ *has next element* v_i **do**
      if θ_i *separates* v_i *and* t **then**
        Delete cut which v_i points to from $\Theta(G_\oplus)$
        Move v_i from $W(G_\oplus)$ to $D(u_i)$
while $L(t)$ *has next element* u_i **do**
  $(R, V_\alpha \setminus R) := \theta_i$, $t \in R$, (cut in $l(t)$ which u_i points at)
  forall *vertices* v_j *in* $D(u_i)$ **do**
    $\theta_i \leftarrow (R \setminus C_j, (V_\alpha \setminus R) \cup C_j)$ `// by Theorem 4.4.3`
  forall *vertices* v_j *in* $W(G_\oplus)$ **do**
    $\theta \leftarrow (R \cup C_j, (V_\alpha \setminus R) \setminus C_j)$ `// by Theorem 4.4.3`
Resolve all crossings in $l(t)$ `// by Lemma 4.4.3`
Add all vertices in $L(t)$ to $W(G_\oplus)$
Add all (non-crossing) cuts in $l(t)$ to $\Theta(G_\oplus)$
Isolate t
return $W(G_\oplus), \Theta(G_\oplus)$

Algorithm 31: Old Inter-Edge Addition

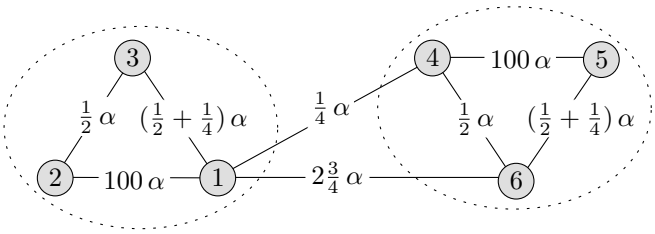
Input: $G = (V, E, w)$, α, $\mathcal{C}$, new edge $e_\oplus = \{b,d\}$, $b \in C_b$, $d \in C_d$
if *inter-cluster quality of* C_d *and* C_b *is maintained* **then** *Case 1*:
  return $\mathcal{C}$ (do nothing)
else if $\frac{2c(C_b,C_d)}{|V|} \geq \alpha$ **then** *Case 2*:
  return $(\mathcal{C} \setminus \{C_b, C_d\}) \cup \{\{C_b \cup C_d\}\}$ (merge C_b and C_d)
Case 3 (default): dissolve C_b and C_d and *contract* all other nodes
perform adapted Cut-Clustering on this instance
return $(\mathcal{C} \setminus \{C_b, C_d\}) \cup \{$newly formed clusters of nodes from C_b and $C_d\}$



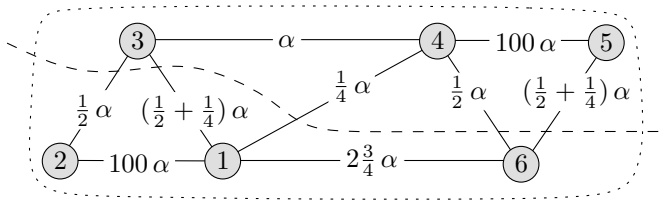
(a) Graph G^0 with clustering $\mathcal{C}(G^0)$ by the Cut-Clustering method



(b) *min-cut tree* $T(G^0_\alpha)$ inducing the clustering $\mathcal{C}(G^0)$ shown in Figure (a)



(c) Adding edge $\{1,6\}, 11/4\alpha$ yields G^1 with clustering $\mathcal{C}(G^1)$ resulting from *Case 1* of the inter-edge-add algorithm



(d) Adding edge $\{3,4\}, \alpha$ yields G^2 with the trivial clustering $\mathcal{C}(G^2)$ by *Case 2*, which violates Equation (4.4.2) (dashed cut)

Figure 4.4.11. A dynamic instance violating the clustering quality. Weights are parameterized by α. After two modifications to G^0 the algorithm returns one cluster which can be cut (dashed) with a cut value that violates *quality*.

4.4.8.2 A Counter-Example for *Case 3*

Finally we give an example instance which the algorithm given in [193] fails to cluster correctly due to shortcomings in *Case 3*.

4.4.9 Approximate Guarantees

This slightly secluded section returns to the static *min-cut tree* and asks first questions about a speed-up of that algorithm at the expense of accuracy. Recall that the static clustering algorithm based on *min-cut tree*, Algorithm 25, is proven to yield the following quality guarantee for the resulting clustering:

$$\underbrace{\frac{c(C, V \setminus C)}{|V \setminus C|}}_{\text{inter-cluster cuts}} \leq \alpha \leq \underbrace{\frac{c(P,Q)}{\min\{|P|,|Q|\}}}_{\text{intra-cluster cuts}} \quad \forall C \in \mathcal{C} \quad \forall P, Q \neq \emptyset \quad P \dot\cup Q = C$$

Using a *b*-*Min-Cut Tree*. Suppose now we cannot provide a rigorous *min-cut tree*, but only an approximate one. In the following we shall see how this affects what we can still guarantee about the quality as stated in Equation 4.4.9.

Definition 4.4 *A* b-min-cut tree $T^b(G)$ *is a tree on* V*. The weight of the lightest edge* e *on the unique path in* $T^b(G)$ *between* u *and* v *is equal to the weight of the cut induced by* e *in* $T^b(G)$*. Furthermore, the weight of* e *is larger than the actual min-*u*-*v*-cut by a factor of at most* $b \geq 1$*.* *b-min-cut tree*

Note that a 1-*min-cut tree* is a *min-cut tree*. Along the lines of the original proof of Equation 4.4.9 in [87] we can now derive lemmata which are analogous to those originally proven. The crucial point in the following is the impact of b.

helping lemma

Lemma 4.4.11 *Let $T^b(G)$be a b-min-cut tree of G, and (u,w) an edge in $T^b(G)$, which induces the cut (U,W) in G, with $w \in W$. For any non-trivial 2-partition $\{U_1, U_2\}$ of U with $u \in U_1$ we get:*

$$c(U_2, W) \leq b \cdot c(U_1, U_2) + (b-1) \cdot c(U_1, W) \tag{4.4.6}$$

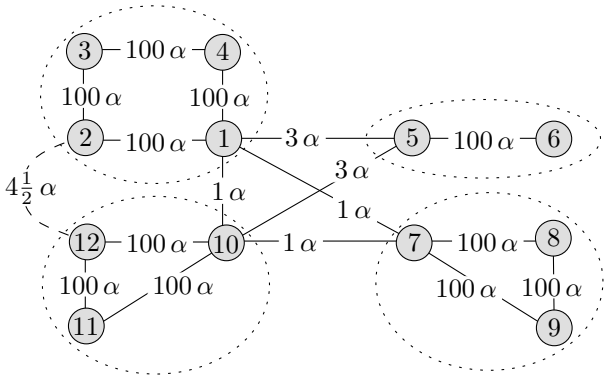
compared to the following if $b = 1$:

$$c(W, U_2) \leq c(U_1, U_2) \qquad (\text{if } b = 1)\ . \tag{4.4.7}$$

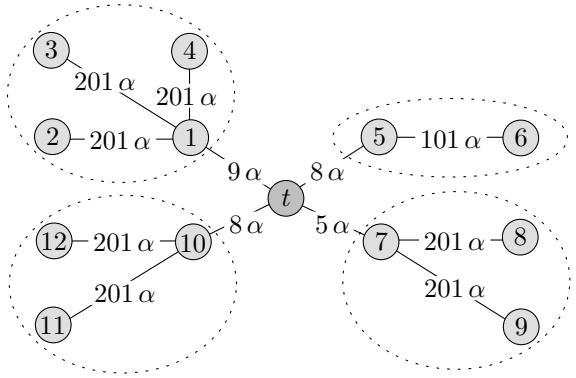
Proof. Consider the cut $(U_1, W \cup U_2)$, which also separates u and w. It cannot be much lighter than cut (U,W) induced by (u,w) in the *b-min-cut tree*, more precisely:

$$\begin{aligned} c(U, W) &\leq b \cdot c(U_1, W \cup U_2) \\ c(U_1 \cup U_2, W) &\leq b \cdot c(U_1, W \cup U_2) \\ c(U_1, W) + c(U_2, W) &\leq b \cdot (c(U_1, W) + c(U_1 \cup U_2)) \\ c(U_2, W) &\leq b \cdot c(U_1 \cup U_2) + (b-1) \cdot c(U_1, W) \end{aligned}$$

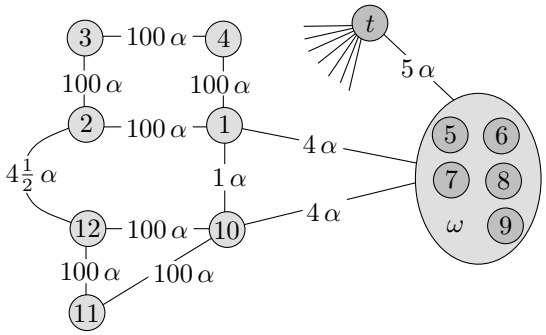
□



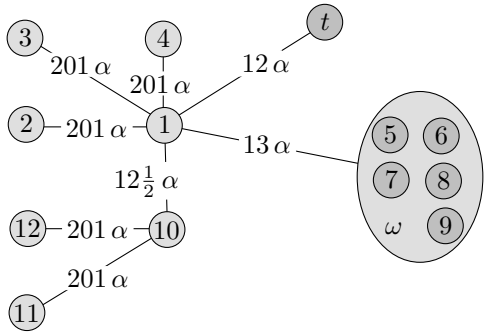
(a) Graph G with clustering $\mathcal{C}(G)$ resulting from the cut-clustering method



(b) *Min-cut tree* $T(G_\alpha)$ inducing the clustering $\mathcal{C}(G)$ shown above



(c) Graph G'_α, resulting from $G_\oplus$ by adding the sink t and contracting the vertices in $\{5,6\} \cup \{7,8,9\}$



(d) Min-cut tree $T(G'_\alpha)$ of graph G'_α

Figure 4.4.12. Counter-example for the correctness of *Case 3*. Figures (a) and (b) describe the graph and the *min-cut tree* before edge $\{2, 12\}$ is inserted. The the edge is added and Figure (c) describes the resulting construction given in [193], on which Cut-Clustering is then applied, yielding Figure (d). The result does neither conform to Equation (4.4.2) nor to what is attempted to be proven in [193].

Lemma 4.4.12 *Let G_α be graph G augmented by sink t as in Alg. 25, S the resulting cluster of node $s \in G$ with $s \sim t$ in $T^b(G_\alpha)$, and $\{P, Q\}$ a non-trivial 2-partition of S with $s \in P$, then the following holds:* *intra-quality*

$$\alpha \leq \frac{b \cdot c(P,Q) \; + \; (b-1) \cdot c(P, V \setminus S \cup \{t\})}{\min\{|P|, |Q|\}} \leq b \cdot \frac{c(P, V \setminus P \cup \{t\})}{\min\{|P|, |Q|\}} \tag{4.4.8}$$

compared to the following if $b = 1$:

$$\alpha \leq \frac{c(P,Q)}{\min\{|P|, |Q|\}} \; . \tag{4.4.9}$$

Proof. Since $s \in P$ and $s \sim t$ in the *b-min-cut tree* $T^b(G_\alpha)$ of G_α, Lemma 4.4.11 applies and yields:

$$\begin{aligned}
c(V \setminus S \cup \{t\}, Q) &\leq b \cdot c(P,Q) + (b-1) \cdot c(P, V \setminus S \cup \{t\}) \\
c(V \setminus S, Q) + c(\{t\}, Q) &\leq b \cdot c(P,Q) + (b-1) \cdot c(P, V \setminus S \cup \{t\}) \\
c(\{t\}, Q) &\leq b \cdot c(P,Q) + (b-1) \cdot c(P, V \setminus S \cup \{t\}) \\
\alpha \cdot |Q| &\leq b \cdot c(P,Q) + (b-1) \cdot c(P, V \setminus S \cup \{t\}) \\
\alpha \cdot \min\{|P|, |Q|\} &\leq b \cdot c(P,Q) + (b-1) \cdot c(P, V \setminus S \cup \{t\})
\end{aligned}$$

□

Lemma 4.4.13 *Let G_α be graph G augmented by sink t as in Alg. 25, and let S be the resulting cluster of node $s \in G$. Then the following holds:* *inter-quality*

$$\frac{c(S, V \setminus S)}{b \cdot |V| - |S|} \leq \alpha \tag{4.4.10}$$

compared to the following if $b = 1$:

$$\frac{c(S, V \setminus S)}{|V \setminus S|} \leq \alpha \tag{4.4.11}$$

Proof. Let $T^b(G_\alpha)$ again be a *b-min-cut tree* of G_α. Furthermore, suppose edge (s', t) is an edge in $T^b(G_\alpha)$ whose removal yields S and $V \setminus S \cup \{t\}$. By definition of $T^b(G_\alpha)$, the weight of edge (s', t) in $T^b(G_\alpha)$ is at most b times as large as a min-s'-t-cut in G_α, and its weight is equal to the cut $(S, V \setminus S \cup \{t\})$ in G_α. Consider now the cut $(V, \{t\})$ in G_α, which also separates s' and t. Since the latter cut has at least the weight of a min-s'-t-cut, we get:

$$\begin{aligned}
c(S, V \setminus S \cup \{t\}) &\leq b \cdot c(V, \{t\}) \\
c(S, V \setminus S) + c(S, \{t\}) &\leq b \cdot (c(V \setminus S, \{t\}) + c(S, \{t\})) \\
c(S, V \setminus S) &\leq b \cdot c(V \setminus S, \{t\}) + (b-1) \cdot c(S, \{t\}) \\
c(S, V \setminus S) &\leq b \cdot \alpha |V - S| + (b-1) \cdot \alpha |S| \\
c(S, V \setminus S) &\leq b \cdot \alpha |V| - \alpha |S|
\end{aligned}$$

□

For those familiar with [87], note that the slightly more general existence argument pulled through the according lemmata is not necessary for our purpose. Summing up Lemmata 4.4.12 and 4.4.13 we can now state a Theorem about the guarantees we can make when using a *b-min-cut tree* for clustering.

approximate quality

Theorem 4.4.4 *Given a graph G and a real $b \geq 1$. Let $\mathcal{C}_b$ be a clustering of G identified by Algorithm 25 but using a b-min-cut tree $T^b(G_\alpha)$ instead of a min-cut tree in line 5. Let P, Q, S, and t be defined as above. Then for any cluster $C \in \mathcal{C}_b$ the following bounds hold:*

$$\frac{c(S, V \setminus S)}{b \cdot |V| - |S|} \leq \alpha \leq \frac{b \cdot c(P,Q) + (b-1) \cdot \max\{c(P, V \setminus S \cup \{t\}), c(Q, V \setminus S \cup \{t\})\}}{\min\{|P|, |Q|\}} \tag{4.4.12}$$

Note that Theorem 4.4.4 also applies to non-simple and/or weighted graphs. While the given bounds are rather clumsy, compared to the case $b = 1$, they do show that a factor b for the quality of a *min-cut tree* does carry over and still yields guarantees on the goodness of a clustering. We leave the question about how to find a *b-min-cut tree* open. A potential starting point might be the sampling technique of Benzúr and Karger [36] which lets us compute $(1 \pm \epsilon)$-approximate min-s-t-cuts in a reduced graph with only $n \log n/\epsilon^2$ edges.

Section 4.5

Time-Dependent Graph Clustering

Oh my Strogg, they're after the databrain!

(Strogg Nexus, Outskirts, Quake Wars)

In the past sections of this chapter on clustering dynamic graphs we had our focus on an *online* setting. There, the task consisted of—roughly speaking—updating a clustering after the graph has changed. Recall that our primary goal was to *quickly* obtain a *good* clustering of a current *time step*. As our secondary goal, we tried to enforce a *smooth* transition between two steps. In this final section we shall investigate a specific *offline* dynamic setting, i.e., all *time steps* of the dynamic graph are known: Generally speaking, a *time-dependent* clustering is a clustering of a dynamic graph where the result respects the temporal evolution of the graph and reveals the evolution of the clustering, as in Figure 4.5.1. We retain our goals from the *online* setting, being the quality of each *time step* and the *smoothness* of transitions between *time steps* (and, of course, speed), however we require one additional point: We postulate a correspondence between the clusters of consecutive *time steps*, i.e., one should be able to track how a cluster evolves over time and see if it grows, joins others or dissolves. The crucial point is, that some advantages of a *smooth* dynamic clustering are lost, if there is no good way to actually *follow* clusters over time. Obvious applications for such a clustering of graph sequences are the identification of microscopic and macroscopic trends in the community structure. For this purpose the *batch size* of updates will be rather large, compared to close-to-realtime *online* settings. Both for cause studies and for the prediction of the future behavior of an unsupervised network, such analyses are invaluable.

good vs. smooth
offline
time-dependent

track clusters

trends in community structure

Our approach is particularly opposed to the intuitive and immediate (and arguably reasonable) idea to proceed as follows in practice: For each *time step* G_i of $\mathcal{G}$, find a good

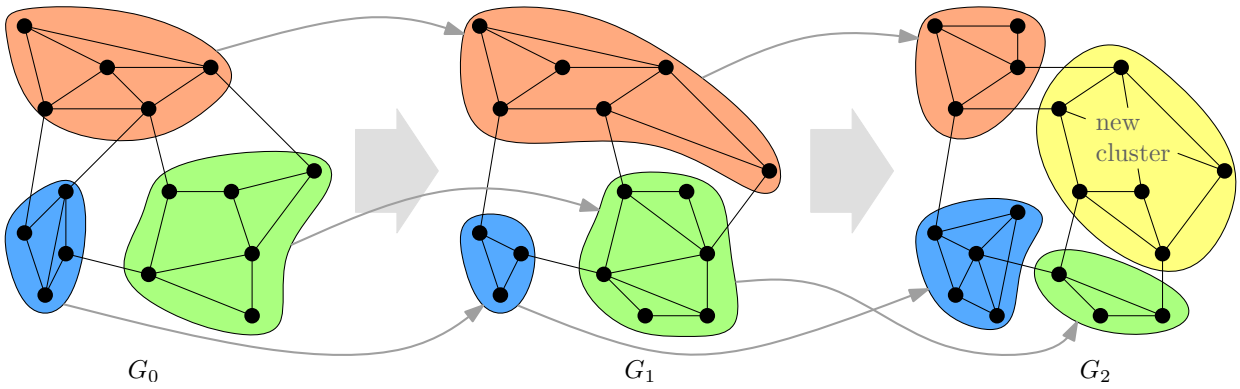


Figure 4.5.1. These three *time steps* of a dynamic graph, do not only feature a *smooth dynamic clustering*, but also a way to track clusters over time (gray arrows); thus it is a *time-dependent* clustering. Note that in G_2 a new cluster has emerged from parts of the green and the red clusters in G_1.

static clustering $\mathcal{C}_i$, and then find a matching between the clusters of $\mathcal{C}_{i-1}$ and $\mathcal{C}_i$. In this formulation one attempts to track the elements of independent clusterings, and even ignores *smoothness* when identifying the clusterings, something we will discuss the disadvantages of below in Section 4.5.2. Towards our goal of a true *time-dependent* clustering, we propose a powerful ILP-based toolkit which solves a vanilla *offline* setting for quality and *smoothness*, and then requires an additional matching stage in order to yield a correspondence of clusters. We will even take one step back and start with an *online* scenario which allows for a rigorous balance between quality and *smoothness*. We then build upon this and extend the proposed formulation to an *offline* setup which is capable of solving many problem statements for *offline* clustering. Then we will turn to *time-expanded* graph clustering, which we advocate to be a better approach for practical *time-dependent* clustering, since it adds to the latter an immediate correspondence of clusters over time.

Time-expanded clustering is one of the most exciting approaches I tackled during my work. Compared to the few related methods in the literature, it is very elegant and it works well. Without the excellent support from my student Dieter Glaser on this topic, Section 4.5.3 would probably not be part of this thesis. Furthermore it is nice to be backed by sound and rigorous problem statements and a theoretically optimal ILP formulation. At this point I would like to thank Florian Hübner, Martin Nöllenburg and especially Marco Gaertler for the fruitful discussions on ILPs. None of the content herein has yet been published.

