Engineering Graph Clustering Algorithms

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

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

genehmigte

Dissertation

von

Andrea Kappes

aus Schwäbisch Hall

Tag der mündlichen Prüfung:	28. April 2015
Erster Gutachter:	Prof. Dr. Dorothea Wagner
Zweiter Gutachter:	Prof. Dr. Ulrik Brandes

DOI: 10.5445/IR/1000049269

Acknowledgments

First of all, I would like to thank my advisor Dorothea Wagner for giving me the possibility to work in her group. Among many things, I am especially grateful for her patience, encouragement, and support during the last year, making it possible for me to finish my dissertation after my son was born.

My work as a PhD student would have been much more cumbersome and boring without all of my former and current colleagues. In particular, I would like to thank all coauthors of my publications for the good discussions and fruitful cooperation. Special thanks go to Robert Görke for introducing me to the concepts behind graph clustering and teaching me how to write scientific texts. A big thank-you goes to Tanja Hartmann, Andreas Gemsa, and Markus Völker for proof-reading parts of this thesis and giving valuable feedback. Furthermore, I would like to thank Lilian Beckert, Elke Sauer, Bernd Giesinger and Rolf Kölmel for their friendly help with administrative tasks and technical issues, Ulrik Brandes for willingly taking over the task to review this thesis, and David Bader for giving me the opportunity to visit his group at Georgia Tech.

Finally, I would like to thank everyone that supported and encouraged me outside the office, especially my parents and my husband André. It is still a mystery to me how André managed to find time to proof-read parts of this thesis somewhere in between changing diapers and cooking delicious meals to keep me well-nourished in the last phase of the write-up. Last but not least, thanks are due to my son Johann who always managed to brighten me up with his smiles and who reminded me from time to time what life is all about.

Deutsche Zusammenfassung

Netzwerke im Sinne von Objekten, die in Beziehung zueinander stehen, sind allgegenwärtig. Paradebeispiele sind soziale Netzwerke, die Beziehungen zwischen Individuen modellieren, seien es Freundschaftsbeziehungen in Online-Communities wie Facebook oder Google+ oder persönliche Netzwerke, die Freunde, Arbeitskollegen oder Familienangehörige verbinden. Aber auch in vielen anderen Zusammenhängen treten Netzwerke auf: Produkte können miteinander gekauft werden, Proteine miteinander interagieren oder wissenschaftliche Publikationen andere Artikel zitieren, um nur einige Beispiele zu nennen. In vielen Bereichen kommt dabei dem Begriff einer dicht vernetzten Gruppe eine semantisch beachtenswerte Bedeutung zu. In den oben genannten Netzwerken können solche Gruppen Freundeskreisen, Gruppen ähnlicher Produkte, funktionellen Gruppen von Proteinen oder Wissenschaftsbereichen entsprechen.

Auf einer abstrakten Ebene werden Netzwerke als Graphen modelliert und dicht vernetzte Gruppen Cluster genannt. Grob gesprochen liegt dem Begriff Cluster zugrunde, dass der damit verbundene Teilgraph viele Kanten enthält und es gleichzeitig wenige Kanten gibt, die Knoten im Cluster mit dem restlichen Graphen verbinden. Da dieses Paradigma vergleichsweise vage ist, existieren verschiedene Formalisierungen, die versuchen, die Güte einer Clusterung zu quantifizieren.

In der vorliegenden Arbeit werden im Wesentlichen zwei verschiedene Ansätze betrachtet. Der erste bemisst Clusterungen anhand der der Dünne der durch die Cluster induzierten Schnitte, während der zweite Ansatz sich mit der Optimierung des Maßes Surprise [5] beschäftigt.

Schnittbasiertes Clustern. Eines der grundlegenden Probleme der Algorithmik ist die Bestimmung eines minimales Schnittes in einem Graphen, das heißt, einer Zerlegung des Graphen in zwei Teile, die mit einer minimalen Anzahl Kanten verbunden sind. Ein naiver Algorithmus zum Clustern von Graphen könnte dies benutzen, um den Graphen iterativ anhand von minimalen Schnitten zu zerlegen, so dass möglichst wenige Kanten getrennt werden. In der Praxis erweisen sich minimale Schnitte jedoch als problematisch, da diese oftmals nur einen oder weniger Knoten vom Rest des Graphen trennen. Deshalb empfehlen sich für Algorithmen zum Clustern von Graphen Maße, die die Größe der einzelnen Teile miteinbeziehen, wie die *conductance*, *expansion* oder *density* eines Schnittes. Aufbauend auf diesen Schnittmaßen können konkrete Separiertheitsmaße definiert werden, die bemessen, wie gut die einzelnen Cluster voneinander getrennt sind; dies bringt allerdings mehrere Freiheitsgrade mit sich.

Im ersten Teil der Arbeit werden systematisch alle möglichen Kombinationen dieser Freiheitsgrade untersucht, was zu einer Familie von Maßen für die Separiertheit von

Clustern führt. Wir betrachten das generische Problem, die Separiertheit der Cluster zu optimieren und gleichzeitig eine Gütegarantie an die Dichte der einzelnen Cluster nicht zu unterschreiten. Dazu wird eine agglomerative Heuristik für distanzbasiertes Clustern betrachtet [204], die in jedem Schritt zwei Cluster mit minimaler Distanz vereinigt und untersucht, inwieweit sich effiziente Algorithmen für diese Heuristik auf das Clustern von Graphen übertragen lassen. Um diese Frage zu beantworten, werden Eigenschaften identifiziert, die das dabei verwendete Gütemaß erfüllen muss und die betrachteten Maße im Hinblick auf diese Eigenschaften klassifiziert.

Die Intuition hinter dem betrachteten Optimierungsproblem ist, dass die schnittbasierten Maße große Cluster bevorzugen, die nur durch wenige Kanten verbunden sind, wohingegen Garantien an die Dichte leichter durch die Erzeugung vieler kleiner Cluster zu erfüllen sind. Für alle vorgeschlagenen Maße wird untersucht, ob diese Intuition zutreffend ist, indem überprüft wird, ob bei der Verwendung agglomerativer Heuristiken lokale Optima auftreten können.

Diese theoretischen Betrachtungen werden durch eine umfassende, empirische Studie ergänzt. Ein Vergleich der erwähnten Heuristik mit einer zweiten Heuristik, die im Wesentlichen auf dem Verschieben von Knoten und Teilmengen von Knoten zwischen Clustern basiert, zeigt, dass die letztere oft zu besserer Qualität führt. Im zweiten Teil der Experimente wird untersucht, wie sich die Wahl der verwendeten Maße für die Separiertheit von Clustern und die Dichte der einzelnen Cluster auf die Anzahl der Cluster, die Clustergrößenverteilung und das Verhalten in Hinblick auf synthetische Daten mit einer eingepflanzten Clusterstruktur auswirkt.

Surprise. Einen anderen Ansatz zum Clustern von Graphen liefern Clustermaße, die auf einem *Nullmodell* basieren. Die Idee ist hierbei, zu einem gegebenen Graphen und einer gegebenen Clusterung einen Zufallsgraphen auf derselben Knotenmenge zu betrachten, der einige Eigenschaften des Ausgangsgraphen beibehält. Nun wird die tatsächliche Anzahl der Kanten innerhalb von Clustern mit demselben Wert auf dem Zufallsgraphen verglichen. Ist dieser Wert wesentlich höher, als zu erwarten wäre, ist die Clusterung gut an den Graphen angepasst. Ein Clustermaß, das auf diesem Prinzip basiert und das von einer Reihe oft benutzter Algorithmen optimiert wird, ist die Modularity einer Clusterung [173]. Ein verwandtes, relativ neues Maß ist Surprise, das die Wahrscheinlichkeit minimiert, dass mindestens so viele Kanten zufällig innerhalb von Clustern erzeugt werden wie tatsächlich beobachtet werden. Experimentelle Studien [5, 7, 8] liefern Indizien dafür, dass dieses Maß einige der in letzter Zeit aufgekommenen Kritikpunkte an Modularity wie den Hang zu übermäßig großen Clustern in großen Graphen nicht zu teilen scheint.

Der zweite Teil der Arbeit widmet sich sowohl der Komplexität des Problems, eine Clusterung mit optimaler Surprise zu finden als auch praktischen Algorithmen, um dieses Problem zu lösen. Aufbauend auf der Beobachtung, dass Optimallösungen bezüglich Surprise immer Pareto-optimal bezüglich der Maximierung der Anzahl der Kanten und der Minimierung der Anzahl der Knotenpaare innerhalb von Clustern ist, wird gezeigt, dass das assoziierte Entscheidungsproblem $\mathcal{NP}$-vollständig ist, Optimallösungen in Graphen mit konstanter Baumbreite jedoch mit polynomieller Laufzeit gefunden werden können. Aus praktischer Sicht werden Methoden vorgeschlagen, die eine optimale Lösung auf kleinen Instanzen mit Hilfe von ganzzahligen linearen Programmen berechnen, sowie Algorithmen betrachtet, die heuristisch gute Lösungen für große Graphen finden. Beide

Ansätze werden experimentell evaluiert und mit einer bereits bestehenden Metaheuristik [9] verglichen. Zusätzlich wird ein neues Clustermaß abgeleitet und diskutiert, das Eigenschaften von Modularity und Surprise miteinander kombiniert.

Generierung von Testinstanzen. Wie bei allen algorithmischen Fragestellungen stellt sich bei der Evaluierung von Algorithmen zum Clustern von Graphen die Frage nach geeigneten Testinstanzen. Um die praktische Anwendbarkeit eines Algorithmus zu belegen, eignen sich hierzu in erster Linie Echtweltdaten aus verschiedenen Anwendungsbereichen. Es gibt jedoch auch eine Reihe von Gründen, die für die (zusätzliche) Verwendung von synthetischen Graphen sprechen. Synthetische Graphen können üblicherweise in jeder beliebigen Größe generiert werden, was hilfreich in Verbindung mit Skalierbarkeitsstudien ist. Weiterhin ermöglichen es Graphgeneratoren, einen Algorithmus auf einer großen Menge von Graphen mit ähnlichen Eigenschaften zu testen, was die zu starke Fokussierung und Anpassung auf wenige Testinstanzen verhindert. Bei der Evaluation von Algorithmen zum Clustern von Graphen bieten synthetische Daten darüber hinaus die Möglichkeit, dem Graphen bereits bei der Erzeugung eine Clusterstruktur einzupflanzen, die im Anschluss zur qualitativen Auswertung benutzt werden kann.