Main Results

- We establish a set of constraints for integer linear programming which can be arranged as to solve most reasonable *online* and *offline* problem statements of dynamic clustering, involving strict requirements for quality and *smoothness*. (Section 4.5.1)
- A bicriterial *online* ILP formulation is shown to be feasible and to behave in exact accordance to intuition, scaling the trade-off between quality and *smoothness* with one sole parameter. (Section 4.5.1.1)
- There is an ILP formulation for many problem statements of *offline* graph clustering which employ quality and *smoothness* as constraints or optimization goals. (Section 4.5.1.2)
- We propose and advocate the new concept of *time-expanded* clustering for *time-dependent* clusterings of dynamic graphs. (Section 4.5.2)
- In a case study on the email graph, viewed via 11 *time steps* of aggregated months, we show how the degrees of freedom of *time-expanded* clustering can reasonably be filled and exhibit the potential of this technique. (Section 4.5.3)

Future Work. *Time-expanded* clustering appears to be able to answer questions that arise in many fields in a simple and sound manner. Since it requires a careful modeling of the instance, this technique calls for a good deal of testing beyond what is covered here. Then, however, I see much potential in this method to work off the shelf for many applications.

4.5.1 ILP-Based Solutions

In Section 4.3 we put our insights about ILPs for clustering algorithms from Section 2.4 to good use and defined a *partial ILP* which operated on a *preclustering* $\tilde{\mathcal{C}}$ that depended on the chosen strategy for how much impact a small *change* in a graph should be allowed to have. Instead of optimizing *modularity* within this small search space and enforcing *smoothness* by the small size of this space, we can take a more brutish approach and put both criteria, *modularity and smoothness*, into the objective function of an ILP and let it search the set $\Psi(G)$ of all clusterings completely.

smoothness as ILP goal

We will briefly review such an ILP formulation for *smooth* clusterings of dynamic graphs in the *online* setting, and then we shall build upon it when designing a full *offline* formulation

in the the next but one subsection. For this part we consider the set V of nodes to be fixed; a generalization is easy, but requires the generalization of distance measures for clusterings to that situation. While we canonically do that by simply ignoring inserted/removed nodes for the affected *time step* in Section 4.3, we refrain from touching the subject here, as obvious solutions are at hand but disrupt notation.

4.5.1.1 An ILP for Bicriterial *Online* Dynamic Updates

***Smoothness* as Part of the Objective Function.** We have seen in Section 2.4 how *performance* can be shaped into a linear objective function for ILPs. Consider now the common formula for *performance* and that of the distance measure Rand $\mathcal{R}$ for sets, which we discussed in Section 2.6.2:

$$\text{perf}(\mathcal{C}) := \frac{m(\mathcal{C}) + \overline{m}^c(\mathcal{C})}{\frac{1}{2}n(n-1)} \qquad \mathcal{R}(\mathcal{C},\mathcal{C}') := 1 - \frac{2(n_{11}+n_{00})}{n(n-1)} = 1 - \frac{n_{11}+n_{00}}{\frac{1}{2}n(n-1)} \tag{4.5.1}$$

Recalling that n_{11} and n_{00} are the numbers of node pairs which are clustered together in both clusterings and separately in both clusterings, respectively, we can take a step back and see that *performance* and the Rand measure are essentially the same; remember that Rand is a *distance* measure, hence the negative sign.

$\mathcal{R}$ as a linear objective function

Thus, along the lines of the same deduction for *performance* in Section 2.4.1, we can see that with the help of decision variables $\mathcal{X}^{\text{er}}$ we can write a linear objective function for $\mathcal{R}(\mathcal{C},\mathcal{C}')$. Let $\mathcal{X}^{\text{er}}$ describe $\mathcal{C}'(G')$ and let δ_{uv}[19] describe $\mathcal{C}(G)$.

$$\begin{aligned}
\mathcal{R}(\mathcal{C},\mathcal{C}') &= 1 - \frac{2}{n\cdot(n-1)} \cdot \sum_{u<v} \Big(X^{\text{er}}_{uv} \cdot \delta_{uv} + (1 - X^{\text{er}}_{uv}) \cdot (1 - \delta_{uv}) \Big) \\
&= 1 - \frac{2}{n\cdot(n-1)} \cdot \sum_{u<v} \Big(2 \cdot X^{\text{er}}_{uv} \cdot \delta_{uv} - X^{\text{er}}_{uv} - \delta_{uv} + 1 \Big) \\
&= \text{const}_1 - \text{const}_2 \cdot \sum_{u<v} \Big((2\cdot\delta_{uv} - 1) \cdot X^{\text{er}}_{uv} \Big)
\end{aligned} \tag{4.5.2}$$

Together with *modularity*'s contribution to the objective function we can now formulate a *bicriterial* objective function, with a scalable trade-off between quality and *smoothness*. Note that we can do away with constants in the individual objective functions, but we should remember them, when we scale this trade-off. In the following the parameter b is used to **b**alance this trade-off.

bicriterial objective function

$$\begin{aligned}
bicrit(G,C,G') &= \text{mod}_{\text{ILP}}(C') - b \cdot \mathcal{R}_{\text{ILP}}(C,C') \\
&= \sum_{u<v} \left(A(u,v) - \frac{\deg(u)\cdot\deg(v)}{2\cdot m} \right) \cdot X^{\text{er}}_{uv} + b \cdot \sum_{u<v} (2\cdot\delta'_{uv} - 1) \cdot X^{\text{er}}_{uv} \\
&= \sum_{u<v} \left(A(u,v) - \frac{\deg(u)\cdot\deg(v)}{2\cdot m} + 2b\cdot\delta'_{uv} - b \right) \cdot X^{\text{er}}_{uv}
\end{aligned} \tag{4.5.3}$$

Similar formulations are possible for other distance measures for clusterings, the crucial point is, that they must allow a linear objective function. For measures based on *counting pairs* (Section 2.6.2) such as the Jaccard index, or the Fowlkes-Mallows measure (see [218] for more on these indices), it is easy to see that this is possible. Measures based on *overlaps* or on *entropy* cannot be integrated with this setup, they require more variables such as the sizes of each node's previous cluster.

[19] We again use a shorthand for Kronecker's symbol $\delta_{uv} = 1$ iff $\mathcal{C}(u) = \mathcal{C}(v)$.

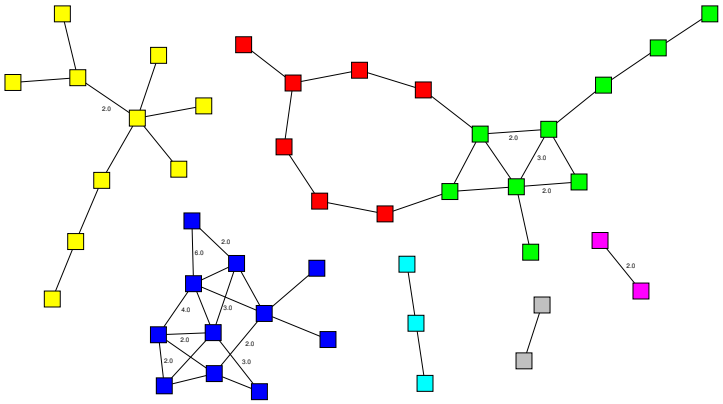


Figure 4.5.2. A snapshot of the email graph for September 2007 containing 3 professors' chairs; 24h lifetime.

Experiments on Bicriterial Updates. As mentioned earlier, our ILPs for *modularity*-optimization cannot handle more than about 200 nodes, and thus we did not include this approach in Section 4.3, as we saw that the modified objective function did not speed up things. In this separate experimental setting we use an excerpt of the dynamic graph of email communication, please refer to Section 5.1.1 for more information on this instance. In a setup similar to that described in Section 4.3.4.1, we use three complete chairs, which yields about 50 nodes. We observe emails for the duration of one week (in September '07) and use *batch updates* of size $b_{\text{batch}} = 10$. Emails are assigned a lifetime of 24 hours, such that after this period a *weight decrease* event δ^- for the edge between the two communicating nodes is triggered.

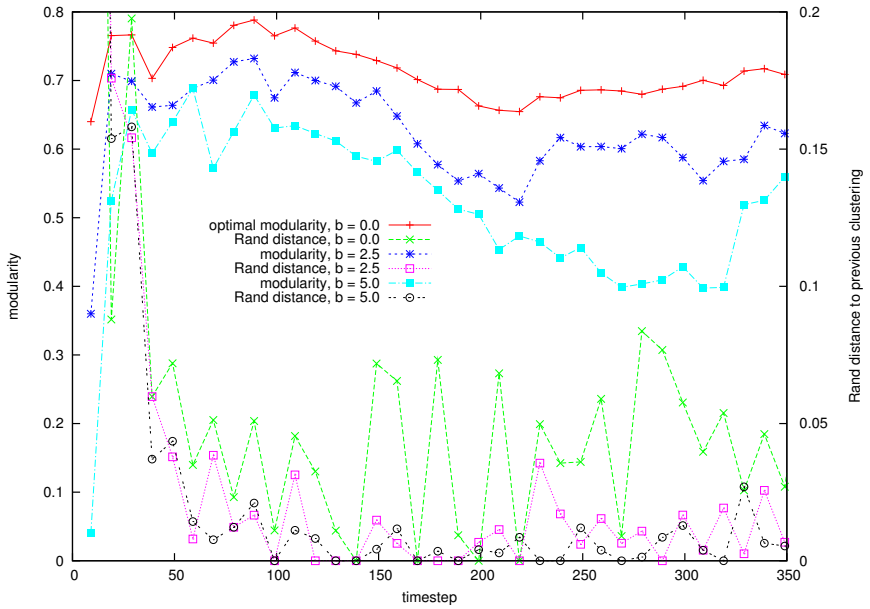


Figure 4.5.3. This plot shows the *modularity* and Rand distance measure for the clustering of a fraction of the email graph with a connection lifetime of 24 hours. Different balance factors have been applied, the *batch size* is 10.

modularity vs. $\mathcal{R}$, balanced via b

Figure 4.5.3 shows *modularity* and Rand distance (with regard to the previously clustered *time step*) with $b = 0.0, 2.5, 5.0$. The index curve for $b = 0.0$ (red) shows the optimal *modularity* values, thus the corresponding Rand distance (green) is not used by the optimization. By concept, the best index values as well as the highest distance measure values are attained. The *modularity* values for b set to 2.5 and 5.0 shrink significantly, as do their corresponding distance measures. Observe how a higher influence of the Rand distance leads to less distance measured to previous clusterings, but also lowers the modularity value. Trivially, this is explained by the fact that necessary optimizations in order to construct optimal *modularity* clusterings cannot be executed, as minimizing the *temporal cost* is more important and hence

nodes are anchored to their old clusters. This underlines the assumption, that the two criteria are opposing and shows that b has the anticipated effect, allowing for an explicit decision about what criterion is more important. High computational demands render this approach useful for only very specific applications, and not for broad practice.

4.5.1.2 An ILP for Bicriterial *Offline* Dynamic Clustering

bicriterial formulations

Translating the postulations that drive the above approach into a classic *offline* setting, yields numerous possible formalization, of which we just name a few in order to get an impression. Suppose a sequence $\mathcal{G} = (G_0, \dots, G_{t_{\max}})$ of graphs is given:

1. Among all sequences $\zeta = (\mathcal{C}_0, \dots, \mathcal{C}_{t_{\max}})$ of clusterings of $\mathcal{G}$ with $\mathcal{C}_i(G_i)$ optimal regarding *quality*, find the sequence ζ_{smooth} that minimizes $\sum_{i=1}^{t_{\max}} \mathit{distance}(\mathcal{C}_{i-1}, \mathcal{C}_i)$.
2. Among all sequences $\zeta = (\mathcal{C}_0, \dots, \mathcal{C}_{t_{\max}})$ of clusterings of $\mathcal{G}$ with $\sum_{i=1}^{t_{\max}} \mathit{distance}(\mathcal{C}_{i-1}, \mathcal{C}_i) \leq D$, find the sequence ζ_{good} that maximizes $\sum_{i=0}^{t_{\max}} \mathit{quality}(\mathcal{C}_i)$.
3. Is there a sequence $\zeta = (\mathcal{C}_0, \dots, \mathcal{C}_{t_{\max}})$ of clusterings of $\mathcal{G}$ such that $\forall i : \mathit{quality}(\mathcal{C}_i) \geq \alpha(\mathcal{C}_i^{\text{optimal}})$ and $\forall i \geq 1 : \mathit{distance}(\mathcal{C}_{i-1}, \mathcal{C}_i) \leq \beta$?
4. Among all sequences $\zeta = (\mathcal{C}_0, \dots, \mathcal{C}_{t_{\max}})$ of clusterings of $\mathcal{G}$ find the sequence ζ_{best} which optimizes $\alpha \sum_{i=0}^{t_{\max}} \mathit{quality}(\mathcal{C}_i) + \beta \sum_{i=1}^{t_{\max}} \mathit{distance}(\mathcal{C}_{i-1}, \mathcal{C}_i)$.

The diversity of possible bicriterial formulations should become obvious, and thus the choice must ultimately depend on the application; furthermore optimality will in practice have to give way to "the best one's algorithms can do". Although this does not diminish their interestingness, in particular not in a theoretical view, we shall not dwell long on these formulations. In this subsection we will describe a flexible framework of constraints for bicriterial integer linear program formulations of *offline* clustering problems which focus on quality and *smoothness*.

concatenate t_{max} ILPs

A Simple Concatenation of *Online* ILPs. Suppose we have an *offline* dynamic graph $\mathcal{G} = (G_0, \dots, G_{t_{\max}})$, and desire a *smooth* dynamic clustering $\zeta = (\mathcal{C}_0, \dots \mathcal{C}_{t_{\max}})$ of $\mathcal{G}$. All we need to do is to use the concept of the preceding subsection and concatenate $t_{\max}$ ILPs, one for each *time step*, by the additional distance terms in the objective function and add up all the objective functions. The crucial point where this approach fails is the fact that in Section 4.5.1.1 above, we exploit that δ_{uv} is a constant. In an *offline* problem no single clustering is fixed and ready to build upon, which renders δ_{uv} a variable such that the objective function is no longer linear.

Xor-variables

Xor-Variables. The solution to this problem is the introduction of a simple set of additional variables. All these new variables $\mathcal{W}$ need to do is evaluate an Xor expression "between" two *time steps* t and $t+1$: n_{11} and n_{00} in $\mathcal{R}$ is contributed to by all pairs $\{u, v\}$ of nodes for which $W_{uv}(t) := X_{uv}^{\text{er}}(t)$ Xor $X_{uv}^{\text{er}}(t+1)$ equals True. Thus let $\mathcal{W}$ be the set of Xor-variables that mediate between the $t_{\max}+1$ *time steps* for each of which we set up an ordinary, static ILP ILP_i for clustering exactly as in Section 2.4.1, using the respective G_i and distinguishing $\mathcal{X}^{\text{er}}$ by the timestamp in brackets as in "$X_{uv}^{\text{er}}(t)$":

ILP_i

$$\mathcal{W}(\mathcal{G}) := \{W_{uv}(t) : \{u, v\} \in \binom{V}{2}\}, 0 \leq t < t_{\max} \tag{4.5.4}$$

$$\text{with} \quad W_{uv}(t) = \begin{cases} 0 & \text{if } X_{uv}^{\text{er}}(t) = X_{uv}^{\text{er}}(t+1) \\ 1 & \text{otherwise} \end{cases} \tag{4.5.5}$$

$W(t) := X^{er}(t)$ Xor $X^{er}(t+1)$

Having $\mathcal{W}$ encode this required Xor-expression can be enforced by the following constraints:

$$\underbrace{\forall\{u,v\} \in \binom{V}{2}, 0 \leq t < t_{\max} : \begin{cases} W_{uv}(t) \leq 2 - X^{\text{er}}_{uv}(t) - X^{\text{er}}_{uv}(t+1) \\ W_{uv}(t) \leq X^{\text{er}}_{uv}(t) + X^{\text{er}}_{uv}(t+1) \\ W_{uv}(t) \geq X^{\text{er}}_{uv}(t) - X^{\text{er}}_{uv}(t+1) \\ W_{uv}(t) \geq -X^{\text{er}}_{uv}(t) + X^{\text{er}}_{uv}(t+1) \end{cases}}_{\text{Xor constraints for } \mathcal{W}}, \tag{4.5.6}$$

$$\underbrace{\forall\{u,v\} \in \binom{V}{2}, 0 \leq t < t_{\max} : W_{uv}(t) \in \{0,1\}}_{\textit{integrality} \text{ constraints for } \mathcal{W}} \tag{4.5.7}$$

size of ILP

For a dynamic graph $\mathcal{G}$ we thus require a total of $|\mathcal{W}| = t_{\max} \cdot \binom{n}{2}$ Xor variables and $4t_{\max} \cdot \binom{n}{2}$ constraints, in addition to the $t_{\max} + 1$ static ILPs which each contribute $\binom{n}{2}$ variables and $3\binom{n}{3}$ constraints (not counting integrality constraints). For a triple x, y and c, with a binary equation $c = x$ Xor y aimed at by Equations 4.5.6, total enumeration of all binary values shows that this equation is correct:

total enumeration

	x	y	c	$c \leq 2 - x - y$	$c \leq x + y$	$x - y \leq c$	$y - x \leq c$
correct	0	0	0	$0 \leq 2$	$0 \leq 0$	$0 \leq 0$	$0 \leq 0$
	1	1	0	$0 \leq 0$	$0 \leq 2$	$0 \leq 0$	$0 \leq 0$
	1	0	1	$1 \leq 1$	$1 \leq 1$	$1 \leq 1$	$-1 \leq 1$
	0	1	1	$1 \leq 1$	$1 \leq 1$	$-1 \leq 1$	$1 \leq 1$
wrong	1	0	0	$0 \leq 1$	$0 \leq 1$	$\mathbf{1 \not\leq 0}$	$-1 \leq 0$
	0	1	0	$0 \leq 1$	$0 \leq 1$	$-1 \leq 0$	$\mathbf{1 \not\leq 0}$
	0	0	1	$1 \leq 2$	$\mathbf{1 \not\leq 0}$	$0 \leq 1$	$0 \leq 1$
	1	1	1	$\mathbf{1 \not\leq 0}$	$1 \leq 2$	$0 \leq 1$	$0 \leq 1$

An Overall Objective Function. Suppose we now set up each ILP_i for $0 \leq i \leq t_{\max}$ and their respective objective functions for quality q_i (as, e.g., in Equation 2.4.3). An objective function that (solely) minimizes the Rand distance of the two consecutive clusterings $C(i)$ and $C(i+1)$ is (compare to Equations 4.5.1 and 4.5.2):

q_i of ILP_i

h_i

$$h_i := 1 - \frac{\sum_{u<v}(1 - W_{uv}(t))}{\frac{1}{2}n(n-1)} \tag{4.5.8}$$

Putting things together we can now set up an overall objective function incorporating q_i for each ILP_i and the available h_i. We refrain from discussing a balance factor β between these two parts and just state the conceptual objective function:

final objective function

$$\text{objective} \quad := \quad \underbrace{\sum_{i=0}^{t_{max}} q_i}_{\textit{snapshot quality}} \quad - \quad \beta \cdot \underbrace{\sum_{i=0}^{t_{max}-1} h_i}_{\textit{temporal cost}} \tag{4.5.9}$$

The prohibitive size to which this ILP for *offline dynamic graph clustering* quickly grows gives this subsection a theoretical character. However, it is important to see that—given some decision about β—optimality can at least be modeled.

Variant Optimization Goals. We have arrived at a solution for the problem statement given in item 4. Although we shall not elaborate on this, observe how the setup described in this subsection can easily be altered as to accommodate, e.g., the problem statement given in item 2 and—given one first computes static optima—item 1 or item 3; we can simple use h_i and q_i in appropriate constraints.

4.5.2 Time-Expanded Clustering

time-expanded clustering

We coin as *time-expanded clustering* an approach towards *time-dependent clustering* which in a single clustering step on a *time-expanded* dynamic graph identifies structural groups *and* their evolution over time. As a simple example consider the temporal evolution of a recommendation system for books as used for example by `Amazon.com`. In particular, consider the book "The Lord of the Rings" by J.R.R. Tolkien. Before the major success of the movie adaptation, the book belonged to the niche community of fantasy literature. Afterwards it belonged to a much broader community including other popular bestsellers. One might even infer from sale statistics that the book does no longer belong to the fantasy community, due to a higher purchase correlation with books like "The Da Vinci Code"[20] or even "The Shell Seekers"[21].

tracking over independent clusterings ...

Why Not Static-Comparatively? Suppose now we took recommendations advertised by `Amazon.com` and built a network of recommendations with books as nodes and an edge (or some edge weight) for each recommendation within, say, a timeframe of one month.[22] Building a graph for each month yields a dynamic graph, and clustering each such graph independently yields a reasonable sequence of clusterings. However, inferring temporal trends from a sequence of independent clusterings requires us to somehow "follow" changing clusters over time—a highly nontrivial task, especially as *smooth* dynamics are not at all enforced.

... via node overlap ...