Görke und Staudt [104] schlagen einen Generator vor, der dynamische Zufallsgraphen mit einer eingebauten Clusterstruktur erzeugt, die sich ihrerseits über die Zeit ändert; dieser kann dazu benutzt werden, Algorithmen zum Clustern dynamischer Graphen [117] zu evaluieren. Im letzten Teil dieser Arbeit wird eine verbesserte Implementierung des Generators beschrieben, die zum einen die Laufzeit und zum anderen den Speicherverbrauch senkt, was die Generierung größerer Testdaten ermöglicht.

Contents

Acknowledgments iii

Deutsche Zusammenfassung v

1 Introduction 1
- 1.1 Overview . . . 2

2 Fundamentals 5
- 2.1 Graphs . . . 5
- 2.2 Clusterings . . . 7
- 2.3 Modularity . . . 9
- 2.4 Algorithms and Complexity . . . 10
- 2.5 Miscellaneous . . . 12

3 Agglomerative Algorithms for Graph Clustering 15
- 3.1 Agglomerative and Divisive Clustering Algorithms . . . 15
- 3.2 Generic Description of Agglomerative Algorithms . . . 17
- 3.3 Running Time Considerations . . . 20
  - 3.3.1 Greedy Merge (CNM) . . . 21
  - 3.3.2 Greedy Vertex Moving (Louvain) . . . 23
- 3.4 Related Work . . . 27

4 Density-Constrained Graph Clustering 31
- 4.1 Quality Measures for Clusterings . . . 32
  - 4.1.1 Intercluster measures . . . 32
  - 4.1.2 Intracluster measures . . . 34
  - 4.1.3 Qualitative Observations . . . 35
- 4.2 Problem Statement and Complexity . . . 35
- 4.3 Related Work . . . 36
  - 4.3.1 Dense Clusters . . . 37
  - 4.3.2 Clustering based on sparse cuts . . . 38
- 4.4 Generic Greedy Agglomeration . . . 41
  - 4.4.1 Properties of Intercluster Sparsity and Intracluster Density Measures 42
  - 4.4.2 Merge Behavior . . . 44
  - 4.4.3 Transferring SAHN Algorithms to DCC . . . 52
  - 4.4.4 Properties of Intracluster Measures . . . 58
  - 4.4.5 Properties of Intercluster Measures . . . 59
- 4.5 Concluding Remarks . . . 64

5 Experiments on Density-Constrained Graph Clustering 67
5.1 Greedy Vertex Moving for Density-Constrained Graph Clustering 68
5.1.1 Comparing Moves Efficiently 69
5.1.2 Dealing with Maximum Measures 70
5.1.3 Connected Moves 71
5.2 Experimental Comparison of Algorithms 72
5.2.1 Qualitative Comparison of Greedy Merge and Greedy Vertex Moving 72
5.2.2 Evaluating Balancedness using the Gini Coefficient 74
5.2.3 Cluster Size Distribution with GM and GVM 75
5.2.4 Effectiveness of Different Objective Functions 76
5.2.5 Reference Algorithms 77
5.2.6 Comparison Based on Intracluster Density Found by Reference Algorithms 77
5.2.7 Implementation and Running Times 80
5.3 Comparison of Density and Sparsity Measures 80
5.3.1 Random Graphs 80
5.3.2 Distance Measures 80
5.3.3 Parameters and Evaluation 81
5.3.4 Results on Planted Partition Graphs 81
5.3.5 Selected clusterings on small example network. 84
5.4 Concluding Remarks 85

6 Surprise - Complexity and Exact Solutions 87
6.1 Definition and Basic Properties 88
6.2 NP-Completeness 91
6.3 Trees and Bounded Treewidth 95
6.4 Exact Solutions for General Graphs 98
6.5 Combining Surprise and Modularity 102
6.6 Concluding Remarks 105
6.7 Proof of Lemma 6.4 107

7 Agglomerative Algorithms for Surprise Optimization 111
7.1 Solving Surprise Maximization via the Weighted Sum Method 112
7.2 Algorithms contained in SurpriseMe 113
7.3 Performance of Algorithms in SurpriseMe 117
7.4 Engineering the Weighted Sum Method 120
7.5 Scanning the Convex Hull Directly 123
7.6 Experimental Results 126
7.7 Concluding Remarks 129

8 An Efficient Dynamic Generator for Clustered Random Graphs 131
8.1 Related Work 133
8.2 Static Model for Planted Partition Graphs 134
8.3 Edge Dynamics 135
8.3.1 Associated Markov Chain and Distribution 135
8.3.2 Data Structures and Implementation 137
8.4 Vertex and Cluster Dynamics 143

8.5 Experiments 145
8.6 Concluding Remarks 145

9 Conclusion **147**

A Integer Linear Programs for Density-Constrained Graph Clustering **151**
A.1 Building Blocks for Density-Constrained Graph Clustering 151
A.2 Example Usage and Discussion 153
A.3 Modelling Equivalence Relations and Boolean Functions 154

B Additional Experiments from Chapter 5 **155**
B.1 Gini Coefficients GVM and GM: Additional Plots 155
B.2 Effectiveness of Different Objective Functions: Additional Plots 156
B.3 Complete Experiments with Planted Partition Graphs 158

Bibliography **167**

List of Publications **183**

Erklärung **185**

Nomenclature

$\binom{\mathcal{S}}{k}$	Set of k-element subsets of $\mathcal{S}$
$\mathcal{C}_{v \to D}$	Clustering after moving vertex v to cluster D in clustering $\mathcal{C}$
$\mathrm{mod}(\mathcal{C})$	Modularity of clustering $\mathcal{C}$
$\overline{e_C}$	Number of non-edges in cluster C
$\overline{G}$	Complement graph of G
$\overline{m}$	Number of non-edges
e_C	Number of edges in cluster C
$e_{A,B}$	Number of edges between clusters A and B
$\mathcal{C}(v)$	Cluster of vertex v
$\mathcal{C}_{\{A,B\}}$	Clustering after merging clusters A and B in clustering $\mathcal{C}$
d_C	Degree of cluster C
d_v	Degree of vertex v
$E(C)$	Set of edges in cluster C
$E(G)$	Set of edges of graph G
$G[V']$	Subgraph induced by $V' \subseteq V$
$i_e(\mathcal{C})$	Number of intracluster edges with respect to clustering $\mathcal{C}$
$i_p(\mathcal{C})$	Number of intracluster vertex pairs with respect to clustering $\mathcal{C}$
$i_{v^2}(\mathcal{C})$	Total sum of squared cluster volumes in $\mathcal{C}$
k	Number of clusters
m	Number of edges
n	Number of vertices
$N(v)$	Neighborhood of vertex v
n_C	Number of vertices in cluster C
p	Number of (unordered) vertex pairs, $\binom{n}{2}$

$p(C)$	Number of vertex pairs in cluster C
$S'(\mathcal{C})$	$-\log_{10} S(\mathcal{C})$
$S(\mathcal{C})$	Surprise of clustering $\mathcal{C}$
$V(G)$	Set of vertices of graph G
v_C	Volume of cluster C
x_C	Number of intercluster edges incident to cluster C
$x_e(\mathcal{C})$	Number of intercluster edges with respect to clustering $\mathcal{C}$
$x_p(\mathcal{C})$	Number of intercluster vertex pairs with respect to clustering $\mathcal{C}$

Chapter 1

Introduction

When people hear the term *network*, they often associate it with the concept of a *social network*, both referring to online networks as Facebook or Google+ and the more traditional type of social networks consisting of individuals and their personal acquaintances, family, and friends. Other examples for networks that are observable in every day life include computer networks that are build for sharing content or computational resources, and road networks. However, networks occur also in many other contexts; for example, biologists study networks of protein-protein interaction, of metabolites and their biochemical reactions, and of food webs modeling predator-prey relationships [128].

All these networks have in common that they consist of objects or actors, together with associations between them. In a mathematical context, networks are often modeled as *graphs*, consisting of a set of *vertices* representing the objects and a set of *edges* representing the associations. Groups in these networks, in the sense of sets of vertices that are connected to each other above the ordinary, often come with a meaning. For example, groups in social networks may correspond to circles of friends or people that share common interests, whereas groups in protein-protein interaction and metabolic networks often correspond to functional units [128]. Such groups are commonly referred to as either clusters or communities, depending on the field of application. Accordingly, methods that detect clusters, given the underlying graph, are usually called graph clustering or community detection algorithms.

In this general form, clustering algorithms can be applied in diverse disciplines; for example, in computational biology, clusters in the above mentioned metabolic networks can be used to annotate uncharacterized proteins. Other less well-known applications of graph clustering include clustering as a preprocessing step in graph drawing algorithms [165], speeding up routing in Mobile Ad hoc Networks (MANETS) [123], and recommendation systems [200].

Algorithms classifying data have a long history [125]. What distinguishes graph clustering from most of the earlier literature is primarily the type of input. While classical approaches are mainly concerned with clustering attributed data that can be represented by points in multi-dimensional space, graph clustering approaches only consider the associations between objects, which do not have to be metric; often, the only information is if two objects are related or not, which corresponds to the concept of an unweighted graph. Although this distinction might seem technical, graph clustering or community detection has grown into an independent field in the course of the last twenty years.

Giving a comprehensive overview of the plenitude of existing graph clustering algorithms is beyond the scope of this thesis. We will discuss related work in the particular chapters and refer to the surveys of Fortunato [83] and Schaeffer [192] for more information. In spite of this abundance of different methods, most of them are implicitly or explicitly based on the intuitive idea that there should be many edges within clusters and only few between them, which is sometimes referred to as the *intracluster density vs. intercluster sparsity paradigm*. As this paradigm is rather vague, there are different measures that try to capture the goodness of a clustering, probably the most popular of them being the modularity of a clustering [173].

1.1 Overview

Since modularity is not without criticism (cf. Section 2.3), the main part of this thesis is devoted to two alternative approaches, together with algorithms optimizing them. In the first part, we consider a family of measures based on the sparsity of the cuts induced by the individual clusters, together with constraints on the density of the individual clusters. In the second part, we investigate the recently suggested measure surprise [15], which is based on a similar idea as modularity, but leads to smaller clusters and a less uniform cluster size distribution in practice. In the following, we give an overview on the structure of this thesis and briefly summarize the main contributions.

Chapter 2: Fundamentals

This chapter introduces notation and concepts that are needed throughout this thesis. Besides briefly reviewing topics related to basic graph theory and the complexity of algorithms, we state more precisely what we mean by clustering a graph and discuss the measure modularity.

Chapter 3: Agglomerative Algorithms for Graph Clustering

Independent of any particular clustering measure, we review two basic metaheuristics that have been used in the literature to optimize objective functions in the context of graph clustering The first one is based on greedily merging clusters, whereas the second moves vertices between clusters on several levels of abstraction of the graph. We discuss similarities of the former with the classical approach of SAHN clustering [204] used in data mining applications, and exemplify how running times for both approaches can be estimated in the case of modularity clustering. In the course of this discussion, we derive a weak lower bound on the worst case complexity of the Louvain method [32]. We conclude the chapter by an overview on agglomerative graph clustering algorithms from the literature.

Chapter 4: Density-Constrained Graph Clustering

One of the fundamental algorithmic problems in graph theory is the computation of a minimum edge cut in a graph. A naïve clustering algorithm might use this to recursively split the graph according to the partition induced by a minimum cut. In practice, minimum cuts are problematic, as they often separate one or few vertices from the remainder

of the graph, which leads to clustering with a very skew cluster size distribution. Hence, measures that take the size of the two parts into account, like the *expansion*, *conductance*, or *density* or a cut, are usually more appropriate. All of these cut measures have been coined before in the context of clustering [79, 115, 130]; however, using them to construct concrete measures to evaluate whole clusterings leaves several degrees of freedom.

In this chapter, we explore all combinations of these degrees of freedom to systematically assemble a set of concrete measures for intercluster sparsity, which are based on the sparsity of the cuts induced by the clusters. To derive a family of optimization problems from these, we consider the problem of optimizing intercluster sparsity while retaining guarantees on the intracluster density of the clustering, which we term Density-Constrained Clustering (DCC). The intuition behind this formalization is that intercluster sparsity measures strive towards coarse clusterings, whereas intracluster density favors small clusters. We investigate whether this intuition is correct by determining whether each of the proposed measures might lead to local optima when used in the context of agglomerative algorithms. In the second part, we identify properties of intercluster sparsity and intracluster density measures that render the application of efficient algorithms for SAHN clustering to DCC possible, and obtain an almost complete classification of the proposed measures with respect to these properties. Part of this chapter is based on joint work with Robert Görke and Dorothea Wagner [101, 102].

Chapter 5: Experiments on Density-Constrained Graph Clustering

We complement the theoretical considerations in Chapter 4 with a comprehensive experimental evaluation. A comparison of the metaheuristics based on moving vertices and merging clusters on real-world instances shows that the former often leads to better quality in the context of DCC, while being fast in practice. We further validate the appropriateness of this metaheuristic by comparing the quality of the resulting clusterings to general clustering algorithms from the literature with respect to the objective of DCC. In the second part, we investigate to what extent the choice of intercluster sparsity and intracluster density measure influences the number of clusters, the cluster size distribution, and the ability to reconstruct a hidden clustering in synthetic benchmark data. This chapter is based on joint work with Robert Görke and Dorothea Wagner [96, 103].

Chapter 6: Surprise - Complexity and Exact Solutions

An orthogonal view on graph clustering is represented by clustering measures that are based on a *null model*. The idea behind these is to compare the observed number of edges within clusters, which we call *intracluster edges*, to the number of intracluster edges in a random graph G that inherits some of the properties of the original graph G'. If the number of intracluster edges is substantially larger than expected, this indicates that the clustering is well adapted to the actual link structure in the graph. The recently proposed measure surprise quantifies the probability that G' contains at least as many intracluster edges as G. Experimental studies [5, 7, 8] suggest that surprise works well with respect to synthetic benchmarks, and does not suffer to the same extent from modularity's inclination to produce overly large clusters when applied to large sparse networks.

In this chapter, we take first steps towards a theoretical analysis of surprise. We observe that optimal solutions with respect to surprise are Pareto optimal with respect to maximizing the number of intracluster edges and minimizing the number of vertex pairs within clusters. This bicriteria view can be exploited to show that the associated decision problem is $\mathcal{NP}$-complete in general, but allows for polynomial time algorithms in graphs of bounded tree width. From a more practical point of view, we propose and evaluate methods that compute optimum solutions on small instances using a sequence of integer linear programs. Furthermore, we discuss a novel clustering measure that can be seen as a compromise between modularity and surprise. Part of this chapter is based on joint work with Tobias Fleck and Dorothea Wagner [81, 82].

Chapter 7: Agglomerative Algorithms for Surprise Optimization

As the exact methods from Chapter 6 are infeasible on large instances, there is still need for heuristic algorithms that find clusterings of good surprise efficiently. We propose to exploit the bicriteria view introduced in Chapter 6 and optimize a weighted sum of the number of intracluster pairs and edges, using a sequence of different weight parameters. An experimental evaluation of the existing metaheuristic SurpriseMe [9] shows that the quality of its results is implicitly due to the fact that it integrates the weighted sum method, although with a different motivation in mind. We take this as a starting point to further engineer this approach, both with respect to quality and running time.

Chapter 8: An Efficient Dynamic Generator for Clustered Random Graphs

The selection of suitable test instances is one of the core challenges when evaluating algorithms. To prove the practical applicability of algorithms, it is common practice to use real world data from different fields of application. There is however a list of reasons that speak in favor of the (additional) use of synthetic data. Synthetic graphs can usually be generated in any possible size, which is helpful in scalability studies. Furthermore, testing an algorithm on a large set of graphs with similar properties allows for a certain degree of significance. When evaluating graph clustering algorithms, synthetic data additionally allow for the incorporation of a hidden cluster structure that can be subsequently used in a qualitative evaluation.

Görke and Staudt [104] propose a generator for dynamic random graphs with a hidden reference clustering that itself changes over time. We describe an enhanced implementation of this generator, which differs in the data structures used and therefore allows for faster practical and worst case running times, as well as linear space requirements. This chapter is based on joint work with Robert Görke, Roland Kluge, Christian Staudt, and Dorothea Wagner [97, 98]; part of the related work section will appear in our survey on dynamic graph clustering [117].

Chapter 9: Conclusion

This chapter contains a summary of the main insights obtained in this thesis. We briefly discuss similarities between the two main approaches to graph clustering we considered, in the light of the bicriteria view introduced in Chapter 6, and close with a short outlook.

Chapter 2

Fundamentals

In this chapter, we introduce some notation, definitions, and concepts that are used throughout this thesis. In the first part, we review basic concepts from graph theory. Readers familiar with this topic may safely skip this part and only use it as reference in case some of the terms and symbols we use are unclear. In the second part, we clarify what we mean by clustering a graph and introduce some clustering specific notation. We then introduce the quality measure modularity, as it is one of the most popular approaches to graph clustering and will occur as example or reference in most of the following chapters. We proceed by recapitulating terms in connection with the complexity of algorithms, and conclude this chapter with some definitions we need that do not fit in any of the preceding categories. As a quick way to recall the abbreviations we introduce in this chapter, we refer to the nomenclature. The definitions we give make frequent use of the following basic notation on sets and subsets.

Sets and Binomial Coefficients. Let $\mathcal{S}$ be a set of objects. We denote by $\binom{\mathcal{S}}{k}$ the set of all k-element subsets of $\mathcal{S}$. In particular, $\binom{\mathcal{S}}{2}$ is the set of all (unordered) pairs of objects in $\mathcal{S}$. To ease notation, we will allow binomial coefficients $\binom{n}{k}$ for all n and $k \in \mathbb{N}_0$. If $k > n$, $\binom{n}{k} = 0$ by definition.

2.1 Graphs

This section contains basic definitions from graph theory. We orient ourselves at the notation of two standard books on network analysis [36, 64]; these provide also a good reference for further information on concepts related to graphs or networks.

Basic Definitions. A finite *graph* $G = (V, E)$ is a tuple consisting of a set V of *vertices* (nodes) and a set E of *edges linking* or *connecting* pairs of vertices. The two vertices connected by an edge e are called its *endvertices*, and we say that u and v are *incident* to e. The vertex set and edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively. Unless otherwise stated, we denote by $n := |V|$ and by $m := |E|$ the number of vertices and edges in G, respectively. We sometimes also use the abbreviation $p := \binom{n}{2}$ for the number of unordered vertex pairs in G. Graphs can be *undirected* or *directed*. In undirected graphs, edges are *undirected*, i.e., the order of the endvertices of an edge is immaterial. An undirected edge linking two vertices u and $v \in V$ is denoted by $\{u, v\}$. In directed graphs, edges are *directed*, i.e., each edge has a *source* and a *target vertex*.

A directed edge with source u and target v is represented by an ordered pair (u, v). In both undirected and directed graphs, we may allow the edge set E to contain the same edge more than once, i.e., E can be a multiset. If an edge occurs several times in E, the copies of that edge are called *parallel edges*. Furthermore, in some cases we may abuse notation and allow edges of the form $\{v, v\}$ or (v, v) linking a vertex with itself; such edges are called *loops*. A graph is *simple* if it does not contain loops or parallel edges. An *(edge) weighted* graph is an ordered triple (V, E, ω) such that (V, E) is a directed or undirected graph and $\omega\colon E \to \mathcal{D}$ a mapping from the set of edges to some domain $\mathcal{D}$, usually $\mathbb{Z}$ or $\mathbb{Q}$. For most purposes, an unweighted graph $G = (V, E)$ is equivalent to a weighted graph with *unit edge weights* $\omega(e) = 1$ for all $e \in E$. Unless otherwise stated, we will only consider undirected, unweighted, finite and simple graphs in this thesis,

Adjacency, Subgraphs and Cliques. If two vertices are connected by an edge, they are *adjacent* and we call them *neighbors*. For a vertex v, we denote by $N(v)$ the set of its neighbors, the *neighborhood* of v. Its cardinality $|N(v)|$ is called the *degree* d_v of v. A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. If E' contains all edges between vertices in V', G' is called the subgraph *induced* by V'. The subgraph induced by V' in G is denoted by $G[V']$. If $E = \binom{V}{2}$, the undirected graph $G = (V, E)$ is called *complete*. A subgraph that is complete is called a *clique*. A *maximal clique* $G' = (V', E')$ is a clique that cannot be further expanded, i.e., a clique such that there is no vertex $v \in V \setminus V'$ that is connected to all vertices in V'. In contrast to that, a *maximum clique* is a clique with maximum cardinality in G.