In fact, [132] follows this approach in an attempt to track communities in the network of publications (nodes) and citations (edges), compiled from the `CiteSeer` [5] database. We mentioned this work in Section 4.1 but point out the efforts the authors make to render their time-spanning clustering *smooth* and meaningful. The first step is to find a static clustering per *time step* which is "stable". The authors do this by computing several static clusterings for each *time step*, each one based only on a 95% fraction of the nodes, and then taking clusters from the first clustering as a "natural community" if their best match (measured by a match coefficient as in Equation 1.2.14) is greater than some threshold. Given such natural communities for each *time step*, the authors then again find the best match for a subset of interesting clusters in neighboring *time steps*. A very similar approach is followed in [182]. Here, the authors exploit the neat fact that CPM, the method used for computing the static clusterings[23] of *time steps*, behaves as a *coarsenig technique* of two clusterings, if applied to the "union graph" of the two *time steps*. They thus use the same algorithm for each static clustering step and—in a special way—for following clusters over time. This particular property of their method at the same time exhibits a behavior that can oppose intuition: Clusters can never dissolve and be divided up into other growing clusters.

... or via common coarsenings

While both of these approaches are reasonable (given one agrees with their static clustering techniques) and find their individual solutions to the problem that a dynamic clustering requires some stability, in essence they patch together multiple unrelated static clusterings. We already stated arguments against this approach in the introduction of Section 4.1. This is exactly the point we wish to address (see Figure 4.5.1): we aim at a technique which can informally be specified as follows:

informal aims of time-expanded clustering

5. Find a sequence ζ of clusterings of $\mathcal{G}$ which[24]
   (a) *(quality)* yields a good static clustering $\mathcal{C}_i$ per *time step*,
   (b) *(smoothness)* enforces a *smooth* transition between the clusterings of subsequent *time steps*, allowing users to retain their *mental map* and their derivations made of previous *time steps*,

[20] Dan Brown, 2003, Transworld Publishers, UK Bantam Books (UK) and Doubleday Group (US)
[21] Rosamunde Pilcher, 1987, Thomas Dunne Books / St. Martin's Press
[22] Such networks have been used and visualized, e.g., in [100].
[23] The Clique Percolation Method [76], see Section 2.1.
[24] Our specification of goals strongly resembles that made in [53] (see Section 4.1.1) and has in parts been motivated by that work, in spite of differing techniques and applications.

(c) *(noise removal)* is robust against noise and outliers in single *time steps*, and

(d) *(cluster correspondence)* tells us without further degrees of freedom (i.e., uncertainty) which clusters in neighboring *time steps* belong to each other and thus reveals trends and breakpoints.

patching has downsides

Any patching technique will, on the one hand, suffer from outliers which in some *time step* differ in terms of their neighbors, and on the other hand, introduce much freedom in the patching stage where matchings between *time steps* are sought. While this freedom can be useful, it potentially is an additional bias and can introduce systematic distortion. Needless to say, a non-expert user can hardly do well when confronted with any parameteric choices for two stages.

time-expanded graphs

The *Time-Expanded Graph* and *Clustering*. *Time-expanded graphs* have been used in different fields and for different purposes, however, what these approaches had in common was to model a dynamic problem instance in a way that adequately incorporates the relationship between *time steps*. They have their roots in flow computations in dynamic graphs. In 1962, Ford and Fulkerson [88] first published this approach in a textbook. The rough problem setting they address is to find a schedule of commodity transfers between the nodes of a network such that in a given timeframe the maximum commodity is transported between the designated source and sink, while changing traversal times and maximum capacities of edges are respected. To give one example application, in route planning, and in particular for timetable information problems of public transport systems, *time-expanded graphs* are used to model instances in a way which allows for the application of established tools for shortest path queries. Roughly speaking this directed graph contains as nodes copies of stations for each point in time where a train arrives or leaves. Then these nodes are connected by reachability, i.e., edges between copies of the same station represent switching trains and edges between different stations represent taking a train. We recommend the works [73, 188] for recent advances and a good overview on this topic. In order to describe the adaptation we propose for graph clustering, we start with a formal definition of the terms *time-expanded graph* and *time-expanded clustering* in Definition 4.5.

$\mathcal{G}_{TE}$

Definition 4.5 *Given a finite graph sequence $\mathcal{G} = (G_0, \dots, G_{t_{\max}})$, with $G_i = (V_i, E_i, \omega_i)$ and an integer T we define the* time-expanded graph $\mathcal{G}_{TE} = (\mathcal{V}, \mathcal{E}, \tilde{\omega})$ *by*

$$\begin{aligned}
\mathcal{V} &:= \{(v,i) \mid 0 \le i \le t_{\max}, v \in V_i\} \\
\mathcal{E} &:= \mathcal{E}_{\text{graph}} \cup \mathcal{E}_{\text{time}} \quad \textit{with} \text{ intra- } \textit{and} \text{ inter-time } \textit{edges:} \\
&\quad \mathcal{E}_{\text{graph}} := \{\{(u,i),(v,i)\} \mid v,w \in V_i, \{u,v\} \in E_i, 0 \le i \le t_{\max}\} \\
&\quad \mathcal{E}_{\text{time}} := \{\{(v,i),(v,j)\} \mid v \in V_i, v \in V_j, |i-j| \le T\} \\
\tilde{\omega}((u,i),(v,j)) &:= \begin{cases} \omega_i & \textit{on } \mathcal{E}_{\text{graph}} \textit{ (with } i = j \textit{ and } u \neq v\textit{)} \\ \omega_{inter} & \textit{on } \mathcal{E}_{\text{time}} \textit{ (with } i \neq j, u = v\textit{, } \omega_{inter} \textit{ is still to be defined)} \end{cases}.
\end{aligned}$$

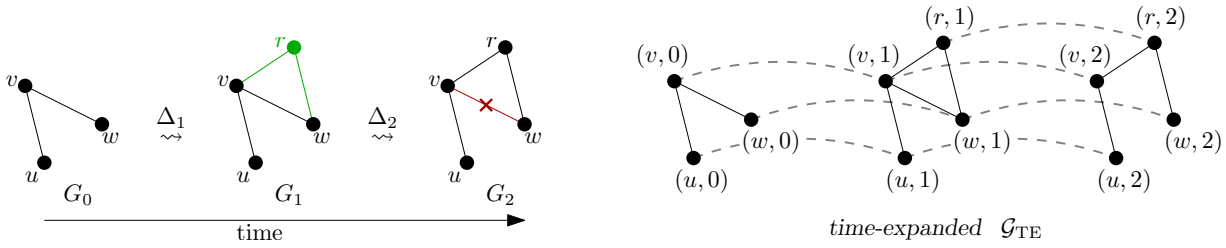
$\mathcal{C}_{TE}$ *slice*

Given a finite graph sequence $\mathcal{G}$, a time-expanded clustering $\mathcal{C}_{TE}$ *of $\mathcal{G}$ is a clustering of $\mathcal{G}_{TE}$. The clusterings $\mathcal{C}_i(G_i)$ which $\mathcal{C}_{TE}$ canonically induces are called* slices.

Figure 4.5.4 is a simple example of a *time-expanded* graph. A good clustering $\mathcal{C}_{\text{TE}}$ of a dynamic graph $\mathcal{G}_{\text{TE}}$ will fulfill formulation 5 above. Note that this definition turns a blind eye on three subtleties one has to keep in mind when turning towards an actual implementation: (i) we assume in the definition of $\mathcal{E}_{\text{time}}$ that nodes which exist in different *time steps* of $\mathcal{G}$ have the same identifier in all *time steps*; (ii) *inter-time edges* may be allowed to span between *time steps* with a distance greater than one, thus the *span* T has to be specified; (iii) the weight function ω_{inter} for *inter-time edges* is a delicate degree of freedom for a *time-expanded graph* and requires a particularly thoughtful definition. In some sense the definition of ω_{inter} inherits the burden of the patching stage discussed above. However, ω_{inter} is defined *before*

span T *ω_{inter}*

clustering and thus snapshots are not clustered agnostically of each other; moreover ω_{inter} can be defined with more basic and reliable assumptions than a patching stage. We will revisit these items later but for now settle with the above definition for simplicity.



(a) Three *time steps* of a dynamic graph $\mathcal{G}$ **(b)** Graph $\mathcal{G}_{\text{TE}}$ of $\mathcal{G}$, *inter-time edges* are dashed.

Figure 4.5.4. An example of a dynamic graph and its *time-expanded* graph

We can now state a framework algorithm for *time-expanded clustering* in Algorithm 32. In the experiments that follow we successfully used variants of greedy *modularity* agglomeration (see Section 2.2.5) as algorithm $\mathcal{A}$. Anticipating later results on ω_{inter}, we found that the *cosine similarity* of the adjacency vectors of (v,t) and $(v,t+1)$ is an excellent starting point, and setting $T=1$ is a good choice. Two add-ons are at hand: Suppose $\mathcal{G}_{\text{TE}}$ becomes prohibitively large for $\mathcal{A}$, we can simply use a *sliding window* in line 2. Points in time where the clustering seems to undergo a transition can easily be identified by measuring distances between consecutive clusterings, see Section 2.6 for such measures.

Algorithm 32: Time-Expanded Clustering

Input: *Dynamic graph* $\mathcal{G}$, ω_{inter}, static clustering algorithm $\mathcal{A}$

1 Construct $\mathcal{G}_{\text{TE}}$ from $\mathcal{G}$ using ω_{inter} `// see Def. 4.5`
2 $\mathcal{C}_{\text{TE}} \leftarrow \mathcal{A}(\mathcal{G}_{\text{TE}})$ `// actually cluster`
3 $C_i \leftarrow \mathcal{C}_{\text{TE}}(\mathcal{G}_{\text{TE}})|_{G_i}$ `// obtain slices`

sliding window

transition of the clustering

A typical real-world setting, which is illustrated in Figure 4.5.5 below, are the collaboration dynamics in science. Researchers usually start out working in a narrow field, then, by interdisciplinary commitment, they enter other communities, possibly migrating to another field entirely. In the 80's *Thomas Lengauer* concerned himself with algorithmic graph theory,

a migration in science

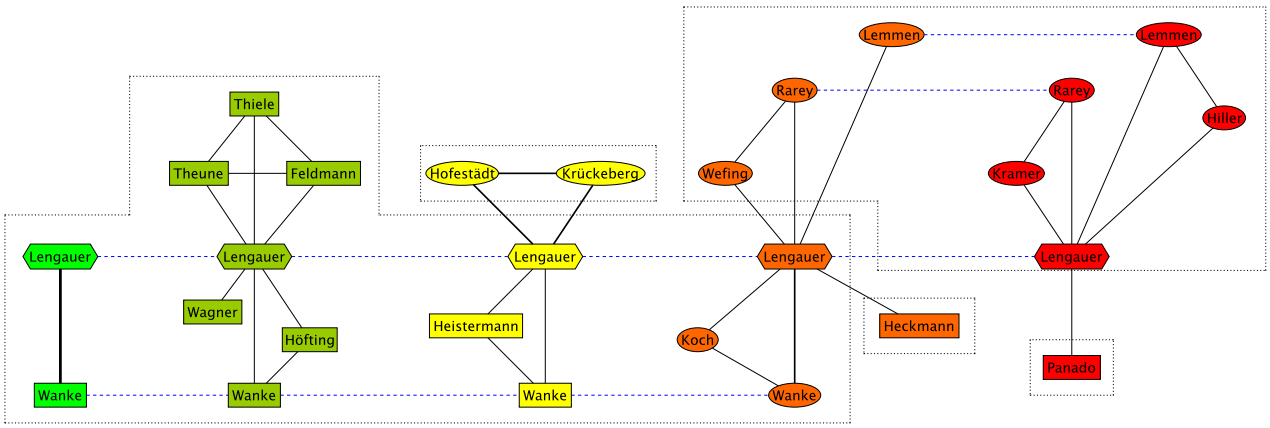


Figure 4.5.5. An excerpt from the collaboration graph of T. Lengauer is shown for the years 1988, 1992, 1993, 1996, and 1998. Round shapes correspond to publications in the field of biology, and rectangular shapes to those in computer science. The *time-expanded graph* is clustered by the large boxes and *inter-time edges* are dashed.

focusing on planarity. However, in the early 90's he began collaborations in the field of bioinformatics and biology and at the present he is an established scientist of the bioinformatics

community. The clustering of the time-expanded collaboration graph reveals this development, identifying the community transition in the 90's. Next we shall apply the concept of *time-expended* graph clustering to a larger data set.

4.5.3 *Time-Expanded* Clustering of the Email Graph

monthly time steps

In the following we will discuss a case study, where *time-expanded clustering* is used. In particular, this is the changing network of email contacts at KIT's Fakultät für Informatik, viewed at 11 monthly *time steps*. For technical details about this instance, please refer to Section 5.1.1. What this section cannot accomplish is a systematic comparison between *time-expanded clustering* techniques and other approaches for *offline* clustering tasks of dynamic graphs. The fact that a clear and consolidated formalization of an aim, which is valid and usable beyond a few specific applications, has not yet emerged in the literature, leaves too many dimensions to explore in a systematic study. However, it very well serves as a proof of concept and shows the potential of this approach.

aim: item 5

Aim. On a high level, the aim of this study is simple: In the changing network of email contacts we want to identify clusters and track them over time. Recalling our guidelines in item 5, we will need to focus on properties (a)-(c), *cluster correspondence* (d) is implicitly solved by concept. Since for our instance, we know about an underlying *ground truth* as a reference clustering—the structure of chairs of the department—we can compare our findings with this information.

known ground truth

4.5.3.1 Specification of the Method

only greedy

Remember that the static clustering technique employed in *time-expanded clustering* has so far been wildcarded entirely. We performed experiments with the methods MCL, ICC and the greedy maximization of *modularity* (greedy in the following); all three algorithms yielded reasonable results, in accordance with their respective peculiarities, but for brevity we will confine our study to the latter algorithm. For the same reason we restrict ourselves to measurements by *modularity*.

mostly modularity

The one major degree of freedom of a *time-expanded* graph is the definition of ω_{inter} (see Definition 4.5), the edge weight functions for *inter-time edges*. We restrict our insights to choices of ω_{inter} which yielded reasonable results. Needless to say, there is a dependency between the design of ω_{inter} and the chosen clustering algorithm; this is not desirable but inevitable. Two more parameters of the model were addressed: we varied the *span* T of *inter-time edges* and an edge-pruning threshold p; we also normalized ω_i on $\mathcal{E}_{\text{graph}}$. We restricted ourselves to the setup where *inter-time edges* only exist between copies of identical nodes. In order to get a first impression of the data set, Figure 4.5.6 depicts the canonic clusterings by reference and by *time steps* of $\mathcal{G}_{\text{TE}}$.

parameters: ω_{inter},span T, p

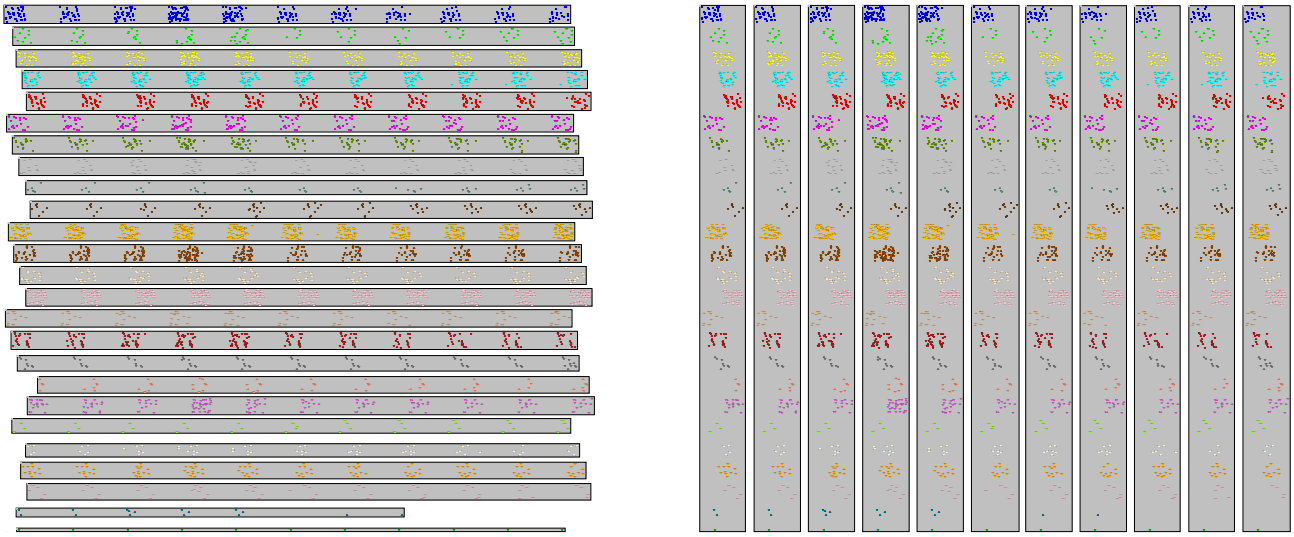
4.5.3.2 Parametric choices

ω_{inter} = const.

Baseline Setups. Starting with the simplest of setups, we compute $\mathcal{G}_{\text{TE}}$ and $\mathcal{C}_{\text{TE}}$ as follows: using as *time steps* the unmodified one-month snapshots and $\omega_{\text{inter}} = \alpha =$ constant. More precisely we use $\omega_{\text{inter}} = 1, \ldots, 10$, $T = 1, \ldots, 9$, and evaluate the effect of a pruning threshold $p = 1, \ldots, \omega_{\text{inter}} - 1$ for *noise removal*. Simply put, all edges with weight less than p are removed from the graph. In these baseline setups we observed the expected dependency of the shape of the clustering on α and T: Large *inter-time* weights and large *spans* T lead to a *tall time-expanded* clustering with hardly ever changing *slices*. A small but non-zero value for p does not only increase the *modularity* values of the *slices* but increase their similarity to the reference—corroborating that the reference relies on stronger *intra-chair communication*. For brevity we skip our intermediate experiments and rather report on the setup we recommend, as it worked best for us and follows a reasonable intuition.

edge pruning

parameter testing



(a) Gray boxes represent the *time-expanded* (pseudo-) clustering by reference, i.e. chairs.

(b) Gray boxes represent *time steps*.

Figure 4.5.6. Two canonic clusterings for $\mathcal{G}_{\text{TE}}$, by reference and by *time step*. Relative node positions are preserved throughout the *time steps*. These can be viewed as the two extreme cases of $\mathcal{C}_{\text{TE}}$, where either *inter-time* or *intra-time weights* dominate.

The Final Setup. The range of original edge weights reveals a few outliers, probably due to some *carbon-copying* habit or automatism. In order to reduce this effect we introduce the following normalization of the *intra-time edge weights* of each individual *time step* to the interval $[0, 1]$:

normalize ω_i

$$\omega_i^{\text{no}}(e) = \frac{\log(\omega_i(e))}{\log(\omega_i^{\max})} \qquad \text{on } \mathcal{E}_{\text{graph}}, \text{ with individual } \omega_i^{\max} \text{ for each } \textit{time step } i \tag{4.5.10}$$

Fixed weights for *inter-time edges* ignore the role nodes have in *time steps*. If such a role changes between two *time steps*, the corresponding edge should have a low weight. The *role* a node is in fact a concept from network analysis, however, for our purpose a rather superficial measure suffices. We adapt the *cosine similarity* (see Section 1.2.3) to our setting and—intuitively speaking—define *inter-time weights* for $\mathcal{E}_{\text{time}}$ as the *cosine similarity* of the corresponding (labeled) adjacency vectors. More formally this evaluates to the following, somewhat clumsy formula:

$\omega_{inter} \sim$ cosine

$$\omega_{\text{inter}}^{\cos}((v,i),(v,j)) := \frac{\sum_{u \in V_i \cap V_j} (\omega_i((v,i),(u,i)) \cdot (v,j),(u,j))}{\sqrt{\sum_{u \in V_i} (\omega_i((v,i),(u,i)))^2} \cdot \sqrt{\sum_{u \in V_j} (\omega_j((v,j),(u,j)))^2}} \tag{4.5.11}$$

It strictly follows the intuition of the *cosine similarity*, such that it yields 1 if the neighborhood of v is the same in both *time steps* i and j, and 0 if in the two *time steps*, the copies of v do not share a single *intra-time adjacency* to some pair of nodes $(u,i),(u,j)$, respectively.

4.5.3.3 Results and Measurements

Quality Measurements and Pruning. First, we list quality indices of the clusterings of the 11 *time steps*, still with a varying pruning threshold $0 \leq p \leq 0.45$ for noise reduction. Interestingly, for $p = 0$ the average weight of *inter-* and *intra-time edges* is 0.69 and 0.22, respectively, while for $p = 0.45$ these values are 0.84 and 0.54. This indicates that a node's behavior hardly varies and corroborates the relevance of the reference clustering, but also

impact of p on $\mathcal{G}_{TE}$

exemplifies noise reduction. Tables 4.5.1 and 4.5.2 list the average (wrt. p and time) values of the reference clustering defined by the chairs, and the *slices* of the *time-expanded* clustering, respectively.[25] Observe how the *slices* surpass the already excellent quality of the reference.

index	cov_ω	mod_ω	$\text{icc}_\omega^{\text{av}}$
1. quartile	0.9125	0.8373	0.7218
3. quartile	0.9659	0.8832	0.8514
minimum	0.8171	0.7472	0.5836
maximum	0.9839	0.8986	0.8717
average	0.9308	0.8548	0.7835

Table 4.5.1. Quality of the reference clustering (chairs of the department) per *time step*

index	cov_ω	mod_ω	$\text{icc}_\omega^{\text{av}}$	$\mathcal{NVD}^*$
1. quartile	0.9246	0.8582	0.7738	0.4123
3. quartile	0.9693	0.9104	0.9640	0.3319
minimum	0.8569	0.7668	0.7180	0.2557
maximum	0.9927	0.9276	0.9953	0.5799
average	0.9422	0.8770	0.8799	0.3753

Table 4.5.2. Quality of $\mathcal{C}_{\text{TE}}$'s *slices* and the distance to the reference clusterings, $\mathcal{NVD}^*$, an asymmetric version of Equation 2.6.2.

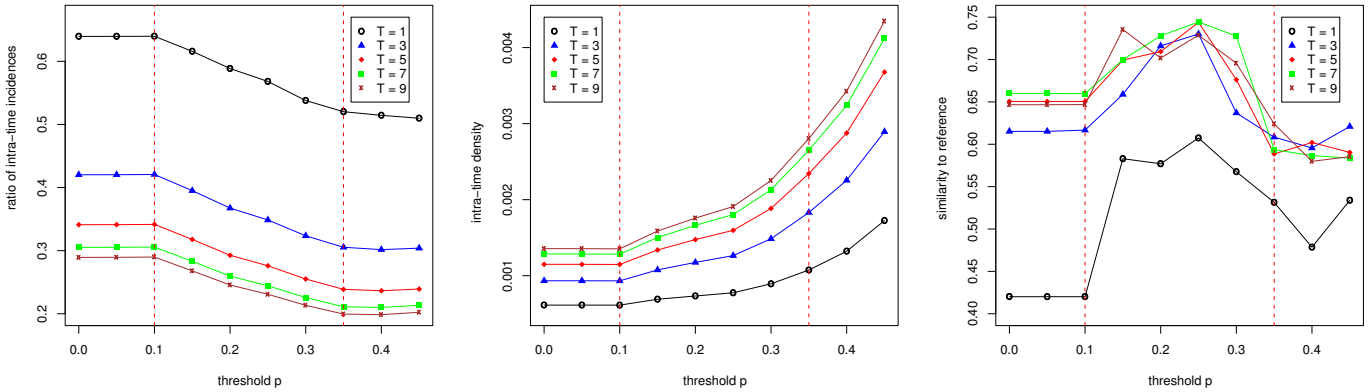
slices vs. reference

Distances are low, but far from 0, as sometimes two or three reference clusters are summarized into one cluster of a *slice*. We briefly elaborate on p, as a pruning threshold as high as possible can significantly contribute to rendering an instance computable. Figure 4.5.7 shows how properties of $\mathcal{G}_{\text{TE}}$ and $\mathcal{C}_{\text{TE}}$ change with p and T. As a side note, we removed disconnected nodes from graphs, since these had no communication for a whole month. The rough insights are as follows. Only $T = 1$ lets *intra-time adjacencies* play a significant role, otherwise we get very close to the reference. High values of p densify the *time steps* and reduce the distance to the reference. Omitting much of our further studies and discussions we conclude that $T = 1$ is a good choice for our instance and that only a small threshold $p \leq 0.15$ is feasible; we will continue using $T = 1$ and $p = 0$ without further notice, in order to be "closer" to a parameter-free procedure.

high T ⇒ close to ref.

high p ⇒ close to ref.

use $T = 0$, $p = 0$



(a) Ratio of incident *intra-time weight* to sum of node weights

(b) Ratio of *intra-time weight* to number of *intra-time node pairs*

(c) Similarity of *slices* to reference (as a similarity measure $1 - \mathcal{NVD}^*$)

Figure 4.5.7. Influence of the pruning threshold p and the *span* T on various properties

example excerpt

Figure 4.5.8 shows an excerpt of the *time-expanded* clustering $\mathcal{C}_{\text{TE}}$ of $\mathcal{G}_{\text{TE}}$ which nicely shows a point of transition in the structure of the network: The beige cluster dissolves into the gray one, and the red one instantaneously joins a green cluster. Figure 4.5.10 shows this transition in the full context of $\mathcal{C}_{\text{TE}}$.

[25] We here use a set overlap measure (instead of a *graph-based* measure) since in this case the partition of the set of nodes is more important than the edge structure, an asymmetric version is chosen since the reference is a known *ground truth*, which others have to match to, not vice versa. See Section 2.6.2 for other aspects.

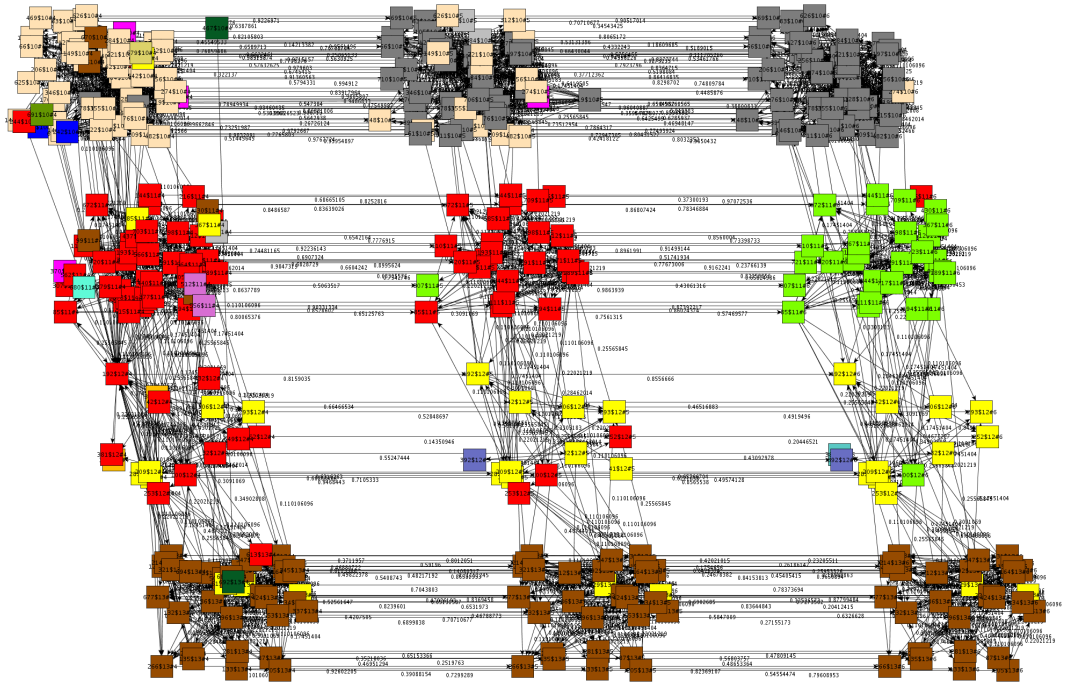


Figure 4.5.8. An excerpt of $\mathcal{G}_{\text{TE}}$ with color-coded $\mathcal{C}_{\text{TE}}$, $T = 1$ and $p = 0$; vertical slabs are *time steps*, horizontal slabs are chairs of the reference. Observe, e.g., the transition of nodes from the beige cluster to the gray cluster, for $T = 3$ this phenomenon vanishes.

slice vs. static

Comparison to Static Clustering. We can now review the *slices* of $\mathcal{C}_{\text{TE}}$ in the light of individual static clusterings of each *time step*. Figures 4.5.9a and 4.5.9b precisely show the expected trade-offs: In terms of *modularity*, maximizing *modularity* individually works best, only closely followed by the *slices* and finally the reference. On the other hand, the *slices* are much closer to the reference, i.e., *smoother*. Summarizing, we obtain exactly the desired behavior.

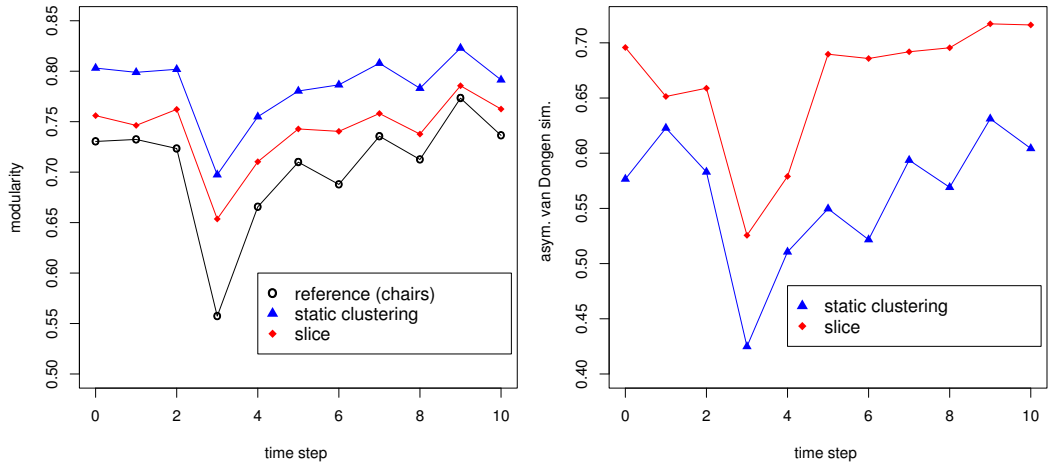
slices are smooth and good

discussion of appropriateness

Discussion of $\mathcal{C}_{\text{TE}}$. In terms of measurable quantities such as quality, distance to the reference and to static baseline clusterings, and *smoothness* our approach *without any* peculiar parametric settings appears to work excellently: *span* $T = 1$, pruning threshold $p = 0$, ω_{inter} based on *cosine similarity* of adjacencies and ω_i logarithmically scaled and normed to $[0, 1]$. But can our approach answer our initial question from Section 4.5.3? Are the identified and tracked *slices* of $\mathcal{C}_{\text{TE}}$ and their transitions meaningful? Within the allowable bounds of data privacy protection we shall now discuss a few insights about the background of changes and transitions in the clustering.[26]

We enumerate the chairs (horizontal slabs) in Figure 4.5.10 from top (0, dark blue) to bottom (25, cyan singleton) and the *time steps* (vertical slabs) from left (0) to right (10); please note that Figure 4.5.10 is displayed sideways. Chairs 5 and 8 (light gray) are part of the same institute, thus the rigorous togetherness. Chair 3 joins this cluster (light gray) as it is closely affiliated to that institute. An organizational change split the common institute of chairs 6 and 11 approximately at *time step* 2. Instead of parting into different clusters, 6 and 11 together with the small chair 9 join the cluster (light green) of chair 1. "Chairs" 12 and 4 are no real chairs but central institutions with many uniform, non-collaborational contacts, thus they are happy to join any attracting cluster. Chair 4 is part of the aforementioned red cluster but after the transition of that to the light green cluster, chair 4 becomes an individual

[26] See Section 5.1.1 for more on this issue.



(a) *Modularity* values for the three clustering: reference, static, *sliced*

(b) Similarity of the static clusterings and of the *slices* to the reference, measured by $\mathcal{NVD}^*$ similarity

Figure 4.5.9. A comparison of individual static clusterings and *slices* shows the expected behavior: In terms of *modularity* the *slices* closely follow individual clusterings while being much closer to the reference.

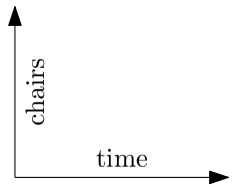
cluster—due to little density inside the new, larger cluster. Similarly, chair 12 in parts belongs to the red cluster but after the transition at *time steps* 5 and 6, it focuses more on the yellow cluster, which instead loses chair 14 due that chair's co-founding the moss green cluster with chair 0 at *time step* 8—we suspect a common project of which the kick-off would coincide with the start of a new semester.

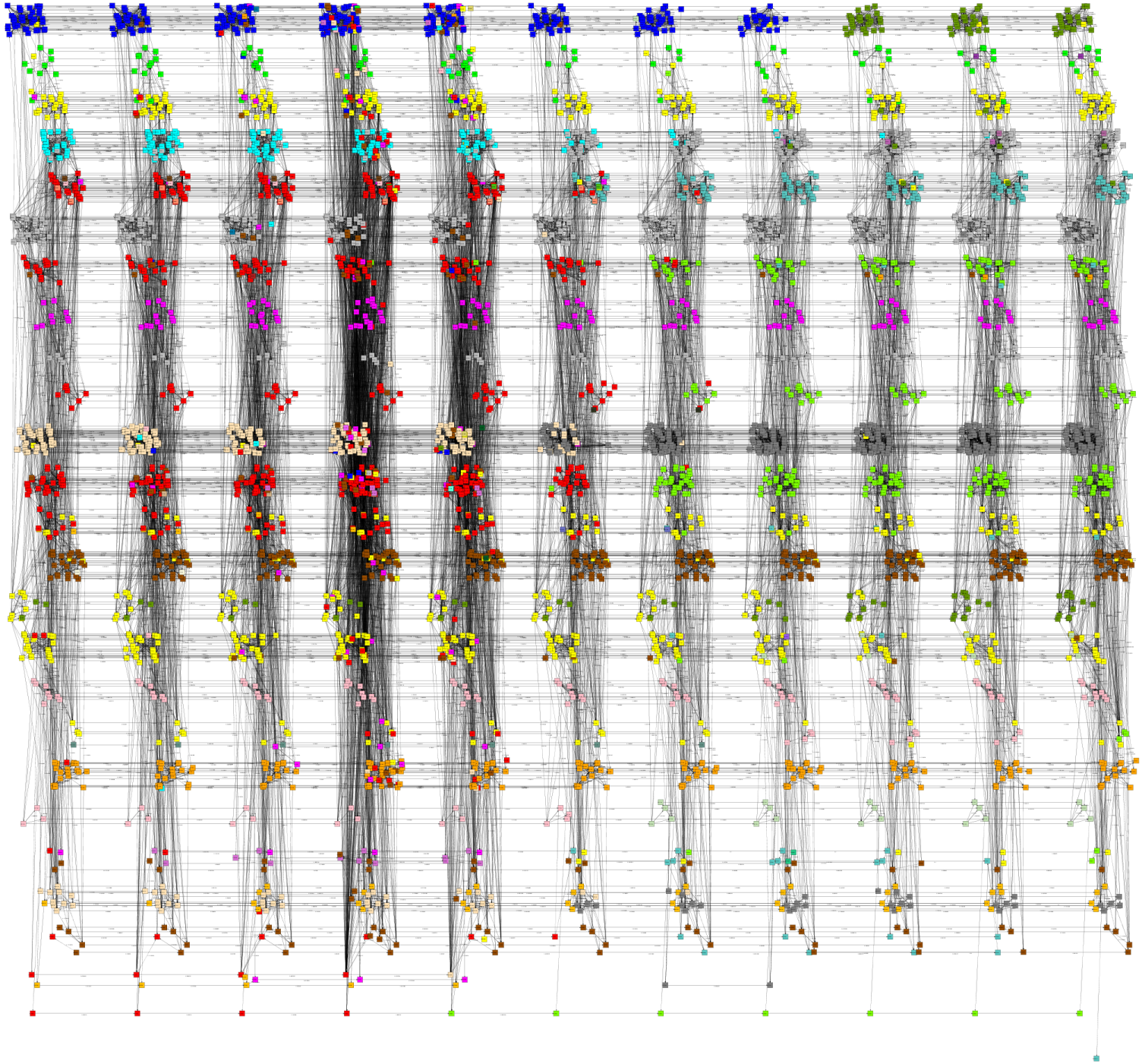
4.5.3.4 A Final Word on *Time-Expanded* Graph Clustering

making a case for $\mathcal{C}_{TE}$

We started our experimental case study of the *time-expanded* graph with the simplest of setups. We learned that having to adjust many parameters—especially if the employed clustering method adds to these—can work and finally yield good results, but requires laborious tuning. We found that using parameter-free *cosine similarity* for ω_{inter} nicely conforms to intuition, and that some reasonable scaling of *intra-time edge weights* to the interval $[0, 1]$ should be done. Then, however, the most user-friendly setup worked very well: no pruning ($p = 0$), only the most basic $\mathcal{G}_{\text{TE}}$ (*span* $T = 0$) and parameter-free greedy as the employed clustering algorithm. The obtained clustering $\mathcal{C}_{\text{TE}}$ and its *slices* did not only describe the peculiarities and the dynamics of this real-world network very well, they even compete with individual static clusterings in terms of quality. Instances of this size pose no problem for today's hardware, but are unsolvable by an ILP as described in Section 4.5.1.2. Concluding, *time-expanded clustering* solves the task stated in item 5 and shows much potential to work off-the-shelf for reasonably modeled dynamic graphs.

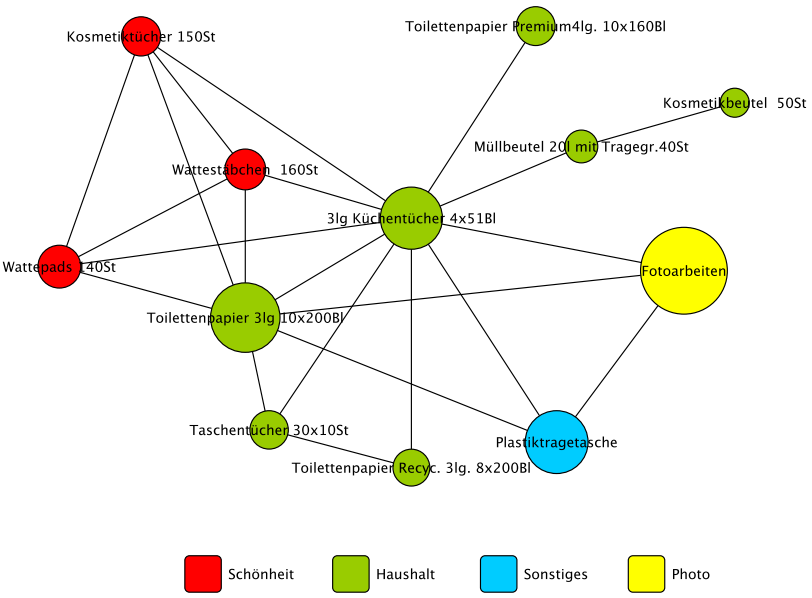
Figure 4.5.10. The *time-expanded* graph of 11 *time steps* of the email graph. Each *time step* (vertical slab, if using the page sideways) represents one month of communication, ω_{inter} mimics the *cosine similarity* of neighborhoods and no pruning is done. The identified *time-expanded* clustering is shown by node colors. The reference, given by the chairs of the department, is indicated by horizontal slabs. Relative positions of nodes are preserved over time, such that individual persons can be tracked. Needless to say, graph-drawing conventions for esthetic layouts had to stand back. However, the initial layout of each individual chair is drawn with force-directed methods; nodes that join the graph at later *time steps* are also placed using force-directed methods on their first appearance. The high *intra-time density* for $t = 3$ coincides with Christmas.





Chapter 5

Epilogue



Clustering *dm*'s range of products by customer preference yields—among others—the sales-dominating cluster which accommodates the infamous shopping bag ("Plastiktragetasche"). Budget cosmetics seem to be inevitable for a *dm*-shopper, they are tightly connected to everyday products such as toilet tissue and photo services; not to *premium* toilet tissue, though. Some sets of real-world data are true jewels and sources of fascination.

Contents

Section 5.1

Data Sets and Applications

Having the computer scientist understand the problem setting of the application works better than the other way round.

(Richard Manning Karp, Touring Award winner, comment after plenary talk at WADS'09)

WHAT IS MORE IMPORTANT FOR AN EXPERIMENTAL EVALUATION: millions of systematically generated random graphs or one or two real-world instances?[1] Certainly, both need to contribute to most studies. Numerous real-world data sets have their part in this thesis, either visibly in several examples or in background feasibility studies. In this informal section, some details about them, so far left open, shall be explained. However this section is by no means intended to be comprehensive, and should just serve as an overview. Despite of the fact that the network of Autonomous Systems in the Internet (AS) is used in many examples and case studies of this thesis, details on this graph are omitted, as it is discussed in-depth in many studies, we recommend [100] for a first reference with many further pointers.

AS omitted

Negotiating and gathering such data, understanding it, filtering, converting and modeling it into a meaningful graph was a fascinating yet time-consuming part of my work—and often not less challenging and relevant than actually clustering the graph. I owe my thanks to quite a few people who helped me obtain these useful data sets and I shall seize the opportunity to express my gratitude in the corresponding subsections below.