Paths and Cycles. A *walk* from x_0 to x_k in a graph $G = (V, E)$ is an alternating sequence $x_0, e_1, x_1, e_2, x_2, \ldots, x_{k-1}, e_k, x_k$ of vertices and edges, where $e_i = \{x_{i-1}, x_i\}$ in the undirected case and $e_i = (x_{i-1}, x_i)$ in the directed case. The vertex x_i is called the *successor* of x_{i-1} and k the *length* of the path. A *path* is a walk where $e_i \neq e_j$ for $i \neq j$; it is *simple*, if $x_i \neq x_j$ for $i \neq j$. A path with $x_0 = x_k$ that contains at least two edges is a *cycle*. In an undirected, unweighted graph, the *distance* between two vertices u and v is the minimum length of any path connecting them.

Connectivity. An undirected graph is *connected* if any two of its vertices are linked by a path. A maximal connected subgraph of a graph $G = (V, E)$, i.e., a connected subgraph $G' = (V', E')$ such that there are no edges in E linking vertices in V' to vertices in $V \setminus V'$, is called a *connected component* of G. An *(edge) cut* is a partition $(S, \overline{S})$ of V into two non-empty subsets S and S'; sometimes, cuts are also identified with the set of edges *crossing the cut*, i.e., the set of edges $\{u, v\}$ such that $u \in S$ and $v \in S'$. A *(vertex) separator* is a set S of vertices such that the subgraph induced by $V \setminus S$ is not connected. If vertices u and v are contained in different connected components of this subgraph, S is called a *(vertex) separator* of u and v. It is a *minimum vertex separator* if it has minimum cardinality among all vertex separators of u and v.

Forests and Trees. An undirected graph that does not contain any cycles is called a *forest*. If it is additionally connected, we call it a *tree*. The term *leaves* refers to vertices of degree 1 in a tree. Sometimes it is convenient to mark a special vertex in a tree T and call it its *root*; in this case, T is a *rooted tree*. In a rooted tree, we can think of the vertices ordered at several levels according to their distance to the root. The number of this levels minus 1 is called the *height* of T; equivalently, we can define the height as the maximum distance between a leaf and the root. For a vertex v in a tree rooted at r, the subtree rooted at v consists of the set of vertices u such that v is contained in the (unique) shortest path from r to u. A *binary tree* is a rooted tree such that every vertex v is either a leaf or it is connected to exactly two vertices in the subtree rooted at v.

Complement Graph and Contraction. For a given graph G its *complement graph* $\overline{G}$ is defined as $\overline{G} = (\overline{V} = V, \overline{E} = \binom{V}{2} \setminus E)$. The pairs of vertices in $\overline{E}$ are called *non-edges* of G and $\overline{m} := |\overline{E}|$. We will sometimes use the expression that a graph is modified by *contracting* a subset C of vertices. By this, we mean that we replace the vertices in C by a single vertex, their *supernode* v. We connect a vertex u in $V \setminus C$ with v iff there is an edge in the original graph connecting u with a vertex $w \in C$. Depending on the specific context, the modified graph may also contain some summarized information about the original graph; for example, edge weights might reflect the number of edges between the subsets of vertices represented by its incident supernodes in the original graph, or vertex weights the number of represented vertices.

Treewidth. *Treewidth* is a property of graphs that has been introduced by Robertson and Seymour [183] in the 1980's. It generalizes the definition of trees in the sense that trees are the only graphs that have a minimum treewidth of 1. Treewidth has been since then frequently occurred in the literature on graph theory [34]. On the one hand, graphs of small treewidth can be iteratively decomposed into smaller graphs using small vertex separators, which allows for the application of dynamic programming algorithms based on this decomposition. On the other hand, several well-known classes of graphs, as for example series parallel graphs, exhibit small treewidth, such that these algorithms are applicable. We will briefly encounter treewidth in Chapter 6 and therefore give its definition here; for a more thorough introduction to treewidth and to algorithms for graphs with bounded treewidth, we recommend the book by Klocks [137] and the survey by Bodlaender and Koster [34]. For the definition of treewidth, we need the concept of *tree decompositions*.

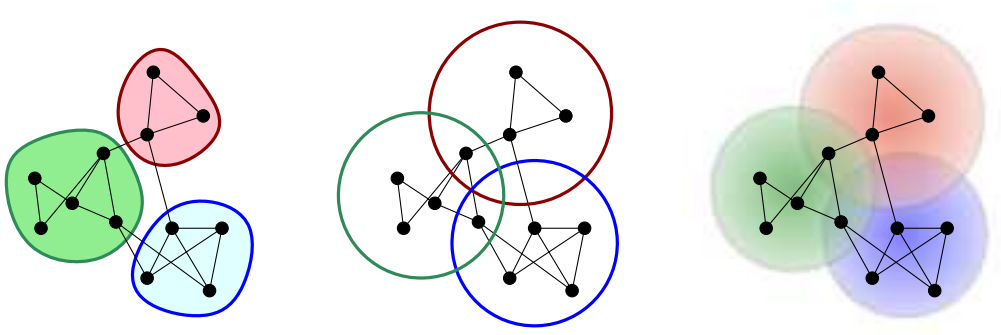
Definition 2.1. A *tree decomposition* of a graph $G = (V, E)$ is a pair $\big(T = (I, F), \{X_i \mid i \in I\}\big)$, where T is a tree and each node $i \in I$ has associated to it a subset of vertices $X_i \subseteq V$, called the bag of i, such that

1. Each vertex belongs to at least one bag: $\bigcup_{i \in I} X_i = V$.
2. For all edges, there is a bag containing both its endpoints, i.e., for all $\{v, w\} \in E$ there is an $i \in I$ with $v, w \in X_i$.
3. For all vertices $v \in V$, the set of nodes $\{i \in I \mid v \in X_i\}$ induces a subtree of T.

The width of a tree decomposition $\big(T = (I, F), \{X_i \mid i \in I\}\big)$ is $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. Graphs with treewidth at most k are called *partial k-trees*.

2.2 Clusterings

As introduced in Chapter 1, from a high level point of view, *clusters* are sets of vertices such that their induced subgraph contains many edges. A *clustering* is a set of clusters. Mathematically, the concrete definition of a clustering is ambiguous and may refer to either *disjoint*, *overlapping*, or *fuzzy* clusterings. The first definition requires the clusters to form a partition of the vertex set, while the second allows clusters to overlap. Clusters in fuzzy (or *soft*) clusterings may also overlap; additionally, each vertex maintains for each of its containing clusters a weight indicating to which extent the vertex the belongs to the cluster. These concepts are illustrated in Figure 2.1.



(a) Disjoint (b) Overlapping (c) Fuzzy

FIGURE 2.1: Different types of clusterings

Although overlapping [62] and fuzzy [229] graph clustering algorithms have been considered in the literature, most of the approaches find disjoint clusters [83]; in this thesis, we restrict ourselves to this case. In the following, we will introduce some notation concerning clusters that we will frequently use throughout all of the chapters. We tried to use standard definitions whenever possible, but, to increase readability, we also introduce some special notation in connection with merging clusters or moving vertices between clusters.

Clustering and Cluster. We define a *clustering* $\mathcal{C} = \{C_1, \dots, C_k\}$ of an undirected graph $G = (V, E)$ as a partitioning of V into *clusters*, i.e., $C_1 \cup \ldots \cup C_k = V$ and for all $C_i \neq C_j \in \mathcal{C}$, $C_i \cap C_j = \emptyset$. Mathematically speaking, a cluster is not more than a subset of V. There is however usually a subtle undertone when we refer to a subset as a cluster, as this often means that the cluster adheres to the intracluster density vs. intercluster sparsity paradigm introduced in Chapter 1. We will furthermore often identify a cluster with the subgraph induced by its vertices. For a vertex v, $\mathcal{C}(v)$ denotes the cluster containing v. An edge $\{u, v\}$ is called *intracluster edge*, if $\mathcal{C}(u) = \mathcal{C}(v)$, otherwise it is an *intercluster edge*. If $|\mathcal{C}| = 1$, $\mathcal{C}$ is called the *all clustering*, and if $|\mathcal{C}| = |V|$, it is called the *singleton clustering*. In both cases, we say that $\mathcal{C}$ is *trivial*. A clustering $\mathcal{C}$ is a refinement of a clustering $\mathcal{D}$, if any cluster in $\mathcal{D}$ is a union of clusters in $\mathcal{C}$. $\mathcal{C}$ is a *proper* refinement of $\mathcal{D}$, if additionally $\mathcal{C} \neq \mathcal{D}$. If not defined otherwise, the variable k will always refer to the number of clusters in $\mathcal{C}$.

Properties of Clusters. If C is a cluster, we denote by $n_C := |C|$ the number of vertices it contains. If $n_C = 1$, C is called a *singleton*. The *volume* v_C of C is defined as the sum of the vertex degrees in C, $v_C := \sum_{v \in C} d_v$. In contrast to that, the *degree* d_C of C indicates the number of clusters $B \neq C$ such that there exists an edge $\{u, v\}$ with $\mathcal{C}(u) = C$ and $\mathcal{C}(v) = B$. In this case, B is called a *neighbor* of C. The set of intracluster edges within C, i.e., the edges in the subgraph induced by C is denoted by $E(C)$. Its cardinality $|E(C)|$ is abbreviated by e_C. Moreover, we denote by $p_C := \binom{n_C}{2}$ the number of unordered vertex pairs within C. The total number of intracluster edges with respect to $\mathcal{C}$ is then denoted by $i_e(\mathcal{C}) := \sum_{C \in \mathcal{C}} e_C$ and the total number of vertex pairs within clusters by $i_p(\mathcal{C}) := \sum_{C \in \mathcal{C}} p_C$. Similarly, $x_e(\mathcal{C}) := m - i_e(\mathcal{C})$ denotes the number of intercluster edges and $x_p(\mathcal{C}) := \binom{n}{2} - i_p(\mathcal{C})$ the number of intercluster pairs in $\mathcal{C}$. If the clustering is clear from the context, we will sometimes omit $\mathcal{C}$ and just write i_p, i_e and x_e. For two clusters A and B, $e_{A,B} := |\{\{u, v\} \in E \mid u \in A, v \in B\}|$ is the number of edges between A and B. Using this, the number of intercluster edges incident

to C can be written as $x_C := e_{C,V\setminus C}$. $\overline{e_C} = \left|\binom{V(C)}{2} \setminus E(C)\right|$ is the number of *intracluster non-edges* of C.