5.1.1 Email Contacts at KIT's Fakultät für Informatik

It was in the late summer of 2006 when my former colleague Martin Holzer and I sat together and pondered possible source of real-world clustered graphs, naturally accompanied by Cassandra warnings. This data set of email traffic is the first fruit of our efforts to get into contact with such sources. Klaus Scheibenberger, head of the department which manages the technical infrastructure of KIT's Fakultät für Informatik (ATIS), very quickly agreed to the general idea. Quickly after, Olaf Hopp, head of IT-services, arranged an automated script which filters, anonymizes and summarizes the logs of the central email server[2], and makes them available to us on a daily basis.[3] Table 5.1.1 shows an excerpt of such a log, the *email id* serves to identify emails with multiple recipients.

email server logs

collaboration graph

The obvious interpretation of such data as a graph is to let nodes represent persons, i.e., email accounts, and edges represent emails exchanged between the two incident nodes;

[1] By concept, the answer to this question is exclusively known by your reviewer.

[2] This central email server routes emails for about two thirds of the department, some institutes and chairs are independent.

[3] My sincere thanks to Olaf Hopp and Klaus Scheibenberger, who did not only take on the work to technically set up the automatic tool which deviates the logs to us, but also took the time to discuss our results with us and maintained the tool throughout all technical reorganization at ATIS.

timestamp	sender	sender chair	recipient	recipient chair	email id
2006-09-13	59	20	60	16	1GNPYp-L9
2006-09-13	61	6	16	6	1GNPaI-Ec
2006-09-13	61	6	62	6	1GNPaI-Ec

Table 5.1.1. Excerpt of a summarized email log

the weight of an edge then encodes the number of exchanged emails, in order to avoid unnecessary parallel edges. In order to actually obtain a graph, we collect the logs of a certain timeframe and interpret them as an implicit edge list. If *time steps* represent a duration of one month or more, in *dynamic graphs* we usually restrict *time steps* to nodes which partake in at least one email. Keeping disconnected nodes, that communicated in earlier *time steps*, would let the set of nodes grow steadily. An example of a static graph which consists of the members of one chair and represents the accumulated communication of four months is shown in Figure 5.1.1. To model a *fully dynamic* graph that does not only accumulate edge weight, we assign a *lifetime* to the contribution of an email: A sent email contributes to the tie between two nodes for, say, 72 hours, then the email expires and the two nodes need to communicate again, in order to maintain their level of connectedness. In this setting we again let disconnected nodes drop out, in order to also have full node dynamics.

implicit edge list

lifetime

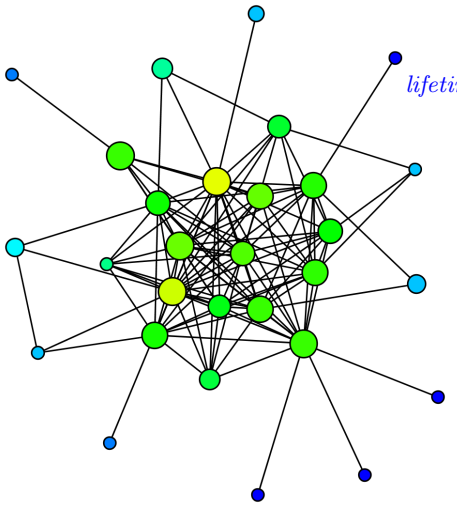


Figure 5.1.1. The color of a node represents its degree (from blue to red) and its size is prop. to its *betweenness*.

Three facts make this data set particularly precious, (i) it is very reliable, (ii) it is fully dynamic and (iii) there is a *ground-truth* clustering which underlies it: it is reasonable to assume that the subdivision of this department into chairs yields a clustering, since collaboration and communication between the members of the same chair can be expected to be more regular than for different chairs. We multiply confirmed this in previous sections, e.g., in Section 4.5.3.3, and Figure 4.5.9 in particular. However, this assumption must be handled with care, as there is no *ground-truth* distribution which supports this, just common sense—and quite a few quality measures.

Since September 2006 we have now collected more than three years of communication amounting to about 400K emails yielding 500K pairs of sender and recipient, of which about 400 arrive per day. On weekdays, a daily snapshot involves between 300-500 different nodes depending on the time of year, and on weekends or general holidays this number can reach below 50. On a microscopic scale, dynamics are, among other things, due to colleagues discussing, announcing talks or instructing student workers, or to student workers who join the network for only a brief duration etc. Although the *ground-truth* clustering defined by the chairs is rather stable, there are various points that add macroscopic dynamics to the microscopic noise: New projects kick off that require increased collaboration between participating chairs, in turn, others conclude. An even larger and more far-reaching impact has the head of a chair, if she decides to leave. Such an event can change the focus of a whole chair and even motivate organizational changes. In fact we have seen such a transitions, as the generation of founding fathers of the department gradually retired.

three years

0.5M emails

micro- and macrodynamics

Applications. The email graph permeates this whole thesis and was invaluable to it. We used it in the evaluation of *modularity* (Figure 2.3.18), as an example instance for clustering distances (Section 2.6.4.3), we showcased *LunarVis* with this network (Section 3.2.3) and then used it in the three fine-grained dynamic studies on *modularity*-driven- (Section 4.3.4.1), *min-cut tree*- (Section 4.4.5) and bicriterial ILP clustering (Section 4.5.1.1) and finally we investigated the dynamics of this network with *time-expanded clustering* (Section 4.5.3).

5.1.2 Sales and Products of *dm*

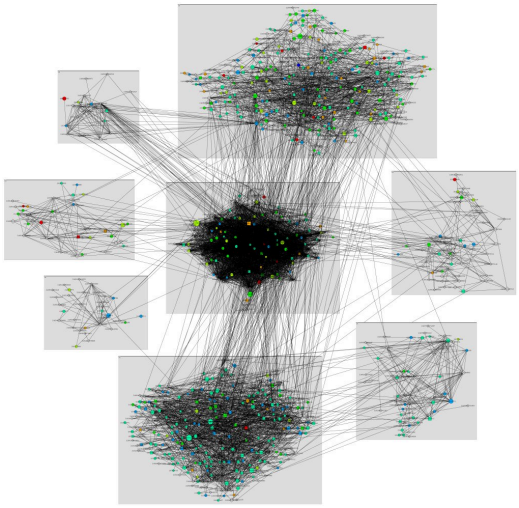


Figure 5.1.2. A graph of the customers of one store, edges represent a similar shopping behavior. The lower cluster, containing mostly bluish (young) nodes is dominated by baby food.

dynamic customer profiles

Getting into contact with the German drugstore chain *dm* was the boldest and least promising of our plans for real-world data sets. However, with some persistence, my former colleague Martin Holzer finally managed to arrange for a meeting with Erich Harsch, the executive of *filiadata*, which is the IT subsidiary company of *dm*. To our surprise he was immediately convinced by our plans and we quickly agreed on a loose collaboration which allowed us to use a subset of their collected sales data for our evaluations. The formidable policy of *dm* and the kind staff of *filiadata*, and Andreas Gessner in particular, made it possible for us to learn much about our algorithms and about the issues which are relevant for *filiadata* and *dm*. My special thanks go to Michael Martin of *filiadata*, who was the person in charge of our collaboration and who was always available for discussions and friendly help.

Among the many interesting issues we addressed, our collaboration with *dm* and *filiadata* recently motivated a diploma thesis about how graph clustering techniques can be used to model customer profiles from an evolving set of sales data. Luckily, a very creative and diligent student, Selma Mukhtar [169], chose to conduct this diploma thesis. Her results ultimately helped to convince *filiadata* to engage her straight after she finished her thesis.

5.1.3 Literature Databases

collaboration and citation graphs

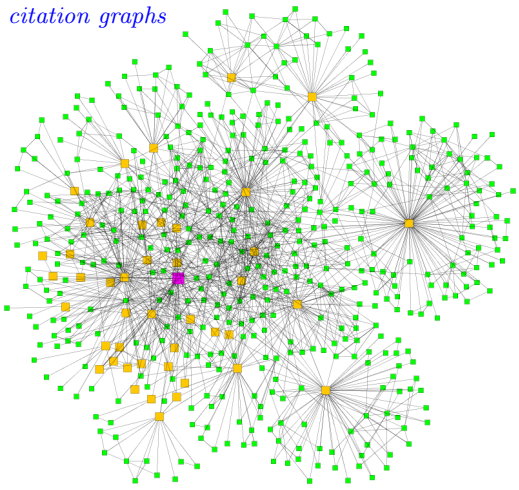


Figure 5.1.3. D. Wagner (purple), her neighbors (yellow) and her 2-hop neighbors (green).

A data source which is well represented in scientific literature is scientific literature itself. Digitized catalogs and databases are readily available through resources such as Citeseer [5], DBLP [4] or arXiv [7], which strongly facilitate research. Networks based on either the collaboration of authors or on the citations between publications have been studied in many investigations and have also been used as benchmark sets for graph algorithms. Such graphs usually feature a rather skew degree distribution, tending towards a power-law.

We often dealt with the data sets of Citeseer and DBLP, and I would like to thank my trusty student worker Hai Wei for the excellent tools he designed and programmed for extracting all kinds of networks from these sets; in fact we also used these tools for the *dm* data and the patent data. In a graph based on scientific collaboration the nodes represent researchers and edges model the strength of their collaboration by somehow using the number of co-authored publications. The graph shown in Figure 5.1.3 shows the 2-hop neighborhood of my advisor, Prof. Dorothea Wagner, in the collaboration graph according to a snaphot of the DBLP database from 2007.

5.1.4 Patent Registrations

technological trends

We recently started a collaboration with the Institute for Economic Policy Research (IWW) at KIT. Among other things, the IWW concerns itself with measurements and predictions of technological trends in research and economy. Does spacial closeness foster a visible synergy

between related technological fields which are, e.g., worked on by neighboring companies? Is there a critical threshold for the accumulation of technological knowhow, beyond which a company attracts related research? One way to address such questions is on the basis of patent registrations, see [124] for an overview. Patent registrations are categorized by the International Patent Classification (IPC) of the World Intellectual Property Organization (WIPO). This classification scheme assigns to an "inventive thing" a rough *section*, e.g., "section A – human necessities", and subclassifies the invention by a fine hierarchy of a total of roughly 70.000 IPC keys into *class*, *subclass*, *group* and *subgroup*. A full key could finally be "G01N 33/483" which stands for "Physical analysis of biological material" The IPC is a tool for quickly finding out whether some idea has already been registered or not, since for such a (very typical) query, the exact name of a potential patent registration cannot be known, but if it exists, it must be registered under a specific IPC key.

WIPO, IPC

An interesting point is, that an invention must be registered under all keys it is considered to be relevant for, patent law does not protect the idea with respect to any other keys. Thus, a patent registration which uses several keys implicitly indicates a tie between those keys. We use this fact to build a network of IPC keys, roughly on the *subclass* level, where edge weights represent the ratio of patents two keys share. A cluster in this graph of IPC keys thus corresponds to a set of keys of which patent registrations often use several at once—yielding technological clusters. Using, e.g., one-year snapshots of patents this graph is actually dynamic. We used this data set as a second test environment for *time-expanded* clustering. A strong *smoothness* is necessary to keep the *slices*, which suggest a classification, from changing every year. Figure 5.1.4 is an excerpt of such a *time-expanded* graph.

similarity of IPC keys

time-expanded clustering

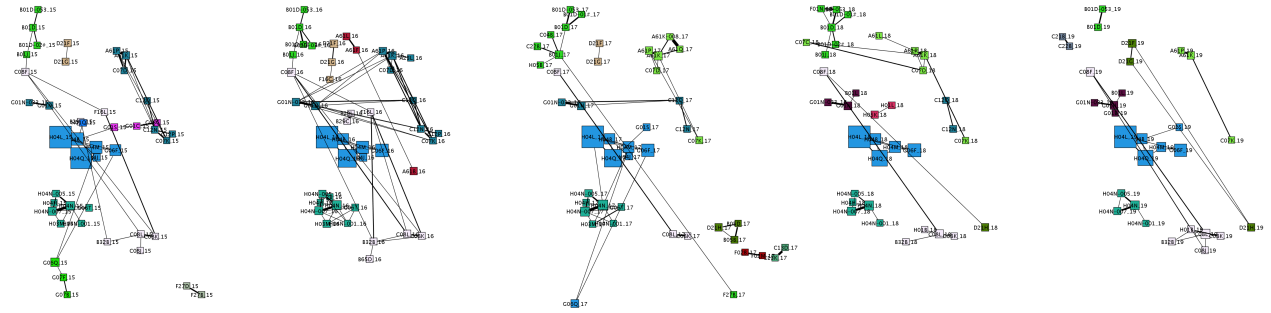


Figure 5.1.4. A 5-year excerpt of the *time-expanded* graph of IPC keys (*inter-time* edges are hidden), based on Finnish inventions registered at the European Patent Office. Node colors represent a *time-expanded* clustering, identified as in Section 4.5.3. Nodes scale by the number of patents registered under their key. Guess what areas the dominating blue nodes (prefixes H04L and H04Q) cover!

5.1.5 Online Shopping

Due to data privacy protection I do not know much background about the two data sets Stefanie Nagel dealt with in her diploma thesis [170] in a cooperation with *epoq knowledgeware* and its co-executive Michael Bernhard. This smart and diligent student addressed us with a proposal to do her diploma thesis on the topic of graph clustering algorithms for the products of an online shop, based on shopping cart data. We agreed on a topic which also included some theory, of which perhaps the most interesting part is a study on how network analysis can help to find the "right" clustering algorithm for a data set. Several graph models were considered until we decided on edge weights encoding the probability that two products are bought together, given at least one of them is bought (rel. to the Jaccard coeff., see Equation 1.2.12).

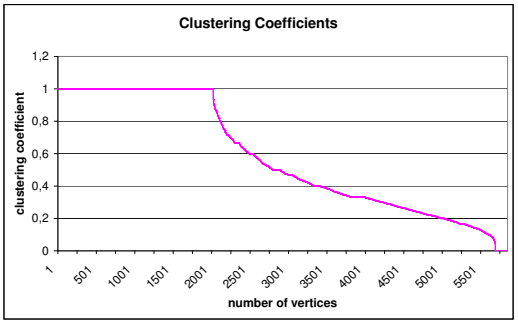


Figure 5.1.5. Distribution of *clustering coefficients* in the graph

5.1.6 Lipidomics

lipids

lipidome

metabolome

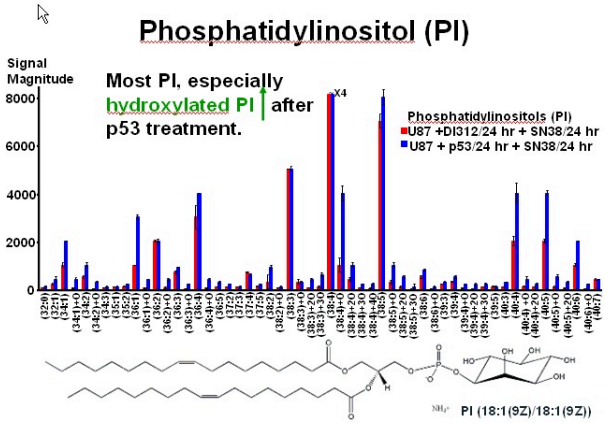
Biotechnology in general is a rapidly growing field of which *lipidomics* is a sub-discipline that deals with *lipids*. *Lipids* are naturally-occurring molecules and include all fats, fat-soluble vitamins, diglycerides, and many others, whose main biological functions are structurally composing cell membranes, storing energy, and signaling. *Lipids* have been found to be good indicators of an organism's reaction to changes and diseases. The *lipidome* of an organism or of some part of it is the signature of lipids of that entity and is itself one member of the *metabolome*, together with *sugars*, *nucleotides* and *amino-acids*. Roughly speaking, the *metabolome*, consisting of these four major groups of molecules, is the signature of functional molecules of an organism. According to [224], *lipidomics* can be defined as the large-scale study of pathways and networks of cellular lipids in biological systems.

mass spectrometry

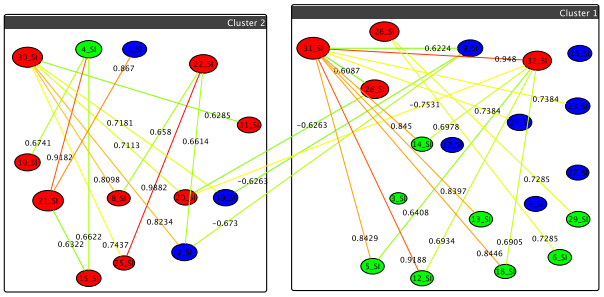
glioblastoma

therapy

Recent advances in high precision measurements of the *metabolome*, and of *lipids* in particular, by the method of *mass spectrometry* employing *Fourier transform ion cyclotrone resonance* (FT-ICR MS) opened up new ways to address questions about the *lipidome*, see [225] for an overview of this matter. One particular topic a group of researchers from Tallahasse, Florida is concerned with, is measuring and understanding how an organism (i.e., its *lipids*) infected by cancer reacts to curative treatments. Anke Meyer-Bäse from Florida State University and Mark R. Emmett and Huan He from the National High Magnetic Field Laboratory contacted us with a proposal to collaborate on the evaluation of their measurements. More precisely, the aim is to model the *lipids* as a graph based on similar behavior concerning treatments, and to use graph clustering in order to find clusters of *lipids* that exhibit a consistent behavior. Our current focus is on *glioblastoma* (a highly invasive brain tumor) cells and their treatment with *cytotoxic* (toxic to cells) *chemotherapy* "SN-38", a *wild-type tumor suppressor protein* "p53" (gene therapy), combinations of these and control setups. FT-ICR MS basically measures how many infected cells undergo *apoptosis* (die) under the treatment. Figure 5.1.6 depicts the main ingredients of this collaboration, biochemical experiments and their measurements (a) and computational methods for the interpretation of the gathered data (b). Many thanks to Anke Meyer-Bäse for the energy she put into getting this collaborative project started, and for translating between the two worlds of mathematics and biology.



(a) Measurement profiles for the group of *Phosphatidylinositol*, *polar* (negatively charged) lipids, with (blue) and without (red) p53-treatment. The chemical formula describes a *Phosphatidylinositol*.



(b) Two clusterings, by colors and boxes, of the *Gangliosides*, according to their behavior when treated with or without p53, respectively. Edges and sizes encode differences between the two measurements.

Figure 5.1.6. A glimpse of our collaboration with *lipidomics* research, *mass spectrometry* measurements (a) and a graph visualization which helps to understand reactions to treatments and *lipid* correlations (b).

Section 5.2

Side Notes

A narrow strip of flattened soil,
a beginning and an end,
and in between, an ocean of adventure.

(Homage to singletrails,
BIKE Magazin, January 2008)

DISTRACTIONS ABOUND for any PhD student. But while distractions are usually pleasant or even welcome, other duties are not. Among other things, I collected these oddities in this clearly dispensable and informal section, if only for my personal records. I also take the opportunity to mention and appreciate the students I advised in the past years.

Distractions and Curiosities

Graph Drawing Competitions. Infamous for their time-consuming but addictive nature, the competitions of the annual *International Symposium on Graph Drawing* (*GD*), had me fascinated thrice during my time as a PhD student with its not-so-serious problem statements. Contests range from specific tasks on given data sets to freestyle drawing with special appeal. At GD'05 Michael Baur, Marco Gaertler and I presented a tool for exploring how actors migrate between movie genres over time [32] (Figure 5.2.2a).

At GD'07 Thomas Schank made awesome dynamic visualizations of the graph of professors at Karlsruhe's Fakultät für Informatik. Annual snapshots of these scientists and their surrounding coauthors were taken and used to model ties between them. A smooth dynamic visualization then showed the evolution of this collaboration network. My modest part in this project was extracting and preparing the annual snapshots from the DBLP [4] database (see Section 5.1.3). Figure 5.2.1 shows one such snapshot.

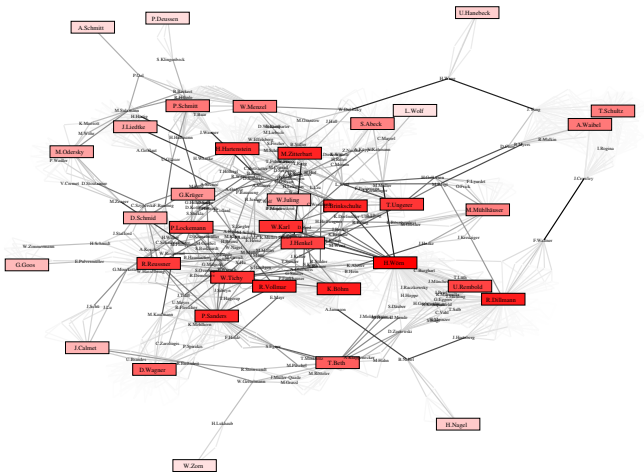
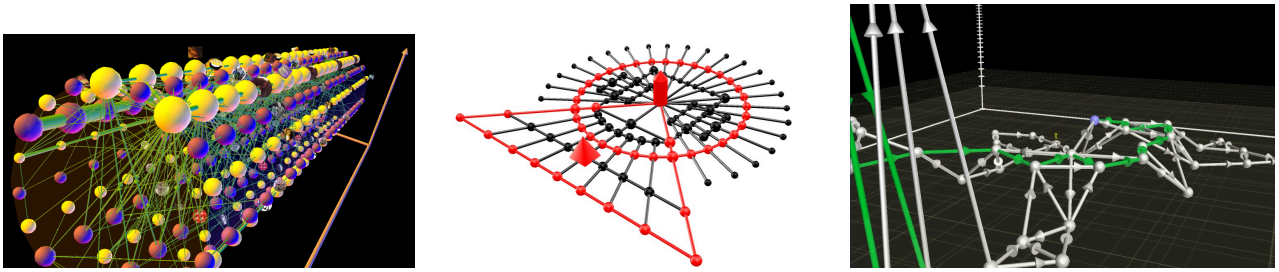


Figure 5.2.1. Snapshot of a dynamic visualization of Karlsruhe's CS professors and their collaboration, runner-up, social network graphs competition, GD 2007

Algorithmus der Woche. In fact not exclusively for the GD contest '06 and its logo (see Figure 5.2.2b), together with Steffen Mecke, I developed *Flow Commander* [119], a tool for visualizing graph algorithms in 3D, as shown in Figure 5.2.2c. The initial incentive was to find a good means to explain the Push-Relabel algorithm [112] to students. This then lead to us participating in the GD contest, but also to a nice tool which we designed for pupils within the project "Algorithmus der Woche" (algorithm of the week). This project took place



(a) Exploring movie genres over time and tracking the migration of actors through them, contribution to the evolving-graph drawing competition using the Internet Movie Database, honorable mention, GD 2005.

(b) This graph drawing depicts the abstracted surroundings of Karlsruhe Castle, it became the ubiquitous logo of the conference and is engraved on a plaque in the pedestrian zone of Karlsruhe, GD 2006.

(c) Algorithm visualization in 3D with *Flow Commander*, never again struggle explaining the Push-Relabel algorithm, freestyle contest, honorable mention, GD 2006; contribution to "Algorithmus der Woche".

Figure 5.2.2. Graphs in 3D, the contributions to the International Symposia on Graph Drawing and to the project "Algorithmus der Woche" are still at use for teaching the Push-Relabel algorithm.

in the context of the "Jahr der Informatik" (year of informatics, 2006) and explained in an understandable way many basic algorithms to pupils, both on an online website, where our tool can be played with, and in a book [214]. The final result was a framework for graph algorithms, and it is still used at least once per year—during the lecture on maximum-flow algorithms. In fact, Berthold Vöcking's idea to cast the collected contents of the project "Algorithmus der Woche" into a book was a great success, work on an English version was initiated shortly after the German book had been published. It was an interesting task for Steffen and me to compose our chapters on the Push-Relabel algorithm for maximum-flows in a way which non-computer scientists could understand, and it was encouraging to see the success of the book.

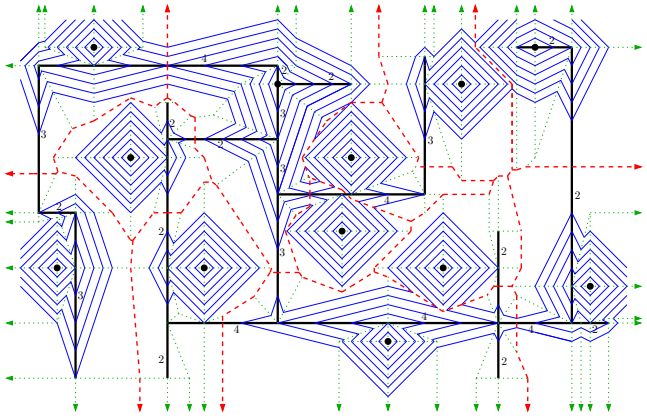


Figure 5.2.3. Half-finished *city Voronoi diagram* (red) of 12 point sites (disks) in the L_1 plane, augmented by an arrangement of fast line segments (bold). Cover of the proceedings of VD'05 and logo of the Algorithms Group at TU/e, Eindhoven. Blue shapes depict a wavefront, and green arrows indicate its combinatorial shape.

Selected Trivia. During my untroubled first year, a drawing from my diploma thesis [114], which at the same time became my first publication [121, 122, 123], was elected to adorn the cover of the proceedings of the 2nd *International Symposium on Voronoi Diagrams in Science and Engineering* (VD'05), where, by a strange coincidence, I also received a *best presentation award*. Later, the drawing (in fact, part of it) became the logo of the *TU/e Algorithms Group* at Technische Universiteit Eindhoven, where my former advisor Alexander Wolff moved to. Figure 5.2.3 shows the drawing which exemplifies a wavefront expansion.

During my time as a PhD student, my teaching duties comprised four exercise courses, two seminars, two practical courses, six diploma theses and five student research projects. I was in charge of editing and preparing six books for publication, wrote parts of seven milestone, activity or roadmap reports and one final report for the European Commission within the project "DELIS"[4]. Not counting workgroup-internal business, I traveled 12 different countries, gave 18 talks and attended 23 conferences and project meetings.

[4]FET open project within FP-6-IST of the EU: 'Dynamically Evolving, Large-scale Information Systems"

Students of Mine

During my years as a PhD student, I met and advised many smart students. Putting aside the time invested to help, teach and advise them, I owe my thanks to many of them for promoting research topics by their ideas and creativity and for relieving me of numerous burdens.

	Diploma Theses:
Lin Huang	*"A Node's Perspective of Changing Properties in Dynamic Networks"* [134] models how individual nodes move through the *core hierarchy* of the evolving AS graph and a graph of co-sold *dm* products and thereby change their properties.
Dieter Glaser	*Time-expanded* graph clustering is pioneered in *"Zeitexpandiertes Graphenclustern – Modellierung und Experimente"* [110], which evaluates models, measures and algorithms both theoretically and practically.
Florian Hübner	*"The Dynamic Graph Clustering Problem – ILP-Based Approaches Balancing Optimality and the Mental Map"* [136] develops many fundamental concepts about *smooth* dynamic clusterings based on ILPs, and evaluates them experimentally.
Stefanie Nagel	*"Optimisation of Clustering Algorithms for the Identification of Customer Profiles from Shopping Cart Data"* [170] is a comprehensive case study about how to find the most appropriate graph clustering algorithm for a real-world task.
Tanja Hartmann	*Minimum-cut tree clusterings* are dynamized in *"Clustering Dynamic Graphs with Guaranteed Quality"* [129], a strongly theoretical work with many results about dynamic cuts, which also proves its practical applicability.
Selma Mukhtar	*"Dynamische Clusteranalyse für DM-Verkaufsdaten"* [170] shows how much care is necessary to model bulky real-world data, but also reveals how graph clustering can find and follow customer profiles.
	Student Research Projects:
Abian Blome	*"Empirical Analysis of k-Betweenness"* [37] investigates what consequences a delayed recomputation of *betweenness* has on *betweenness*-based clustering algorithms.
Lin Huang	The name says it all for *"Survey on Generators for Internet Topologies at the AS Level"* [135], which measures many properties from network analysis.
Myriam Freidinger	*"Minimale Schnitte und Schnittbäume"* [95] investigates if and how *min-cuts* can be used to build *min-cut trees* and develops and evaluates heuristics.
Christian Schulz	*"Design and Experimental Evaluation of a Local Graph Clustering Algorithm"* [198] pursues high-speed graph clustering without index maximization.
Christian Staudt	As the name suggests, *"Algorithms and Experiments for Modularity-Driven Clusterings of Dynamic Graphs"* [203] aims at *modularity*-driven *online* dynamic clustering.
	Student Workers:
Hai Wei	His parsers, database tools, and visionary graph specification method are still in use.
Moritz Kroll	Without him, many errors might still lurk in the lecture notes for "Algorithmentechnik".
Pascal Maillard	Some of the results of this smart "theory-worker" still await application.
Jens Müller	Rewriting my ancient (and terrible) Java code was a valiant deed.
Florian Böhl	This "3D-worker" had our tool for algorithm visualization make a quantum leap.
Christian Schulz	Migrating from Java to C++ really sped clustering up a lot.
Christian Staudt	From the dynamic generator via dynamic clusterings to a powerful Matlab framework.
"The Markers"	Many thanks to Houssem Belloum, Tirdad Rahmani, Xuan Khanh Le, Thomas Pajor and Julian Dibbelt for their work on marking all that students' homework.

Section 5.3

Conclusion

...all good things must come to an end...

(Q to Jean-Luc Picard,
Star Trek: The Next Generation,
Episode 7x25/26, "All Good Things...")

The work conducted in this thesis advances the three areas that have been addressed: static graph clustering, network analysis and dynamic graph clustering. At the same time, many questions turned up, calling for more work. A common denominator of many such questions stems from the split view onto theoretical reasoning and practical behavior. However, before daring an outlook, the principal achievements of this thesis shall be summarized.

Modularity-driven clustering as it is done in practice has received corroboration by the NP-hardness of *modularity* optimization and by the good behavior of the greedy agglomerative heuristic in a systematic evaluation and a comparison to established clustering algorithms. Even the gap to a *modularity*-optimal clustering, computed with an integer linear program, was consistently small, in practice. The quality measure *modularity* itself has been shown to comply with human intuition of clustering goodness in large parts. Doubts concerning the usage of *coverage* as the base measure for *modularity* have been settled by the fact that replacing *coverage* by the more reliable measure *performance*, in the concept of *modularity*, yields an equivalent measure. At the same time, words of warning have been said about the measure *modularity* and the greedy algorithm. In non-simple graphs, this measure still works, but requires careful notation, and the probability space that supports it is not sound without loops and parallel edges. This result puts some previous works into question. For the quality of a clustering found by the the widespread greedy algorithm, no approximation factor can be given, in the worst case. The design and analysis of Orca, a fast clustering algorithm for huge graphs, revealed that without relying on any single index, simple structural operations can lead to clusterings with higher quality—even *modularity*—than *modularity*-driven algorithms. Together with its sole competitor, Orca is the only graph clustering algorithm that can tackle graphs approaching billions of elements. The first feasible measures for the comparison of graph clusterings have been proposed. Moreover it has been shown that traditional measures, which ignore the edge set of a graph, conflict with human intuition.

The need for visualizations of large clustered graphs fueled the development of *LunarVis*. This tool reveals abstract properties of a graph partition at a glance, however, more importantly, it allows for the simultaneous perception of structure inside clusters, element-level properties of nodes and edges, and connectivity between elements of the partition. An application in the field of network analysis showed that these analytic visualizations, or network *fingerprints*, are indeed suitable for guiding an analysis by revealing unknown traits. Discrepancies between the load distribution in a peer-to-peer network and a randomized simulation thereof could be exposed and further investigated. With instances growing in size, the importance of such *fingerprints* will increase, as they offer an easy overview of many properties of a network. In a veritable foray into network analysis, the relevance of the *core decomposition* to the above analysis then led to a random generator for graphs with a predefined *core* structure, which can additionally accommodate the concept of *preferential attachment*.

The upcoming field of dynamic graph clustering still lacks established problem statements and methods. The design of a random generator for dynamic graphs which feature an implanted *ground-truth* clustering that changes over time was a first and important step towards reliable and unbiased measurements of dynamic clustering algorithms. Returning to *modularity*-driven algorithms, dynamic versions of the most widespread greedy heuristic for *modularity* maximization and the currently fastest local variant have been proposed. These algorithms are designed for the basic and reasonable task of quickly updating a graph clustering with high *modularity*, after the graph changes. The outcomes of an experimental evaluation on both generated and real-world instances strongly support the dynamic approach: In comparison to re-clustering a changed graph from scratch, the dynamic algorithms are not only much faster, they also achieve *smoother* clustering dynamics, i.e., consecutive clusterings do not differ much, and consistently yield a higher quality than their static counterparts. Using the same search space as the dynamic heuristics, even a locally optimal integer linear program could not compete with the heuristics in terms of any of the above three criteria. Clustering algorithms which allow for a provable clustering guarantee are rare, even more so in dynamic scenarios. A fully dynamic version of a clustering algorithm based on *minimum-cut trees* has been presented, which dynamically maintains the bottleneck quality the static clustering algorithm guarantees. Apart from an asymptotic speed-up in most combinatorial cases and very *smooth* updates of clusterings in general, the work on this dynamic clustering algorithm yields many new insights into the structure of minimum cuts in changing graphs. For a whole family of problem statements in an *offline* setting, a framework for optimal graph clusterings of dynamic graphs has been given. For a typical *offline* scenario, *time-expanded* graph clustering has been proposed and evaluated in a case study. This method does not only yield good clusterings of each single snapshot of a dynamic graph, with *smooth* transitions between consecutive clusterings, but also provides correspondences between the clusters of different snapshots. In contrast to the few existing approaches for such a task, on the one hand, *time-expanded* graph clustering uses true *offline* knowledge about the instance when computing the clusterings, and on the other hand, this technique avoids the additional, error-prone step of matching clusters between *time steps*, in order to actually track clusters over time.

Summary. The common practice of using *modularity* for static graph clustering kicked off this thesis. *Modularity* can now be claimed to be largely understood. Graphs which previously were prohibitively large can be clustered very well, even without *modularity*, using Orca. Traditional graph drawing conventions become secondary criteria in the *fingerprints* of large, partitioned networks, made by *LunarVis*. Quick and *smooth* dynamic clustering algorithms are indispensable when analyzing or monitoring large and evolving networks. Using the first sound notions of clustering *smoothness*, methods are now at hand which reliably cluster changing graphs faster, *smoother* and better than static methods. One proposed method even allows quality guarantees, in case sparse bottlenecks between, and good connectivity within clusters must be ensured over time. Finally, trend analysis and evolutionary graph clustering can be conducted with the *time-expanded* approach, which solves *offline* clustering problems in dynamic graphs and allows to track clusters over time.

Outlook. All things considered, the practitioner has little knowhow on graph clustering, but needs a reliable tool, which is simple to use. In such a tool, graph theory and established methods from network analysis will make the choices a practitioner cannot make in a meaningful way. It can quickly analyse the network and decide which clustering technique to apply and how to set parameters. The result will then be presented to the user in an adequate format, revealing many other traits of a network besides the clustering itself. This thesis contains advances towards such a vision, however, graph theory, algorithm engineering and experimental analysis will have to continue hand in hand quite another little bit. The work in this thesis suggests that much of what has been learned in static graph clustering can be used to reliably cluster changing graphs. I conjecture that *offline* graph clustering harbors a potential for predictions about future clustering structure in evolving graphs.

Bibliography

[1] CPLEX. Mathematical programming optimizer, ILOG, http://www.ilog.com/products/cplex.

[2] lp-solve. A Mixed Integer Linear Programming (MILP) solver, free under the LGPL, http://lpsolve.sourceforge.net/.

[3] Plankton: Visualizing NLANR's Web Cache Hierarchy, 1998.

[4] DBLP - DataBase systems and Logic Programming, 2007. `http://dblp.uni-trier.de/`.

[5] CiteSeer, Scientific Literature Digital Library, 2008. `citeseer.ist.psu.edu`.

[6] Laboratory for Web Algorithmics, 2008. `law.dsi.unimi.it`.

[7] arXiv.org e-Print archive, 2009.

[8] Gaurav Agarwal and David Kempe. Modularity-Maximizing Graph Communities via Mathematical Programming . *The European Physical Journal B*, 66(3):409–418, December 2008.

[9] Charu C. Aggarwal and Philip S. Yu. Online Analysis of Community Evolution in Data Streams. In SDM'05 [199].

[10] Vinay Aggarwal, Stefan Bender, Anja Feldmann, and Arne Wichmann. Methodology for Estimating Network Distances of Gnutella Neighbors. In *GI Jahrestagung*, pages 219–223, 2004.

[11] Vinay Aggarwal, Anja Feldmann, Marco Gaertler, Robert Görke, and Dorothea Wagner. Analysis of Overlay-Underlay Topology Correlation using Visualization. Technical Report 2005-31, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2005.

[12] Vinay Aggarwal, Anja Feldmann, Marco Gaertler, Robert Görke, and Dorothea Wagner. Analysis of Overlay-Underlay Topology Correlation using Visualization. In *Proceedings of the 5th IADIS International Conference WWW/Internet Geometry*, 2006. Awarded as outstanding paper.

[13] Vinay Aggarwal, Anja Feldmann, Marco Gaertler, Robert Görke, and Dorothea Wagner. A Visualization-Driven Approach to Overlay-Underlay Engineering. In Friedhelm Meyer auf der Heide, editor, *Proceedings of the Final Workshop of DELIS*, HNI-Verlagsschriftenreihe, pages 81–97. Heinz Nixdorf Institut, Universität Paderborn, December 2007.

[14] Vinay Aggarwal, Anja Feldmann, Marco Gaertler, Robert Görke, and Dorothea Wagner. Modelling Overlay-Underlay Correlations Using Visualization. *Telektronikk*, 104(1):114–125, 2008.

[15] Vinay Aggarwal, Anja Feldmann, and Christian Scheideler. Can ISPs and P2P Systems Cooperate for Improved Performance? *ACM SIGCOMM Computer Communication Review*, 37(3):29–40, 2007.

[16] Réka Albert and Albert-László Barabási. Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*, 74(1):47–97, 2002.

[17] Charles J. Alpert and Andrew B. Kahng. Recent Directions in Netlist Partitioning: A Survey. *Integration: The VLSI Journal*, 19(1-2):1–81, 1995.

[18] José Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. Large Scale Networks Fingerprinting and Visualization Using the k-Core Decomposition. In *Advances in Neural Information Processing Systems 18*, pages 41–50. MIT Press, 2006.

[19] José Ignacio Alvarez-Hamelin, Marco Gaertler, Robert Görke, and Dorothea Wagner. Halfmoon - A new Paradigm for Complex Network Visualization. Technical Report 2005-29, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2005.

[20] Yuan An, Jeannette Janssen, and Evangelos E. Milios. Characterizing and Mining the Citation Graph of the Computer Science Literature. *Knowledge and Information Systems*, 6(6):664–678, 2004.

[21] Jacob M. Anthonisse. The rush in a directed graph. Technical Report BN 9/71, Stichting Mathematisch Centrum, 2e Boerhaavestraat 49 Amsterdam, Oct 1971.

[22] Alexander I. Archakov, Vadim M. Govorun, Alexander V. Dubanov, Yuri D. Ivanov, Alexander V. Veselovsky, Paul Lewi, and Paul Janssen. Protein-protein interactions as a target for drugs in proteomics. *Proteomics*, 3(4):380–391, April 2003.

[23] Alex Arenas, Jordi Duch, Alberto Fernandez, and Sergio Gomez. Size reduction of complex networks preserving modularity. *New Journal of Physics*, 9(176), 2007.

[24] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, and Alberto Marchetti-Spaccamela. *Complexity and Approximation - Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 2nd edition, 2002.

[25] James Bagrow. Evaluating local community methods in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008. Doi:10.1088/1742-5468/2008/05/P05001.

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

[27] Vladimir Batagelj and Ulrik Brandes. Efficient Generation of Large Random Networks. *Physical Review E*, (036113), 2005.

[28] Vladimir Batagelj and Matjaž Zaveršnik. An $\mathcal{O}(m)$ Algorithm for Cores Decomposition of Networks. Technical Report 798, IMFM Ljublana, Ljubljana, 2002.

[29] Vladimir Batagelj and Matjaž Zaveršnik. Generalized Cores. Preprint 799, IMFM Ljublana, Ljubljana, 2002.

[30] Michael Baur. *visone - Software for the Analysis and Visualization of Social Networks*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Informatik, November 2008.

[31] Michael Baur, Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Drawing the AS Graph in 2.5 Dimensions. In *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*, volume 3383 of *Lecture Notes in Computer Science*, pages 43–48. Springer, January 2005.

[32] Michael Baur, Marco Gaertler, and Robert Görke. Analyzing the Career of Actors - How to Become Famous Fast, 2005. Graph Drawing Contest at GD'05, Honorable Mention.

[33] Michael Baur, Marco Gaertler, Robert Görke, Marcus Krug, and Dorothea Wagner. Generating Graphs with Predefined k-Core Structure. In ECCS'07 [84]. Online proceedings `http://cssociety.org/ECCS07-Programme`.