Merge and Move Notation. A large part of this thesis is devoted to greedy algorithms for graph clustering based on merging clusters and on moving vertices between clusters. For this reason, we introduce some specific notation to increase the readability of the associated equations. Let $\mathcal{C}$ be a clustering. Then, for $A \neq B \in \mathcal{C}$, we call $\{A, B\}$ a *merge* and abbreviate $AB := A \cup B$. We call $\mathcal{C}_{\{A,B\}} := \mathcal{C} \setminus \{A, B\} \cup \{AB\}$ the *result of this merge*. To increase readability, we sometimes use $\mathcal{C}_{A,B}$ instead of $\mathcal{C}_{\{A,B\}}$. Furthermore, the clustering that results from moving vertex v to cluster D, i.e. $\big(\mathcal{C} \setminus \{\mathcal{C}(v), D\}\big) \cup \{\mathcal{C}(v) \setminus v, D \cup \{v\}\}$, is abbreviated by $\mathcal{C}_{v \to D}$.

2.3 Modularity

Similarly to the variety of conceptually different graph clustering algorithms in the literature, the question how to measure the goodness of a given clustering is answered ambiguously, leading to different *clustering measures*. A clustering measure is a function that maps clusterings to rational numbers, thereby assessing the quality of a clustering.

In the course of this thesis, we will encounter three different approaches: the clustering measures *modularity* and *surprise* and a family of clustering measures derived from the density of the different clusters and the sparsity of the cuts induced by the clustering. In this section, we will only define modularity, as it is the most popular measure in the recent literature on graph clustering and will occur in examples or as a reference throughout this thesis. Definitions of the latter two approaches can be found in Chapter 4 and Chapter 6.

Modularity builds upon the index coverage, which measures the fraction of edges contained within clusters. The coverage $\mathrm{cov}(\mathcal{C})$ of a given clustering $\mathcal{C}$ of a graph G is hence given by

$$\mathrm{cov}_G(\mathcal{C}) = \sum_{C \in \mathcal{C}} \frac{e_C}{m}.$$

To compute the modularity of a given clustering $\mathcal{C}$ on a graph $G = (V, E)$, we compare the coverage of $\mathcal{C}$ in G with the coverage of $\mathcal{C}$ in a random graph $G' = (V, E')$ on the same set of vertices. The distribution of G' arises from an imaginary random process which first removes all edges from G and then redistributes them randomly in the graph, such that the expected degree of each vertex in G' equals its degree in G. In G', it may happen that we draw the same vertex pair or vertex more than once, which may lead both to the existence of parallel edges and loops. This random process is usually referred to as the *configuration null model*. A more detailed description, including comprehensive examples, can be found in Section 2.3 of [94]. Modularity is now defined as the difference between the coverage of $\mathcal{C}$ in G and its expected coverage in G', which leads to the following term:

$$\mathrm{mod}(\mathcal{C}) = \mathrm{cov}_G(\mathcal{C}) - \mathbb{E}\left[\mathrm{cov}_{G'}(\mathcal{C})\right] = \sum_{C \in \mathcal{C}} \frac{e_C}{m} - \sum_{C \in \mathcal{C}} \frac{{v_C}^2}{4m^2}.$$

Modularity can be generalized to weighted [172] and directed [14, 147] graphs, to overlapping or fuzzy clusterings [174, 198], and to a local scenario, where the goal is to evaluate single clusters [50, 52, 158]. For our purposes, the generalization to weighted graphs is of interest, as we will encounter weighted graphs in the context of multilevel algorithms, where we will contract subsets of vertices to supernodes. If edge weights in the contracted graph always summarize the number of underlying edges in the original graph and each supernode has a self loop weighted by the number of edges in the subgraph induced by the vertices it represents, the weighted modularity of a clustering on the contracted graph always equals the modularity of the induced clustering on the original graph. Hence, optimizing the modularity in these aggregate graphs implicitly optimizes modularity in the original graph. Let the weight $\omega(v)$ of a vertex v be defined as

$$\omega(v) = \begin{cases} \sum_{\{u,v\} \in E} \omega(\{u,v\}), & \text{if } \{v,v\} \notin E \\ \sum_{\{u,v\} \in E, u \neq v} \omega(\{u,v\}) + 2 \cdot \omega(\{v,v\}), & \text{if } \{v,v\} \in E \end{cases}$$

With that, the modularity of edge weighted graphs can be expressed as

$$\text{mod}(\mathcal{C}) = \sum_{C \in \mathcal{C}} \frac{\sum_{e \in E(G[C])} \omega(e)}{\sum_{e \in E} \omega(e)} - \sum_{C \in \mathcal{C}} \frac{\left(\sum_{v \in C} \omega(v)\right)^2}{4\left(\sum_{e \in E} \omega(e)\right)^2}.$$

One of the main points of criticism in connection with modularity is its *resolution limit* [84]. This refers usually to the property of modularity that for a connected graph with a certain number of edges, the number of clusters in an optimal clustering is limited by a term which grows with the square of the number of edges in the graph. In other words, the size of the clusters grows with the graph size, which is not desirable in a range of applications. Ways to deal with this include the examination of finer, intermediate clusterings that occur as local maxima during agglomeration [32], the application of a preprocessing step which assigns suitable weights to the edges of an unweighted graph [28], clustering the subgraphs induced by these large clusters [84, 190], or the introduction of a parameter weighing the two terms in the modularity equation against each other, which leads to *multiresolution modularity* [182]. Multiresolution methods using a fixed weight parameter do not circumvent the resolution limit [141]; a possible remedy is to choose the parameter depending on the graph size [212]. Van Traag et al. [212] give a more rigorous definition of what they mean by *resolution-limit-free* by investigating whether clustering a subgraph induced by several clusters in an optimal clustering with respect to a measure yields the same clusters as in the original graph. We will briefly refer to the resolution limit in the context of surprise optimization (Chapter 6) and the optimization of intercluster sparsity measures in the presence of constraints on the intracluster density of a clustering (Chapter 5). In both cases, it is possible to obtain smaller clusters as with a modularity based approach, but the methods are not resolution-limit-free in this strong sense.

2.4 Algorithms and Complexity

This section briefly reviews fundamental complexity classes and the standard cost model to express the complexity of algorithms. For a more thorough introduction to the topic, we refer to the textbook of Garey and Johnson [88].

Decision Problems, P and NP. A *decision problem* $\Pi = (D_\Pi, Y_\Pi)$ consists of a set D_Π of instances together with a subset Y_Π of *yes-instances*. The elements of $D_\Pi \setminus Y_\Pi$ are called *no-instances*. A *polynomial time algorithm* for Π is an algorithm $\mathcal{A}$ such that its running time on any instance I in D_Π is bounded by a polynomial in the size of I, and the algorithm decides if I is a yes-instance. The class $\mathcal{P}$ consists of all decision problems Π such that a deterministic polynomial time algorithm for Π exists. Furthermore, $\mathcal{NP}$ denotes the set of decision problems that allow for a non deterministic polynomial time algorithm to solve them. Informally, this means that a problem is in $\mathcal{NP}$ if there is a deterministic, polynomial time algorithm $\mathcal{A}$ such that there is for any yes-instance I a proof such that $\mathcal{A}$ can verify that I is indeed a yes-instance. Obviously, $\mathcal{P} \subseteq \mathcal{NP}$. If $\mathcal{NP}$ is a subset of $\mathcal{P}$ is still an open problem, but it is widely believed that $\mathcal{P} \neq \mathcal{NP}$.

NPC. A *polynomial transformation* from a decision problem $\Pi_1 = (D_{\Pi_1}, Y_{\Pi_1})$ to another decision problem $\Pi_2 = (D_{\Pi_2}, Y_{\Pi_2})$ is a deterministic polynomial time algorithm that transforms any instance I in D_{Π_1} to an instance I' in D_{Π_2} such that $I \in Y_{\Pi_1}$ if and only if $I' \in Y_{\Pi_2}$. We say that $\Pi_1 \propto \Pi_2$ if there exists a polynomial transformation from Π_1 to Π_2. In this case, Π_2 is in a sense "at least as difficult to solve" as Π_1, as Π_1 can be solved in polynomial time if there is a polynomial time algorithm for Π_2. We say that a decision problem Π is $\mathcal{NP}$-complete, if $\Pi \in \mathcal{NP}$ and for every problem $\Pi' \in \mathcal{NP}$, it holds that $\Pi' \propto \Pi$. The set of all $\mathcal{NP}$-complete decision problems is denoted by $\mathcal{NPC}$. In practice, it suffices to show that $\Pi' \propto \Pi$ for a single decision problem Π' that is known to be $\mathcal{NP}$-complete, as $\propto$ is transitive. There is a large list of problems known to be in $\mathcal{NPC}$ (cf. for example [88]), derived from Cook's theorem [57] stating that the decision problem SAT is $\mathcal{NP}$-complete.

Optimization Problems. A (combinatorial) *optimization problem* $\Pi = (\mathcal{I}, f, m, g \in \{\min, \max\})$ consists of a set of instances I, a function f which maps any instance $I \in \mathcal{I}$ to a set of feasible solutions, a measure m, which maps any feasible solution to a real number, and a goal function g which signifies if we aim to minimize or maximize the measure. Given an instance, the goal is then to find a feasible solution minimizing (maximizing) m. A concept very similar to polynomial transformations can be applied to show that some optimization problems are "hard to solve", called *Turing reductions*. A Turing reduction from a problem $\Pi' \in \mathcal{NP}$ to an optimization problem Π is a polynomial time deterministic algorithm that is able to solve Π' with the help of an (imaginary) black box solver (the *oracle module*) that can be used to compute solutions for Π in constant time. If a Turing reduction from Π' to Π exists, we say that $\Pi' \propto_T \Pi$. If for any, $\Pi' \in \mathcal{NP}$, it holds that $\Pi' \propto_T \Pi$, Π is called $\mathcal{NP}$-hard.