[34] Michael Baur, Marco Gaertler, Robert Görke, Marcus Krug, and Dorothea Wagner. Augmenting k-Core Generation with Preferential Attachment. *Networks and Heterogeneous Media*, 3(2):277–294, June 2008.

[35] Michael Baur and Thomas Schank. Dynamic Graph Drawing in visone. Technical Report 2008-5, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2008.

[36] András A. Benzúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC'96)*, pages 47–55. ACM Press, 1996.

[37] Abian Blome. Empirical Analysis of k-Betweenness, October 2007. Studienarbeit Informatik.

[38] Vincent 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), 2008.

[39] Vincent Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Finding Communities in Large Networks, 2008. `http://findcommunities.googlepages.com/`.

[40] Christian Böhm, Christos Faloutsos, Jia-Yu Pan, and Claudia Plant. RIC: Parameter-Free Noise-Robust Clustering. *ACM Transactions on Knowledge Discovery from Data*, 1(3), December 2007.

[41] Ulrik Brandes. A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[42] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Höfer, Zoran Nikoloski, and Dorothea Wagner. Maximizing Modularity is hard, 2006. ArXiv physics/0608255.

[43] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Höfer, Zoran Nikoloski, and Dorothea Wagner. On Modularity - NP-Completeness and Beyond. Technical Report 2006-19, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2006.

[44] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Höfer, Zoran Nikoloski, and Dorothea Wagner. On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, February 2008.

[45] Ulrik Brandes, Daniel Delling, Martin Höfer, Marco Gaertler, Robert Görke, Zoran Nikoloski, and Dorothea Wagner. On Finding Graph Clusterings with Maximum Modularity. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'07)*, volume 4769 of *Lecture Notes in Computer Science*, pages 121–132. Springer, October 2007.

[46] Ulrik Brandes and Thomas Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *Lecture Notes in Computer Science*. Springer, February 2005.

[47] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Experiments on Graph Clustering Algorithms. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA'03)*, volume 2832 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2003.

[48] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Engineering Graph Clustering: Models and Experimental Evaluation. *ACM Journal of Experimental Algorithmics*, 12(1.1):1–26, 2007.

[49] Gerth Brodal and Riko Jacob. Dynamic Planar Convex Hull. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02)*, pages 617–626. IEEE Computer Society Press, 2002.

[50] Tian Bu and Don Towsley. On Distinguishing between Internet Power Law Topology Generators. In INFOCOM'02 [138].

[51] Thang Nguyen Bui, Frank Thomson Leighton, Soma Chaudhuri, and Michael Sipser. Graph Bisection Algorithms with Good Average Case Behavior. *Combinatorica*, 7(2):171–191, 1987.

[52] CAIDA: The Cooperative Association for Internet Data Analysis. `http://www.caida.org/`, 2008.

[53] Deepayan Chakrabarti, Ravi Kumar, and Andrew S. Tomkins. Evolutionary Clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 554–560. ACM Press, 2006.

[54] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with Qualitative Information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*. IEEE Computer Society Press, 2003.

[55] Qian Chen, Hyunseok Chang, Ramesh Govindan, and Sugih Jamin. The Origin of Power Laws in Internet Topologies Revisited. In INFOCOM'02 [138], pages 608–617.

[56] Fan R. K. Chung. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[57] Aaron Clauset, Mark E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(066111), 2004.

[58] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[59] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Jon Wiley & Sons, Inc., 1991.

[60] Gábor Csárdi and Tam'as Nepusz. The igraph Library for Complex Network Research, 2008. `http://igraph.sourceforge.net/index.html`.

[61] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The Complexity of Multiterminal Cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[62] Leon Danon, Albert Díaz-Guilera, and Alex Arenas. The effect of size heterogeneity on community identification in complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, (P11010), 2006.

[63] Leon Danon, Albert Díaz-Guilera, Jordi Duch, and Alex Arenas. Comparing Community Structure Identification. *Journal of Statistical Mechanics: Theory and Experiment*, 09(P09008):1–10, 2005.

[64] Ron Davidson and David Harel. Drawing Graphs Nicely using Simulated Annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996.

[65] Frank Dehne, Michael A. Langston, Xuemei Luo, Sylvain Pitre, Peter Shaw, and Yun Zhang. The Cluster Editing Problem: Implementations and Experiments. In Henning Fernau, Frank Dehne, Jianer Chen, Michael R. Fellows, and Yijia Chen, editors, *Parameterized and Exact Computation*, volume 4169/2006 of *Lecture Notes in Computer Science*, pages 13–24. Springer, September 2006.

[66] Frank Dehne, Jörg-Rüdiger Sack, and Roberto Tamassia, editors. *Algorithms and Data Structures, 11th International Workshop*, volume 5664 of *Lecture Notes in Computer Science*. Springer, August 2009.

[67] Daniel Delling. *Engineering and Augmenting Route Planning Algorithms*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Informatik, 2009. `http://i11www.ira.uka.de/extra/publications/d-earpa-09.pdf`.

[68] Daniel Delling, Marco Gaertler, Robert Görke, and Dorothea Wagner. Experiments on Comparing Graph Clusterings. Technical Report 2006-16, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2006.

[69] Daniel Delling, Marco Gaertler, Robert Görke, and Dorothea Wagner. Engineering Comparators for Graph Clusterings. In ECCS'07 [84]. As poster.

[70] Daniel Delling, Marco Gaertler, Robert Görke, and Dorothea Wagner. Engineering Comparators for Graph Clusterings. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM'08)*, volume 5034 of *Lecture Notes in Computer Science*, pages 131–142. Springer, June 2008.

[71] Daniel Delling, Marco Gaertler, and Dorothea Wagner. Generating Significant Graph Clusterings. In ECCS'06 [83]. Online Proceedings `http://cssociety.org/tiki-index.php?page=ECCS'06+Programme`.

[72] Daniel Delling, Robert Görke, Christian Schulz, and Dorothea Wagner. ORCA Reduction and ContrAction Graph Clustering. In Andrew V. Goldberg and Yunhong Zhou, editors, *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management (AAIM'09)*, volume 5564 of *Lecture Notes in Computer Science*, pages 152–165. Springer, June 2009.

[73] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Engineering Time-Expanded Graphs for Faster Timetable Information. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 182–206. Springer, 2009.

[74] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering Route Planning Algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.

[75] Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors. *9th DIMACS Implementation Challenge - Shortest Paths*, November 2006.

[76] Imre Derényi, Gergely Palla, and Tamás Vicsek. Clique Percolation in Random Networks. *Physical Review Letters*, 94, 2005.

[77] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing - Algorithms for the Visualization of Graphs*. Prentice Hall, 1998.

[78] Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2nd edition, 2000.

[79] Yefim Dinitz, Alexander V. Karzanov, and M. V. Lomonosov. On the structure of the system of minimum edge cuts in a graph. In A. A. Fridman, editor, *In Studies in Discrete Optimization*, pages 290–306. Nauka, 1976.

[80] Sergey N. Dorogovtsev, Andrew V. Goldberg, and Jose Ferreira F. Mendes. k-Core Organization of Complex Networks. *Physical Review Letters*, 96(040601):1–4, February 2006.

[81] Jordi Duch and Alex Arenas. Community Detection in Complex Networks using Extremal Optimization. *Physical Review E*, 72(027104):1–4, 2005.

[82] Nicolas Ducheneaut, Nicholas Yee, Eric Nickell, and Robert J. Moore. Alone Together?: Exploring the Social Dynamics of Massively Multiplayer Online Games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06)*, pages 407–416. ACM Press, 2006.

[83] *Proceedings of the European Conference of Complex Systems (ECCS'06)*, September 2006. Online Proceedings `http://cssociety.org/tiki-index.php?page=ECCS'06+Programme`.

[84] *Proceedings of the European Conference of Complex Systems (ECCS'07)*, October 2007. Online proceedings `http://cssociety.org/ECCS07-Programme`.

[85] Paul Erdős and Alfred Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[86] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the Internet topology. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 251–262. ACM Press, 1999.

[87] Gary William Flake, Robert E. Tarjan, and Kostas Tsioutsiouliklis. Graph Clustering and Minimum Cut Trees. *Internet Mathematics*, 1(4):385–408, 2004.

[88] Lester R. Ford, Jr. and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[89] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3–5):75–174, 2009.

[90] Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Science of the United States of America*, 104(1):36–41, 2007.

[91] Santo Fortunato, Andrea Lancichinetti, and Filippo Radicchi. New Benchmark in Community Detection. Arxiv.org/abs/0805.4770, 2008.

[92] Fabrizio Frati, Patrizio Angelini, and Michael Kaufmann. Straight-line Rectangular Drawings of Clustered Graphs. In Dehne et al. [66], pages 25–36.

[93] Linton Clarke Freeman. A Set of Measures of Centrality Based Upon Betweeness. *Sociometry*, 40:35–41, 1977.

[94] Linton Clarke Freeman. *The Development of Social Network Analysis: A Study in the Sociology of Science*. Booksurge Publishing, 2004.

[95] Myriam Freidinger. Minimale Schnitte und Schnittbäume, 2007. Studienarbeit Informatik.

[96] Arne K. Frick, Andreas Ludwig, and Heiko Mehldau. A Fast Adaptive Layout Algorithm for Undirected Graphs. In *DIAMCS International Workshop*, volume 894 of *Lecture Notes in Computer Science*, pages 388–403. Springer, January 1995.

[97] Thomas M. J. Fruchterman and Edward M. Reingold. Graph Drawing by Force-Directed Placement. *Software - Practice and Experience*, 21(11):1129–1164, November 1991.

[98] Marco Gaertler. Clustering with Spectral Methods. Diplomarbeit, Fachbereich Informatik und Informationswissenschaft, Universität Konstanz, March 2002.

[99] Marco Gaertler. Clustering. In Brandes and Erlebach [46], pages 178–215.

[100] Marco Gaertler. *Algorithmic Aspects of Clustering – Theory, Experimental Evaluation, and Applications in Network Analysis and Visualization*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Informatik, 2007.

[101] Marco Gaertler, Robert Görke, and Dorothea Wagner. Significance-Driven Graph Clustering. In *Proceedings of the 3rd International Conference on Algorithmic Aspects in Information and Management (AAIM'07)*, Lecture Notes in Computer Science, pages 11–26. Springer, June 2007.

[102] Marco Gaertler, Robert Görke, and Dorothea Wagner. Fingerprints - Means For Visual Analytics. In Hong et al. [131]. As poster, see `http://i11www.iti.uni-karlsruhe.de/algobib/files/ggw-fmfva-08.pdf`.

[103] Marco Gaertler, Robert Görke, Dorothea Wagner, and Silke Wagner. How to Cluster Evolving Graphs. In ECCS'06 [83]. Online available at `http://complexsystems.lri.fr/FinalReview/FILES/PDF/p103.pdf`.

[104] Marco Gaertler and Maurizio Patrignani. Dynamic Analysis of the Autonomous System Graph. In *IPS 2004 – Inter-Domain Performance and Simulation*, pages 13–24, March 2004.

[105] Giorgio Gallo, Michail D. Grigoriadis, and Robert E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989.

[106] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, 1979.

[107] Horst Gilbert. Random Graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

[108] Ioannis Giotis and Venkatesan Guruswami. Correlation Clustering with a Fixed Number of Clusters. In *Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 1167–1176, New York, NY, USA, 2006.

[109] Christos Gkantsidis, Milena Mihail, and Ellen W. Zegura. Spectral Analysis of Internet Topologies. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom)*, volume 1, pages 364–374. IEEE Computer Society Press, March 2003.

[110] Dieter Glaser. Zeitexpandiertes Graphenclustern - Modellierung und Experimente. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, February 2008.

[111] Joseph Glaz, Joseph Naus, and Sylvan Wallenstein. *Scan Statistics*. Springer Texts in Statistics. Springer, 2001.

[112] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

[113] Ralph E. Gomory and T.C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, December 1961.

[114] Robert Görke. Ein Schneller Konstruktionsalgorithmus für eine Quickest-Path-Map bezüglich der City-Metrik. Master's thesis, Universität Karlsruhe (TH), October 2004.

[115] Robert Görke, Marco Gaertler, Florian Hübner, and Dorothea Wagner. Computational Aspects of Lucidity-Driven Graph Clustering. *Journal of Graph Algorithms and Applications*, 14(2), 2010.

[116] Robert Görke, Marco Gaertler, and Dorothea Wagner. LunarVis - Analytic Visualizations of Large Graphs. In Hong et al. [131], pages 352–364.

[117] Robert Görke, Tanja Hartmann, and Dorothea Wagner. Dynamic Graph Clustering Using Minimum-Cut Trees. Technical report, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2009. Informatik, Uni Karlsruhe, TR 2009-10.

[118] Robert Görke, Tanja Hartmann, and Dorothea Wagner. Dynamic Graph Clustering Using Minimum-Cut Trees. In Dehne et al. [66].

[119] Robert Görke, Steffen Mecke, and Florian Böhl. Flow Commander, a Visualisation Tool for the Push Relabel Algorithm, 2006. Graph Drawing Competition at GD'06, Honorable Mention.

[120] Robert Görke and Christian Staudt. A Generator for Dynamic Clustered Random Graphs. Technical report, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2009. Informatik, Uni Karlsruhe, TR 2009-7.

[121] Robert Görke and Alexander Wolff. Constructing the City Voronoi diagram faster. In *Proceedings of the 21st European Workshop on Computational Geometry (EWCG'05)*, pages 155–158, March 2005.

[122] Robert Görke and Alexander Wolff. Constructing the City Voronoi diagram faster. In *Proceedings of the 2nd International Symposium on Voronoi Diagrams in Science and Engineering (VD'05)*, pages 162–172, October 2005. Best Presentation Award.

[123] Robert Görke, Alexander Wolff, and Chan-Su Shin. Constructing the City Voronoi diagram faster. *International Journal of Computational Geometry and Applications*, 18(4):275–294, August 2008.

[124] Hariolf Grupp and Ulrich Schmoch. Patent statistics in the age of globalisation: new legal procedures, new analytical methods, new economic interpretation. *Research Policy*, 28(4):377–396, April 1999.

[125] Roger Guimerà and Luís A. Nunes Amaral. Functional Cartography of Complex Metabolic Networks. *Nature*, 433:895–900, February 2005.

[126] Roger Guimerà, Marta Sales-Pardo, and Luís A. Nunes Amaral. Modularity from Fluctuations in Random Graphs and Complex Networks. *Physical Review E*, 70(025101), 2004.

[127] Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.

[128] Ronald J. Gutman. Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04)*, pages 100–111. SIAM, 2004.

[129] Tanja Hartmann. Clustering Dynamic Graphs with Guaranteed Quality. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, October 2008. Diplomarbeit Informatik.

[130] Erez Hartuv and Ron Shamir. A Clustering Algorithm based on Graph Connectivity. *Information Processing Letters*, 76(4-6):175–181, 2000.

[131] Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors. *Proceedings of the 15th International Symposium on Graph Drawing (GD'07)*, volume 4875 of *Lecture Notes in Computer Science*. Springer, January 2008.

[132] John E. Hopcroft, Omar Khan, Brian Kulis, and Bart Selman. Tracking Evolving Communities in Large Linked Networks. *Proceedings of the National Academy of Science of the United States of America*, 101, April 2004.

[133] Martin Höpner and Lothar Krempel. The Politics of the German Company Network. *Competition and Change*, 8(4):339–356, December 2004.

[134] Lin Huang. A Nodes Perspective of Changing Properties in Dynamic Graphs. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, December 2007. Diplomarbeit Informatik.

[135] Lin Huang. Survey on Generators for Internet Topologies at the AS Level, January 2007. Studienarbeit Informatik.

[136] Florian Hübner. The Dynamic Graph Clustering Problem - ILP-Based Approaches Balancing Optimality and the Mental Map. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, May 2008. Diplomarbeit Informatik.

[137] Bradley Huffaker, Jaeyeon Jung, Evi Nemeth, Duane Wessels, and K. Claffy. Visualization of the growth and topology of the NLANR caching hierarchy. *Computer Networks and ISDN Systems*, 30(22-23):2131–2139, November 1998.

[138] *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom)*, volume 1. IEEE Computer Society Press, 2002.

[139] Riko Jacob. *Dynamic Planar Convex Hull*. PhD thesis, BRICS Research Centre, 2002.

[140] Anil K. Jain, M. N. Murty, and Patrick J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, September 1999.

[141] Hawoong Jeong, Sean P. Mason, Albert-László Barabási, and Zoltan N. Oltvai. Lethality and Centrality in Protein Networks. *Nature*, 411, 2001. Brief communications.

[142] Cheng Jin, Qian Chen, and Sugih Jamin. Inet Topology Generator. Technical Report CSE-TR-433, EECS Department, University of Michigan, 2000.

[143] Dieter Jungnickel. *Graphs, Networks and Algorithms*, volume 5 of *Algorithms and Computation in Mathmatics*. Springer, 1999.

[144] Volker Kaibel, Matthias Peinhardt, and Marc Pfetsch. Orbitopal Fixing. In Matteo Fischetti and David Williamson, editors, *Proceedings of the 12th Integer Programming and Combinatorial Optimization Conference*, Lecture Notes in Computer Science, pages 74–88. Springer, 2007.

[145] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On Clusterings - Good, Bad and Spectral. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS'00)*, pages 367–378, 2000.

[146] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On Clusterings: Good, Bad, Spectral. *Journal of the ACM*, 51(3):497–515, May 2004.

[147] Daniel A. Keim, Jörn Schneidewind, and Mike Sips. Scalable Pixel Based Visual Data Exploration. In *Pixelization Paradigm, Revised Selected Papers of the First Visual Information Expert Workshop*, volume 1 of *Lecture Notes in Computer Science*, pages 12–24. Springer, April 2006.

[148] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards Parameter-Free Data Mining. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 206–215. ACM Press, 2004.

[149] Donald E. Knuth. Two notes on notation. *American Mathematical Monthly*, 99:403–422, 1990.

[150] Pushmeet Kohli and Philip H. Torr. Dynamic Graph Cuts for Efficient Inference in Markov Random Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2079–2088, December 2007.

[151] Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew S. Tomkins. On the Bursty Evolution of Blogspace. In *Proceedings of the 12th International World Wide Web Conference (WWW12)*, pages 568–576, Budapest, Hungary, 2003.

[152] Wei Lai, Peter Eades, K. Misue, and Kozo Sugiyama. Preserving the Mental Map of a Diagram. In *Proceedings of Compugraphics '91*, pages 24–33, 1991.

[153] Andrea Lancichinetti, Santo Fortunato, and János Kertéz. Detecting the Overlapping and Hierarchical Community Structure of Complex Networks. Http://arxiv.org/abs/0802.1218, 2008.

[154] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs Over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 177–187. ACM Press, 2005.

[155] Chuan Lin, Young rae Cho, Woo chang Hwang, Pengjun Pei, and Aidong Zhang. Clustering Methods in a Protein-Protein Interaction Network. In Xiaohua Hu and Yi Pan, editors, *Knowledge Discovery in Bioinformatics*, pages 319–355. Jon Wiley & Sons, Inc., May 2007.

[156] Yong Liu, Honggang Zhang, Weibo Gongi, and Don Towslef. On the Interaction between Overlay Routing and Traffic Engineering. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom)*, volume 4, pages 2543–2553. IEEE Computer Society Press, March 2005.

[157] David Lusseau, Karsten Schneider, Oliver Boisseau, Patti Haase, Elisabeth Slooten, and Steve Dawson. The Bottlenose Dolphin Community of Doubtful Sound Features a Large Proportion of Long-Lasting Associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, September 2004.

[158] Damien Magoni. nem: A Software for Network Topology Analysis and Modeling. In *Proceedings of the 10th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE Computer Society, 2002.

[159] Damien Magoni and Jean Jacques Pansiot. Analysis and Comparison of Internet Topology Generators. In *Proceedings of the 2nd International IFIP-TC6 Networking Conference*, pages 364–375. Springer, 2002.

[160] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In *Proceedings of the 9th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2001.

[161] Alberto Medina, Ibrahim Matta, and John Byers. On the Origin of Power Laws in Internet Topologies. *Computer Communication Review*, 30(2), April 2000.

[162] Marina Meilă. Comparing Clusterings by the Variation of Information. In Bernhard Schölkopf and Manfred K. Warmuth, editors, *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, Proceedings*, Lecture Notes in Computer Science, pages 173–187. Springer, August 2003.

[163] Marina Meilă. Comparing Clusterings - An Axiomatic View. In *Proceedings of the 22nd International Conference on Machine Learning*. ACM Press, 2005.

[164] David Meyer. University of Oregon Route Views Project, 2007. Web page; [Online at `http://routeviews.org/`; accessed 31-August-2007].

[165] Henning Meyerhenke, Burkhard Monien, and Stefan Schamberger. Accelerating Shape Optimizing Load Balancing for Parallel FEM Simulations by Algebraic Multigrid. In *20th International Parallel and Distributed Processing Symopsium (IPDPS'06)*, page 10. IEEE Computer Society, 2006.

[166] Burkhard Monien and Stefan Schamberger. Graph Partitioning with the Party Library: Helpful-Sets in Practice. In *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04)*, pages 198–205. IEEE Computer Society, 2004.

[167] Leslie C. Morey and Alan Agresti. The Measurement of Classification Agreement: An Adjustment to the RAND Statistic for Chance Agreement. *Educational and Psychological Measurement*, 44:33–37, 1984.

[168] Stefanie Muff, Francesco Rao, and Amedeo Caflisch. Local Modularity Measure for Network Clusterizations. *Physical Review E*, 72(056107):1–4, 2005.

[169] Selma Mukhtar. Dynamische Clusteranalyse für DM-Verkaufsdaten. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, April 2009. Diplomarbeit Informatik.

[170] Stefanie Nagel. Optimisation of Clustering Algorithms for the Identification of Customer Profiles from Shopping Cart Data. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, October 2008. Diplomarbeit Informatik.

[171] Akihiro Nakao, Larry Peterson, and Andy Bavier. A Routing Underlay for Overlay Networks. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 11–18. ACM Press, 2003.

[172] Mark E. J. Newman. Analysis of Weighted Networks. *Physical Review E*, 70(056131):1–9, 2004.

[173] Mark E. J. Newman. Fast Algorithm for Detecting Community Structure in Networks. *Physical Review E*, 69:066133, 2004.

[174] Mark E. J. Newman. A Measure of Betweenness Centrality Based on Random Walks. *Social Networks*, 27(1):39–54, January 2005.

[175] Mark E. J. Newman. Modularity and Community Structure in Networks. *Proceedings of the National Academy of Science of the United States of America*, 103(23):8577–8582, June 2006.

[176] Mark E. J. Newman, Albert-László Barabási, and Duncan J. Watts. *The Structure and Dynamics of Networks*. Princeton Studies in Complexity. Princeton University Press, 2006.

[177] Mark E. J. Newman and Michelle Girvan. Mixing Patterns and Community Structure in Networks. In Romualdo Pastor-Satorras, Miguel Rubi, and Albert Díaz-Guilera, editors, *Statistical Mechanics of Complex Networks*, volume 625 of *Lecture Notes in Physics*, pages 66–87. Springer, 2003.

[178] Mark E. J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(026113), 2004.

[179] Vincenzo Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri. Extending Modularity Definition for Directed Graphs with Overlapping Communities. January 2008.

[180] Vincenzo Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri. Extending the Definition of Modularity to Directed Graphs with Overlapping Communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(03):p03024 (23pp), 2009.

[181] Andreas Noack and Randolf Rotta. Multi-level Algorithms for Modularity Clustering. In Jan Vahrenhold, editor, *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09)*, volume 5526 of *Lecture Notes in Computer Science*, pages 257–268. Springer, June 2009.

[182] Gergely Palla, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. *Nature*, 446:664–667, April 2007.

[183] Romualdo Pastor-Satorras and Alessandro Vespignani. *Evolution and Structure of the Internet: A Statistical Physics Approach*. Cambridge University Press, 2004.

[184] Marc Pfetsch and Volker Kaibel. Packing and Partitioning Orbitopes. *Mathematical Programming, Series A*, 114(1):1–36, 2008.

[185] Andrew Philippides, Peter Fine, and Ezequiel Di Paolo. Spatially Constrained Networks and the Evolution of Modular Control Systems. In *Proc. 9th International Conference on Simulation of Adaptive Behavior*, Lecture Notes in Computer Science, pages 546–557. Springer, 2006.

[186] Jean-Claude Picard and Maurice Queyranne. On the Structure of All Minimum Cuts in a Network and Applications. *Mathematical Programming, Series A*, 22(1):121, December 1982.

[187] Pascal Pons and Matthieu Latapy. Computing Communities in Large Networks Using Random Walks. *Journal of Graph Algorithms and Applications*, 10(2):191–218, 2006.

[188] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12:Article 2.4, 2007.

[189] William M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, 66(336):846–850, December 1971.

[190] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott J. Shenker. Topologically-Aware Overlay Construction and Server Selection. In INFOCOM'02 [138], pages 1190–1199.

[191] Jörg Reichardt and Stefan Bornholdt. Statistical Mechanics of Community Detection. *Physical Review E*, 74(016110):1–16, 2006.

[192] Barna Saha and Pabitra Mitra. Dynamic Algorithm for Graph Clustering Using Minimum Cut Tree. In *Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, pages 667–671. IEEE Computer Society, December 2006.

[193] Barna Saha and Pabitra Mitra. Dynamic Algorithm for Graph Clustering Using Minimum Cut Tree. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 581–586. SIAM, 2007.

[194] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN) 2002*, January 2002.

[195] Satu Elisa Schaeffer. Graph Clustering. *Computer Science Review*, 1(1):27–64, August 2007.

[196] Satu Elisa Schaeffer, Stefano Marinoni, Mikko Särelä, and Pekka Nikander. Dynamic Local Clustering for Hierarchical Ad Hoc Networks. In *Proceedings of Sensor and Ad Hoc Communications and Networks, 2006.*, volume 2, pages 667–672. IEEE Computer Society, September 2006.

[197] Thomas Schank and Dorothea Wagner. Approximating Clustering Coefficient and Transitivity. *Journal of Graph Algorithms and Applications*, 9(2):265–275, 2005.

[198] Christian Schulz. Design and Experimental Evaluation of a Local Graph Clustering Algorithm, June 2008. Informatik Studienarbeit.

[199] *Proceedings of the fifth SIAM International Conference on Data Mining*. SIAM, 2005.

[200] Srinivasan Seetharaman and Mostafa Ammar. On the Interaction Between Dynamic Routing in Native and Overlay Layers. In *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom)*, pages 1–12. IEEE Computer Society Press, April 2006.

[201] Stephen B. Seidman. Network Structure and Minimum Degree. *Social Networks*, 5:269–287, 1983.

[202] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster Graph Modification Problems. In *Proceedings of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'02)*, volume 2573 of *Lecture Notes in Computer Science*, pages 379–390. Springer, 2002.

[203] Christian Staudt. Algorithms and Experiments for Modularity-Driven Clusterings of Dynamic Graphs, 2009. Studienarbeit Informatik, to appear.

[204] Ralf Steinmetz and Klaus Wehrle, editors. *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*. Springer, 2005.

[205] Mechthild Stoer and Frank Wagner. A Simple Min-Cut Algorithm. *Journal of the ACM*, 44(4):585–591, July 1997.

[206] Alexander Strehl and Joydeep Ghosh. Cluster Ensembles - a Knowledge Reuse Framework For Combining Multiple Partitions. *Journal of Machine Learning*, 3:583–617, 2003.

[207] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the Internet Hierarchy from Multiple Vantage Points. In INFOCOM'02 [138], pages 618–627.

[208] Jimeng Sun, Philip S. Yu, Spiros Papadimitriou, and Christos Faloutsos. GraphScope: Parameter-Free Mining of Large Time-Evolving Graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 687–696. ACM Press, 2007.

[209] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining.* Addison-Wesley, 2006.

[210] John Ronald Reuel Tolkien. *The Lord of the Rings.* George Allen & Unwin, 1954. In three volumes.

[211] Masashi Toyoda and Masaru Kitsuregawa. Extracting Evolution of Web Communities from a Series of Web Archives. In *Proceedings of the 14th ACM Conference on Hypertext and Hypermedia*, pages 28–37. ACM Press, 2003.

[212] University of Oregon Routeviews Project. `http://www.routeviews.org/`, 2008.

[213] Stijn M. van Dongen. *Graph Clustering by Flow Simulation.* PhD thesis, University of Utrecht, 2000.

[214] Berthold Vöcking, Helmut Alt, Martin Dietzfelbinger, Rüdiger Reischuk, Christian Scheideler, Heribert Vollmer, and Dorothea Wagner, editors. *Taschenbuch der Algorithmen.* Springer, 2008.

[215] Ulrike von Luxburg. A Tutorial on Spectral Clustering. *Statistics and Computing*, 17(4):395–416, December 2007.

[216] Dorothea Wagner and Frank Wagner. Between Min Cut and Graph Bisection. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *MFCS '93: Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science*, volume 711 of *Lecture Notes in Computer Science*, pages 744–750, London, UK, 1993. Springer.

[217] Dorothea Wagner and Thomas Willhalm. Speed-Up Techniques for Shortest-Path Computations. In *Proceedings of the 24th International Symposium on Theoretical Aspects of Computer Science (STACS'07)*, volume 4393 of *Lecture Notes in Computer Science*, pages 23–36. Springer, 2007.

[218] Silke Wagner and Dorothea Wagner. Comparing Clusterings – An Overview. Technical Report 2006-04, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2006.

[219] Ken Wakita and Toshiyuki Tsurumi. Finding Community Structure in Mega-scale Social Networks, February 2007. Technical Report on arXiv.

[220] Bei Wang, Jeff M. Phillips, Robert Schreiber, Dennis Wilkinson, Nina Mishra, and Robert E. Tarjan. Spatial Scan Statistics for Graph Clustering. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 727–738. SIAM, 2008.

[221] Colin Ware. *Information Visualization, Second Edition: Perception for Design.* Morgan Kaufmann, 2007.

[222] Duncan J. Watts and Steven H. Strogatz. Collective Dynamics of "Small-World" Networks. *Nature*, 393:440–442, 1998.

[223] Fang Wei, Chen Wang, Li Ma, and Aoying Zhou. Detecting Overlapping Community Structures in Networks with Global Partition and Local Expansion. In *Proceedings of the 10th Asia-Pacific Web Conference*, Lecture Notes in Computer Science, pages 43–55. Springer, April 2008.

[224] Markus R. Wenk. The Emerging Field of Lipidomics. *Nature Reviews Drug Discovery*, 4:594–610, July 2005.

[225] Craig E. Wheelock, Susumu Goto, Laxman Yetukuri, Fabio Luiz D'Alexandri, Christian Klukas, Falk Schreiber, and Matej Oresic. Bioinformatics Strategies for the Analysis of Lipids. In Toshihiko Osawa and Fabio Luiz D'Alexandri, editors, *Lipidomics Volume 2: Methods and Protocols* , volume 2009 of *Methods in Molecular Biology*, pages 339–368. Springer, 2009.

[226] Scott White and Padhraic Smyth. A Spectral Clustering Approach to Finding Communities in Graphs. In SDM'05 [199], pages 274–285.

[227] Wikipedia. Gnutella — Wikipedia, the free encyclopedia, 2007. [Online at `http://en.wikipedia.org/wiki/Gnutella`; accessed 31-August-2007].

[228] Stefan Wuchty and Eivind Almaas. Peeling the Yeast Protein Network. *Proteomics*, 5(2):444–449, 2005.

[229] G. Xu, Sophia Tsoka, and Lazaros G. Papageorgiou. Finding Community Structures in Complex Networks using Mixed Integer Optimisation . *The European Physical Journal B*, 60(2):231–239, November 2007.

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

[231] Etay Ziv, Manuel Middendorf, and Chris H. Wiggins. Information-Theoretic Approach to Network Modularity. *Physical Review E*, 71(046117):1–9, 2005.

List of Figures

List of Tables

List of Algorithms

Index

List of Publications

Book Chapter

[1] **Maximale Flüsse - Die ganze Stadt will zum Stadion**. In: *Taschenbuch der Algorithmen*, pages 361–372. Springer, 2008. Joint work with Steffen Mecke and Dorothea Wagner.

Journal Articles

[2] **Augmenting k-Core Generation with Preferential Attachment**. *Networks and Heterogeneous Media*, 3(2):277–294, June 2008. Joint work with Michael Baur, Marco Gaertler, Marcus Krug, and Dorothea Wagner.

[3] **Constructing the City Voronoi diagram faster**. *International Journal of Computational Geometry and Applications*, 18(4):275–294, August 2008. Joint work with Alexander Wolff and Chan-Su Shin.

[4] **Modelling Overlay-Underlay Correlations Using Visualization**. *Telektronikk*, 104(1):114–125, 2008. Joint work with Vinay Aggarwal, Anja Feldmann, Marco Gaertler, and Dorothea Wagner.

[5] **On Modularity Clustering**. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, February 2008. Joint work with Ulrik Brandes, Daniel Delling, Marco Gaertler, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner.

[6] **Computational Aspects of Lucidity-Driven Graph Clustering**. *Journal of Graph Algorithms and Applications*, 14(2)165–197, January 2010. Joint work with Marco Gaertler, Florian Hübner, and Dorothea Wagner.

Articles in Refereed Conference Proceedings

[7] **Analysis of Overlay-Underlay Topology Correlation using Visualization**. In: *Proceedings of the 5th IADIS International Conference WWW/Internet Geometry*, 2006. Awarded as outstanding paper, joint work with Vinay Aggarwal, Anja Feldmann, Marco Gaertler, and Dorothea Wagner.

[8] **Constructing the City Voronoi diagram faster**. In: *Proceedings of the 2nd International Symposium on Voronoi Diagrams in Science and Engineering (VD'05)*, pages 162–172, October 2005. Best Presentation Award, joint work with Alexander Wolff.

[9] **Dynamic Graph Clustering Using Minimum-Cut Trees**. In: *Algorithms and Data Structures, 11th International Workshop*, volume 5664 of *Lecture Notes in Computer Science*. Springer, August 2009. Joint work with Tanja Hartmann and Dorothea Wagner.

[10] **Engineering Comparators for Graph Clusterings**. In: *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM'08)*, volume 5034 of *Lecture Notes in Computer Science*, pages 131–142. Springer, June 2008. Joint work with Daniel Delling, Marco Gaertler, and Dorothea Wagner.

[11] **Generating Graphs with Predefined k-Core Structure**. In: *Proceedings of the European Conference of Complex Systems (ECCS'07)*, October 2007. Joint work with Michael Baur, Marco Gaertler, Marcus Krug, and Dorothea Wagner.

[12] **How to Cluster Evolving Graphs**. In: *Proceedings of the European Conference of Complex Systems (ECCS'06)*, September 2006. Joint work with Marco Gaertler, Dorothea Wagner, and Silke Wagner.

[13] **LunarVis - Analytic Visualizations of Large Graphs**. In: *Proceedings of the 15th International Symposium on Graph Drawing (GD'07)*, volume 4875 of *Lecture Notes in Computer Science*, pages 352–364. Springer, January 2008. Joint work with Marco Gaertler and Dorothea Wagner.

[14] **On Finding Graph Clusterings with Maximum Modularity**. In: *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'07)*, volume 4769 of *Lecture Notes in Computer Science*, pages 121–132. Springer, October 2007. Joint work with Ulrik Brandes, Daniel Delling, Martin Hoefer, Marco Gaertler, Zoran Nikoloski, and Dorothea Wagner.

[15] **A Visualization-Driven Approach to Overlay-Underlay Engineering**. In: Friedhelm Meyer auf der Heide, editor, *Proceedings of the Final Workshop of DELIS*, HNI-Verlagsschriftenreihe, pages 81–97. Heinz Nixdorf Institut, Universität Paderborn, December 2007. Joint work with Vinay Aggarwal, Anja Feldmann, Marco Gaertler and Dorothea Wagner.

[16] **ORCA Reduction and ContrAction Graph Clustering**. In: *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management (AAIM'09)*, volume 5564 of *Lecture Notes in Computer Science*, pages 152–165. Springer, June 2009. Joint work with Daniel Delling, Christian Schulz, and Dorothea Wagner.

[17] **Significance-Driven Graph Clustering**. In: *Proceedings of the 3rd International Conference on Algorithmic Aspects in Information and Management (AAIM'07)*, Lecture Notes in Computer Science, pages 11–26. Springer, June 2007. Joint work with Marco Gaertler and Dorothea Wagner.

Poster and Reports in Refereed Workshop Proceedings

[18] **Analyzing the Career of Actors - How to Become Famous Fast**, 2005. Graph Drawing Contest at GD'05, Honorable Mention, joint work with Michael Baur and Marco Gaertler.

[19] **Flow Commander, a Visualisation Tool for the Push Relabel Algorithm**, 2006. Graph Drawing Competition at GD'06, Honorable Mention, joint work with Steffen Mecke and Florian Böhl.

[20] **Static and Dynamic Visual Analysis of a Co-Author Network**, 2007. Graph Drawing Contest at GD'07, 2nd Prize, joint work with Thomas Schank and Dorothea Wagner.

[21] **Engineering Comparators for Graph Clusterings**. In: *Proceedings of the European Conference of Complex Systems (ECCS'07)*, October 2007, as poster. Joint work with Daniel Delling, Marco Gaertler, and Dorothea Wagner.

[22] **Evaluating Clustering Techniques - An Engineering Approach Inspired by Unit-Tests**. In: *Proceedings of the European Conference of Complex Systems (ECCS'07)*, October 2007, as poster. Joint work with Daniel Delling, Marco Gaertler, Zoran Nikoloski, and Dorothea Wagner.

[23] **Fingerprints - Means For Visual Analytics**. In: *Proceedings of the 15th International Symposium on Graph Drawing (GD'07)*, volume 4875 of *Lecture Notes in Computer Science*. Springer, January 2008, as poster. Joint work with Marco Gaertler and Dorothea Wagner.

Articles in Non-Refereed Workshop Proceedings

[24] **Constructing the City Voronoi diagram faster**. In: *Proceedings of the 21st European Workshop on Computational Geometry (EWCG'05)*, pages 155–158, March 2005. Joint work with Alexander Wolff.

Technical Reports and Articles on arXiv.org

[25] **Maximizing Modularity is hard**, 2006. arXiv physics/0608255, joint work with Ulrik Brandes, Daniel Delling, Marco Gaertler, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner.

[26] **A Generator for Dynamic Clustered Random Graphs**. Technical report, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2009. Informatik, Uni Karlsruhe, TR 2009-7, joint work with Christian Staudt.

[27] **Analysis of Overlay-Underlay Topology Correlation using Visualization**. Technical Report 2005-31, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2005. Joint work with Vinay Aggarwal, Anja Feldmann, Marco Gaertler, and Dorothea Wagner.

[28] **Dynamic Graph Clustering Using Minimum-Cut Trees**. Technical report, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2009. Informatik, Uni Karlsruhe, TR 2009-10, joint work with Tanja Hartmann and Dorothea Wagner.

[29] **Experiments on Comparing Graph Clusterings**. Technical Report 2006-16, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2006. Joint work with Daniel Delling, Marco Gaertler, and Dorothea Wagner.

[30] **Halfmoon - A new Paradigm for Complex Network Visualization**. Technical Report 2005-29, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2005. Joint work with José Ignacio Alvarez-Hamelin, Marco Gaertler, and Dorothea Wagner.

[31] **How to Evaluate Clustering Techniques**. Technical Report 2006-24, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2006. Joint work with Daniel Delling, Marco Gaertler, Zoran Nikoloski, and Dorothea Wagner.

[32] **On Modularity - NP-Completeness and Beyond**. Technical Report 2006-19, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2006. Joint work with Ulrik Brandes, Daniel Delling, Marco Gaertler, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner.

Thesis

[33] **Ein Schneller Konstruktionsalgorithmus für eine Quickest-Path-Map bezüglich der City-Metrik**. Diplomarbeit Mathematik, Universität Karlsruhe (TH), October 2004.

Curriculum Vitæ

Name	Robert Görke
Date of Birth	9^{th} of February 1978
Place of Birth	Ruit, Ostfildern, Germany
Nationality	German

06/1997	**Abitur** (university entrance qualification), Raichberg-Gymnasium-Ebersbach
08/1997-08/1998	**Alternative civilian service** as stand-by man in the operating theater of Krankenhaus Radolfzell
09/1998	**Enrollment** as a student in Technomathematics at Universität Karlsruhe (TH)
02/2002-12/2002	**Visiting student** at The University of Auckland, Auckland, New Zealand, supported by a scholarship of the German Academic Exchange Service (DAAD)
11/2002	**Postgraduate Diploma of Science** in Applied Mathematics, The University of Auckland
12/2004	**Diploma** in Technomathematics, Universität Karlsruhe (TH)
01/2005-03/2008	**PhD student** and research assistant in the FET-project "DELIS – Dynamically Evolving, Large-scale Information Systems" funded by the EU, Fakultät für Informatik, Universität Karlsruhe (TH). Advisor: Prof. Dr. Dorothea Wagner
since 11/2007	**PhD student** and research assistant in the Priority Programme 1307 "Algorithm Engineering", project "Clustering of Static and Temporal Graphs" funded by the DFG, Fakultät für Informatik, Universität Karlsruhe (TH). Advisor: Prof. Dr. Dorothea Wagner