Big O Notation. To describe the asymptotic growth of functions, we will frequently make use of the *big O notation*. Let f and g be two functions from $\mathbb{N}$ to a subset of the real numbers, usually to $\mathbb{N}$. We say that $f \in O(g(x))$ iff there exist positive constants C and x_0 such that for all $x > x_0$, we have $f(x) \leq Cg(x)$. Similarly, $f \in o(g)$ iff for all $C > 0$ there exists a positive constant x_0 such that for all $x > x_0$, we have $f(x) \leq Cg(x)$. If $f \in O(g)$, we say that $g \in \Omega(f)$. Similarly, $g \in \omega(f)$ iff $f \in o(g)$. If $f \in O(g) \cap \Omega(g)$, we say that $f \in \Theta(g)$.

Assumptions on Time Complexity. Although the above definitions of the classes $\mathcal{P}$ and $\mathcal{NP}$ depend on the computation model of Turing machines, it is well established to measure the complexity of algorithms with respect to a *random access machine (RAM)* that allows for the addition, subtraction, multiplication and division of integer numbers in one unit of time (cf. for example the model in [17]). This is often called the *uniform*

cost model. Note that the restriction to integer numbers is not a large impediment, as we may represent rational numbers by tuples consisting of the numerator and denominator and emulate all basic operations by a constant number of integer operations on a RAM. Although the uniform cost model is widely used, it is probably not equivalent to Turing machines in the sense that problems that are polynomially solvable on a RAM with uniform costs are polynomially solvable on a Turing machine. The problem is that recursive multiplication can lead to very high numbers, such that a Turing machine cannot process them in polynomial time. There are three ways to circumvent this problem [17]: we can either restrict the basic operations of the RAM to additions and subtractions, make sure that all numbers occurring during the algorithm have at most a logarithmic number of bits, or use a *logarithmic cost model* instead of the uniform one. In the latter model, the costs of basic integer operations are proportional to the length of the involved numbers in bits. Unless otherwise noted, we will always use the uniform cost model to assess the time complexity of algorithms and make sure that the occurring numbers are "small enough". An exception is in Chapter 6 in connection with the clustering measure surprise, where the associated numbers can be exponentially large in the size of the input. There, we will briefly switch to the logarithmic cost model to justify that the decision problem associated with surprise optimization is in $\mathcal{NP}$. In Chapter 8, we will additionally assume that we can compute logarithms and draw random numbers between 0 and M, where M is polynomial in the size of the input, in constant time.

To assess the asymptotic time complexity of graph algorithms, we will always assume that graphs are stored in adjacency lists. More precise, we assume that we can iterate over the incident edges of a given vertex v in time linear in d_v. In contrast to that, deciding if two given vertices u and v are connected is in $\min\{d_u, d_v\}$, unlike for adjacency matrices. In practice, we use arrays instead of lists.

2.5 Miscellaneous

Convex and Concave Functions. Let $f\colon \mathbb{R}^d \to \mathbb{R}$ be a function. We say that f is *convex*, if for all $x_1, x_2 \in \mathbb{R}^d$ and for all $t \in [0,1]$, we have

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

It is called *strictly convex* if for all $x_1 \neq x_2 \in \mathbb{R}^d$ and all $t \in (0,1)$, it is

$$f(tx_1 + (1-t)x_2) < tf(x_1) + (1-t)f(x_2)$$

A function f is called *(strictly) concave* if $-f$ is (strictly) convex.

Markov Chains. We will briefly encounter Markov chains in Chapter 8 and therefore review some basic definitions here; a more comprehensive introduction to Markov chains can be found for example in the book of Behrends [25]. A *Markov chain* is a pair $M = (S, P)$, where S is a finite set of *states* and P a row stochastic matrix containing transition probabilities between the states. $C \subseteq S$ is *closed* if for all $i \in C$ and $j \in S \setminus C$ the transition probability from i to j is 0. M is *irreducible*, if there is no proper closed subset of S. We call a distribution vector w *stationary* if w is a left eigenvector of P.

Pareto Optimality. The concept of *Pareto optimality* is usually attributed to a book of Pareto [176] from the 19th century. It is crucial to the task of *multicriteria optimization*. We will briefly touch this subject in Chapter 6 and Chapter 7 and therefore give a formal definition of Pareto optimality, as this term is used ambiguously in the literature [69]. For an introduction and an overview of approaches to multicriteria optimization, we refer to the textbook of Ehrgott [69]. We first need the notion of *componentwise order* on $\mathbb{R}^p$.

Definition 2.2. Let $x = (x_1, \ldots, x_p)$ and $y = (y_1, \ldots, y_p)$ denote two points in $\mathbb{R}^p$. We say that $x \leq y$ iff $x_i \leq y_i$ for all $1 \leq i \leq p$. If $x_i < y_i$ for all $1 \leq i \leq p$, we say that $x < y$.

In the following definition, we consider multicriteria problems where the aim is to minimize all objective functions; this definition can be modified to maximization problems in a straightforward way.

Definition 2.3. Let $\mathcal{X}$ be a set of objects and $f_1, \ldots, f_p \colon \mathcal{X} \mapsto \mathbb{R}$ a set of objective functions. We say that $x \in \mathcal{X}$ is *Pareto optimal*, if there is no $y \neq x \in \mathcal{X}$ such that $\big(f_1(y), \ldots, f_p(y)\big) \leq \big(f_1(x), \ldots, f_p(x)\big)$. If x is Pareto optimal, $\big(f_1(x), \ldots, f_p(x)\big)$ is called a *nondominated point*. Similarly, x is called *weakly Pareto optimal*, if there is no $y \neq x \in \mathcal{X}$ such that $\big(f_1(y), \ldots, f_p(y)\big) < \big(f_1(x), \ldots, f_p(x)\big)$. In this case, $\big(f_1(x), \ldots, f_p(x)\big)$ is called a *weakly nondominated point*.

In our setting, the set $\mathcal{X}$ will correspond to the set of possible clusterings of a graph, and the functions $f_1, \ldots, f_p$ to properties of the clustering as the number of intracluster edges or vertex pairs.

Chapter 3

Agglomerative Algorithms for Graph Clustering

A large part of this thesis is devoted to agglomerative graph clustering with different objective functions (cf. Chapter 4, Chapter 5 and Chapter 7). In this section, we will first discuss what distinguishes agglomerative clustering algorithms from other approaches and introduce some basic concepts. We will then proceed by identifying similarities between traditional SAHN clustering algorithms from the field of data mining as for example *single linkage* [203] and merge-based graph clustering algorithms that greedily optimize an objective function. In Chapter 4, we will exploit this connection by describing under which circumstances efficient SAHN algorithms can be applied to graph clustering in order to optimize different objective functions.

Another generic greedy clustering algorithm is based on a combination of vertex moves and the aggregation of clusters. Special instantiations of this and the merge-based greedy algorithm optimizing modularity lead to the well-known graph clustering algorithms commonly known as CNM [53] and the Louvain method [32]. In Section 3.3, we will discuss efficient implementations of these instantiations and give upper and lower bounds on their worst-case complexity. Most arguments in this section can be transferred to other objective functions. We will conclude with a comprehensive overview on agglomerative clustering approaches used in the literature, with a special focus on graph clustering algorithms.

3.1 Agglomerative and Divisive Clustering Algorithms

Methods for partitioning objects into disjoint clusters based on their similarity can be roughly divided into *agglomerative*, *divisive* and *flat* approaches. Divisive algorithms start with the all clustering and recursively split the clusters into smaller ones until some criterion as for example a desired number of clusters is met. The goal of the splitting process is to divide the objects into two groups, such that the objects in each of the groups are more similar to each other than to the objects in the other group. In the context of graph clustering, this usually translates into the identification of *sparse cuts* in the corresponding subgraphs [16, 115, 130, 160, 199]. Figure 3.1 gives a sketch of this approach. Early examples of divisive graph clustering algorithms include for example

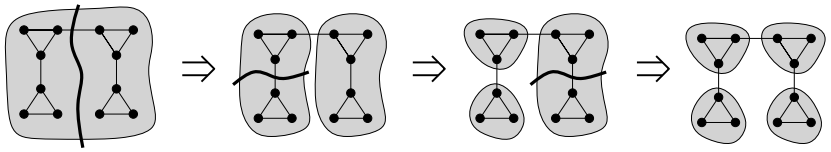
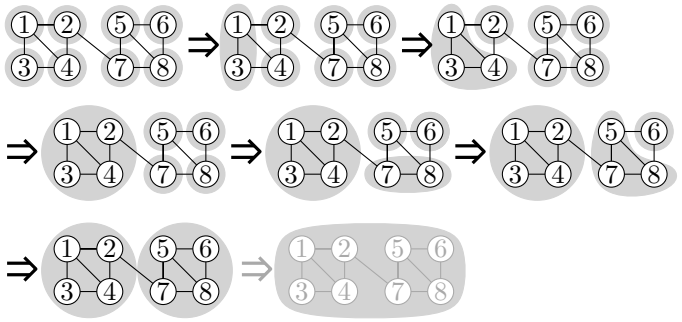
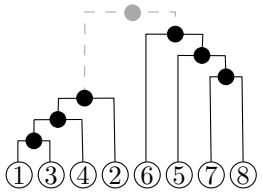


FIGURE 3.1: Sketch of a divisive clustering algorithm



(a) Sequence of merge operations



(b) Corresponding dendrogram

FIGURE 3.2: Sketch of an agglomerative clustering algorithm

the algorithm by Hartuv and Shamir [118] that iteratively splits the clusters according to a minimal cut within the cluster until each cluster fulfills some density requirement. Another example is the edge-centrality based algorithm by Newman and Girvan [173] that successively removes edges with high *betweenness* values and interprets the resulting connected components as clusters.

In contrast to that, agglomerative algorithms usually start with the singleton clustering. In the context of graph clustering, objects correspond to vertices and small subsets of densely connected clusters are successively merged to form larger clusters until some requirement is met or until all vertices belong to the same cluster. Figure 3.2 (a) shows an example of how an agglomerative graph clustering algorithm might proceed.

The third category refers to algorithms which fall in neither category but partition the objects directly into a set of clusters. Well-known methods in this category are for example a wide range of spectral algorithms [48, 115, 199, 207, 219] or Lloyd's algorithm for k-means clustering [154] in the context of clustering data in a metric space. Both of these take the number of clusters as input. Other examples are exact algorithms that solve an optimization problem in the context of clustering as the approach discussed in Section 6 that uses a sequence of linear programs to find optimal clusterings with respect to the measure surprise.

Dendrogram. There are two different points of view on agglomerative clustering. Either one is interested in all intermediate results of the clustering process or only in a single clustering that has some specific features. Examples of such features are a given number of clusters or the property to be the clustering in the hierarchy that is best with respect to some objective function. The latter point of view is often reflected by the stopping criterion. In the former case, one is usually interested in the whole *dendrogram* describing the agglomerative process. The term dendrogram refers to the storage of a sequence of merge operations in a data structure or a graphical representation thereof. Figure 3.2(b) shows a graphical representation of the merge operations in Figure 3.2(a) including the last merge that is depicted in gray. In this case, exactly two clusters are

Algorithm 1: SAHN Clustering

Input : set of objects $\mathcal{O}$, pairwise dissimilarities $d(o_i, o_j)$ for all $o_i, o_j \in \mathcal{O}$
Output: Dendrogram L as list of merge operations

```
L ← []
while |O| > 1 do
    a, b ← arg min_{{a,b}∈(O 2)} {d(a, b)}
    n ← new object representing a and b
    Append (a, b, n) to L
    O ← O \ {a, b} ∪ {n}
    forall o ∈ O \ {n} do
        Compute d(o, n)
return L
```

merged in each step, hence, the merging procedure can be depicted by a binary tree. The leaves of the dendrogram correspond to the vertices in the graph. For each merge of two clusters A and B, a new vertex is introduced representing the union $A \cup B$ and connected to the vertices representing A and B. Moreover, the height in which the vertex is drawn reflects the order of the merge operations. Dendrograms are also well defined if the agglomerative process is aborted; in this case, the dendrogram is a binary forest where each tree induces a cluster. In the example, this corresponds to the subgraph induced by the black vertices. Obviously, divisive clustering algorithms can also be represented by a dendrogram. If the splitting of clusters is stopped before the singleton clustering is reached, then the leaves of the dendrogram do not correspond to vertices in the graph but to clusters thereof.

3.2 Generic Description of Agglomerative Algorithms

Traditional agglomerative or *hierarchical* algorithms that work with pairwise distances between objects usually merge two clusters in each step and do not employ a stopping criterion, which results in a dendrogram that can be represented as a binary tree. These algorithms are often subsumed under the acronym SAHN (sequential, agglomerative, hierarchical, nonoverlapping) [204]. Algorithm 1 shows the pseudocode of a generic SAHN algorithm based on the description by Müllner [166]. The input is a set of objects together with dissimilarity or distance values associated with each pair of objects. Usually, the assumption is that distances are positive and symmetric but do not have to fulfill the triangle inequality, hence, do not necessarily define a metric. The algorithm iteratively identifies the pair of objects a and b with minimum distance and replaces a and b with a single object representing them. This new object n can be thought of as an inner vertex of the dendrogram. After that, distances between the new object and all other objects are computed. Here, the implicit assumption is always that the new distance $d(o, n)$ solely depends on some easy to maintain properties of the clusters corresponding to n and o and the old distances $d(o, a)$ and $d(o, b)$. For example, distance updates in the classical variants *single* [203], *complete*, *average*, *weighted*, *Ward* [221], *centroid* and *median* linkage discussed by Müllner [166] can be carried out by only considering the number of objects in each cluster and the old distances associated with the clusters that are to be merged.

Algorithm 2: Generic Greedy Merge

Input : graph $G = (V, E)$, function objective to maximize
Output: clustering $\mathcal{C}$ of V
$\mathcal{C} \leftarrow$ singletons;
$\mathcal{A} \leftarrow \left\{\{A, B\} \in \binom{\mathcal{C}}{2} \mid \mathsf{objective}(\mathcal{C}_{\{A,B\}}) \geq \mathsf{objective}(\mathcal{C})\right\}$;
while $|\mathcal{C}| > 1$ *and* $\mathcal{A} \neq \emptyset$ **do**
  $\{A', B'\} \leftarrow \arg\max_{\{A,B\} \in \mathcal{A}} \{\mathsf{objective}(\mathcal{C}_{\{A,B\}})\}$;
  $\mathcal{C} \leftarrow \mathcal{C}_{\{A',B'\}}$;
  $\mathcal{A} \leftarrow \left\{\{A, B\} \in \binom{\mathcal{C}}{2} \mid \mathsf{objective}(\mathcal{C}_{\{A,B\}}) \geq \mathsf{objective}(\mathcal{C})\right\}$;
return $\mathcal{C}$;

Closely related to SAHN algorithms, but with a slightly different focus, are *greedy merge* algorithms that are based on a global objective function and try to find good solutions by always greedily merging the two clusters that cause the highest gain in the objective. Pseudocode of a generic greedy merge algorithm can be found in Algorithm 2; here, $\mathcal{C}_{\{A,B\}}$ denotes the clustering that results from merging clusters A and B in clustering $\mathcal{C}$. As our focus is on graph clustering, we stated the algorithm in terms of graphs. It is however straightforward to replace vertices by objects, which makes the similarity to SAHN algorithms more apparent. This generic view based on an objective function goes back to Ward [221], who first considers this general method and then exemplifies its usage by minimizing the *total within-cluster variance*. Contrary to the way we stated generic greedy agglomeration, he assumes, in the tradition of SAHN algorithms, that the best value of the objective function is obtained by the singleton clustering. Instead of realizing the best improvement in each step, the algorithm instead selects the pair of clusters whose merge deteriorates the objective the least. This results in no evident stopping criterion, which is why the output of the algorithm is always the complete dendrogram.

The total within-cluster variance is a good example for an objective function where the greedy merge algorithm is equivalent to SAHN clustering, as the deterioration of the objective function can be interpreted as the distance between the associated clusters. If two clusters A and B are merged, the distance between $A \cup B$ and K only depends on some easy to maintain properties of A, B and K; furthermore, all remaining distances stay untouched. Hence, the algorithm fulfills the key requirements of a SAHN algorithm. On the other hand, for some SAHN algorithms it is easy to identify an objective function that is implicitly optimized. For example, single linkage [203] clustering is connected to maximizing the minimum distance between two objects in distinct clusters, whereas complete linkage [60] greedily minimizes the maximal distance between objects in the same cluster. For some objective functions, it is however not evident if the improvement in the objective function can be stated in terms of a distance function satisfying the requirements for a SAHN algorithm. In Chapter 4, we will discuss global graph clustering measures based on sparse cuts in this light, which translates into statements of how efficient the associated greedy merge algorithms can be implemented.

Another widely used local search approach is primarily based on moving vertices greedily between clusters. Early and well known examples of clustering techniques based on swapping vertices between clusters from the context of graph partitioning are the Kernighan-Lin [134] and KL/FM heuristic [77]. A generic local search procedure based

Algorithm 3: Generic Local Moving (GLM)

Input : graph $G = (V, E)$, clustering $\mathcal{C}_{\text{init}}$ of G, function objective to maximize
Output: clustering $\mathcal{C}$ of G
$\mathcal{C} \leftarrow \mathcal{C}_{\text{init}}$
repeat
  forall $v \in V$ **do**
    $N \leftarrow \arg\max_{C \in \mathcal{C} \cup \{\}} \{\text{objective}(\mathcal{C}_{v \to C})\}$
    if $\text{objective}(\mathcal{C}_{v \to N}) > \text{objective}(\mathcal{C})$ **then** move(v, N)
until *no more changes*
return $\mathcal{C}$

Algorithm 4: Generic Greedy Vertex Moving

Input : graph $G = (V, E)$, function objective to maximize
Output: clustering $\mathcal{C}_0$ of V
$G^0 \leftarrow G$, $h \leftarrow 0$
repeat
  $\mathcal{C}^h \leftarrow \text{GLM}(G^h, \text{Singletons}(G^h), \text{objective})$
  $G^{h+1} \leftarrow \text{contract}(G^h, \mathcal{C}^h)$
  $h \leftarrow h + 1$
until *no more real contractions*
while $h \geq 0$ **do**
  $h \leftarrow h - 1$
  $\mathcal{C}^h \leftarrow \text{project}(\mathcal{C}^{h+1}, G^h)$
  $\big(\mathcal{C}^h \leftarrow \text{GLM}(G^h, \mathcal{C}^h, \text{objective})\big)$; `/* optional modification */`
return $\mathcal{C}^0$

on optimizing an objective function can be formalized as in Algorithm 3. Here, $\mathcal{C}_{v \to N}$ denotes the clustering that results from moving vertex v to cluster N in clustering $\mathcal{C}$. The algorithm starts with an arbitrary clustering; if local moving is used as a self-contained clustering algorithm, this is usually the singleton clustering. One round of the algorithm consists in considering the vertices in an arbitrary order and always moving the current vertex to a cluster that yields the best improvement with respect to the objective function, if such a cluster exists. This can include the decision to *isolate* a vertex, i.e., to create a new cluster only containing this vertex. The algorithm performs several rounds, until no further vertex moves can improve the objective.

Usually, this procedure gets quickly trapped into local optima of the objective function, which leads to clusters of small size that are still far away from the optimal solution. This is why the procedure is often embedded into a *multilevel scheme* that successively contracts the resulting clusters to higher level graphs yielding a coarser view on the data. Algorithm 4 shows a generic description of such an algorithm. The advantage of this approach is that vertex moves on higher levels correspond to moving whole subsets of vertices in the original graph, which increases the local search neighborhood and is therefore less prone to local optima. Note that the benefit of move operations for the objective function on higher levels is always evaluated with respect to the benefit of moving the associated subset of vertices in the original graph. Often, this can be achieved by introducing vertex or edge weights when contracting the preliminary clusters. As an example, weighing edges with the number of edges between the associated clusters,

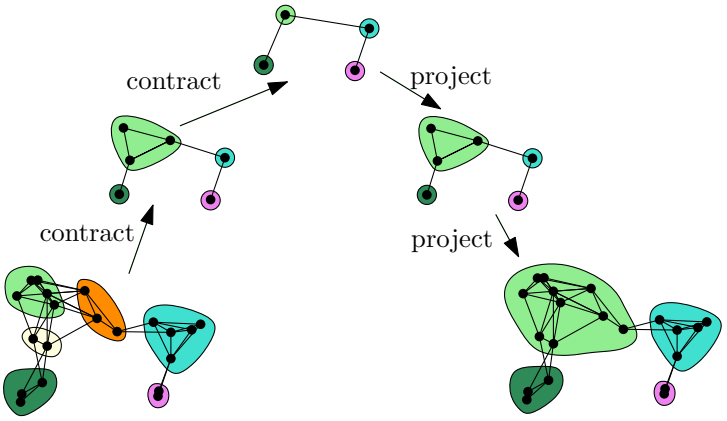


FIGURE 3.3: Sketch of Multilevel Approach

including loops signifying the number of intracluster edges in the individual clusters, and using the weighted version of modularity (cf. Section 2.3) suffices in the context of modularity optimization.

The coarsening procedure ends when a local minimum is obtained, i.e., as soon as the local moving procedure yields the singleton clustering at a certain level. In a second phase, the clustering from the highest level is iteratively *projected* to the lower levels, which means that vertices at level h are grouped together if and only if their corresponding representatives at level $h+1$ are in the same cluster. After projection, an optional step is to further refine the clustering at level h by looking for improving vertex moves. This is promising, as the clustering at level h is now coarser than before and hence not necessarily a local optimum on this level. Figure 3.3 illustrates the whole multilevel approach graphically.

Compared to GENERIC GREEDY MERGE, this algorithm is often faster in practice, as for many objectives, one round of the local moving procedure can be performed in linear time and it converges after few rounds. In the next section, we will exemplarily discuss how both algorithms can be implemented in the context of modularity maximization. Chapter 4 and Chapter 5 contain both theoretical and experimental results on these algorithms in the context of optimizing cut based measures for intercluster sparsity in the presence of constraints on the intracluster density of the resulting clustering. Similarly, in Chapter 7, we build upon GLM to derive efficient algorithms that optimize the measure surprise.

3.3 Running Time Considerations

In this section, we will exemplify how GENERIC GREEDY MERGE and GENERIC GREEDY VERTEX MOVING can be implemented in the context of modularity maximization. For simplicity, we only consider the unweighted case. Except for the upper bound on the number of iterations of GENERIC GREEDY VERTEX MOVING, all results translate to edge weighted graphs in a straightforward way. This special instantiation leads to two well-known and often used algorithms commonly referred to as CNM [53] and the Louvain method [32]. Although the latter has been already proposed in 2008 and there has been a substantial amount of work on modularity maximization since, it is still one of the state-of-the-art algorithms for this problem, especially in the case of huge complex networks

where it is known to be extremely fast while still producing high-quality clusterings. The original Louvain method does not use the (optional) possibility to further improve the clustering by the Local Moving procedure in the second phase of the algorithm. This modification has been considered by Rotta and Noack among other possibilities in a multilevel framework for modularity maximization [189].

3.3.1 Greedy Merge (CNM)

Recalling the definition of modularity from Section 2.3, the improvement in modularity that results from merging the clusters A and B in clustering $\mathcal{C}$ can be expressed as

$$\operatorname{mod}(\mathcal{C}_{\{A,B\}}) - \operatorname{mod}(\mathcal{C}) = \frac{e_{A,B}}{m} - \frac{2v_A v_B}{4m^2}$$

If we interpret this as the similarity between A and B and assume that the algorithm maintains cluster volumes and the number of edges between clusters, each similarity update after a merge operation can be performed in constant time. Furthermore, if we merge clusters A and B, we only have to compute the similarities between the resulting cluster $A \cup B$ and all other clusters, leaving the remaining values untouched. Hence, if we ignore the mainly technical difference that we are dealing with similarities instead of distances, CNM fits into the framework of SAHN algorithms[1].

For general SAHN algorithms, Day and Edelsbrunner [59] propose a generic algorithm that stores for each cluster the distances to all other clusters in a priority queue. The two clusters A and B within minimum distance (or in our case, associated with the largest improvement in the objective function) can be found by querying the priority queue of each vertex for the corresponding neighbor with minimum distance, which results in a running time of $O(n)$ if the priority queues are implemented as, for example, binary heaps. After that, the entries associated with the no longer existing clusters A and B have to be deleted from the priority queues of the other clusters, which can be bounded from above by $O(n \log n)$. In a final step, the new distances of $A \cup B$ are calculated and stored in the priority queues, again resulting in $O(n \log n)$ running time. As the number of clusters decreases by one in each step, the algorithm terminates after at most n iterations, yielding a total running time in $O(n^2 \log n)$. A downside of this approach is that it needs $O(n^2)$ memory, which might be inappropriate in scenarios where the distances could be easily computed on demand without explicitly storing them. This applies for example to Euclidean distances of points in $\mathbb{R}^d$.

In the context of graph clustering algorithms, the algorithm by Day and Edelsbrunner can often be slightly adapted such that it only needs space that is linear in the size of the graph and yields better running times in practice. This is especially beneficial if the number of edges is considerably smaller than n^2, which holds for a wide range of typical application data.

From the above expression for the improvement of modularity when merging two clusters, it is immediate that merging clusters that are not connected by at least one edge can never improve modularity. Hence, it is unnecessary to maintain values associated with unconnected clusters in the priority queues, as the algorithm terminates before they

[1] Note that the improvement in modularity when merging two clusters can be both negative and positive; although we said that, usually, distances in SAHN clustering algorithms are supposed to be positive, this does not pose a problem for any of the SAHN algorithms we consider.

become relevant. Clauset et al. [53] hence propose to only store values associated with connected clusters and furthermore, to use another priority queue to store the maximum values of each of the priority queues associated with the clusters. Furthermore, the entries of each priority queue are additionally stored in a binary search tree, which allows the algorithm to find, delete and update arbitrary values in the queue in logarithmic time. While this is clearly more memory efficient and has the potential to speed up the running time on sparse graphs, it can also be shown that this algorithm has a better worst case running time in case the dendrogram is *well balanced*, i.e., in case the height of the associated binary tree is small. Before giving a short justification of this analysis, we would like to point out that this nested heap structure can be circumvented by a simpler approach without losing efficiency. In this approach, the current cluster graph is always stored in memory, i.e., a graph where the vertices correspond to the clusters currently existing in the graph and edges exist between linked clusters; edge weights reflect the number of edges between the clusters. If, for example, the graph is stored in an adjacency list such that the neighbors of each cluster are sorted according to increasing cluster indices, the new adjacencies of the cluster $A \cup B$ can be computed in $O(d_A + d_B)$ by merging the lists of A and B. Each edge in an undirected graph is stored twice in the adjacency list; if each entry stores a pointer to the other representative of the edge, invalid entries in the other lists can be efficiently deleted. To find the maximum value associated with each edge, the whole set of edges is additionally stored in a priority queue. Hence, the bottleneck of the algorithm is the update of the binary heap after each merge; as we have to update $O(d_A + d_B)$ values, each iteration takes $O\big((d_A + d_B) \cdot \log n\big)$ time.

Clauset et al. [53] give an alternative upper bound on the total number of priority queue operations as follows. Let us denote the set of cluster pairs that are merged during the execution of the algorithm by $\mathcal{M}$. Each of the clusters considered in the merging process corresponds to a vertex D in the associated dendrogram $\mathcal{D}$. As each of these vertices participates at most once in a merge, the total number of priority queue operations can be bounded from above by

$$\sum_{\{A,B\} \in \mathcal{M}} [d_A + d_B] \leq \sum_{D \in \mathcal{D}} d_D \leq \sum_{D \in \mathcal{D}} v_D$$

For simplicity, let us assume that the merging process is complete, i.e., we can consider $\mathcal{D}$ as a binary tree rooted at the vertex that corresponds to the all clustering.[2] We can now sort the vertices of $\mathcal{D}$ into levels corresponding to a breadth first search from the root. The clusters in each of the levels are pairwise non-overlapping, hence, the total volume of the clusters on each level is exactly $2m$. Thus, if d is the height of the dendrogram interpreted as a rooted tree, the total number of priority queue operations can be bounded above by $O(m \cdot d)$, resulting in a total running time of $O(m \cdot d \log n)$. In the best case, this is in $\Theta\big(n(\log n)^2\big)$, but if the dendrogram is unbalanced, the resulting upper bound is worse than the trivial $O(n^2 \log n)$ bound. Wakita and Tsurumi [220] observed that the CNM algorithm sometimes produces very unbalanced dendrograms which makes the algorithm slow in practice. They propose to skew the values in the priority queue in favor of pairs of clusters of similar size, which leads to better efficiency without sacrificing quality. Similar studies have been conducted by Danon et al. [58] and Rotta and Noack [189]. Rotta and Noack additionally consider a modification of the generic greedy merge algorithm called *multistep joining* proposed by Schuetz and

[2] If this is not the case, we can just virtually complete the dendrogram in an arbitrary fashion and then apply the following analysis.

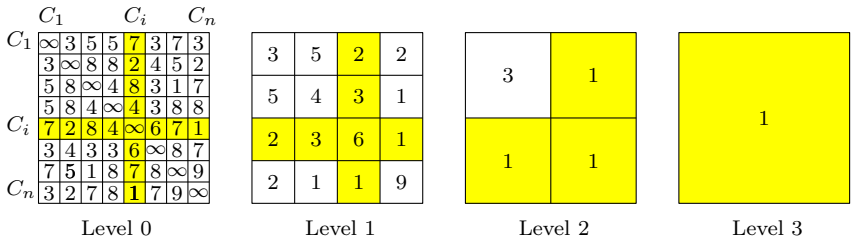


FIGURE 3.4: Sketch of SAHN implementation of Eppstein [70] based on a quadtree. The yellow entries correspond to distances that have to be recomputed when C_i is changed.

Caflisch [193] that computes a small set of disjoint merge operations with high benefit with respect to modularity and merges them concurrently in each step. Similar to the modification of the values in the priority queue, this causes the algorithm to grow clusters in different parts of the graph concurrently, which has a positive influence on the height of the dendrogram.

Another possibility to implement general SAHN algorithms is proposed by Eppstein [70]. The crucial ingredient in this approach is a quadtree that replaces the priority queue as a means of efficiently maintaining the pair of clusters with minimum distance, see Figure 3.4. Level 0 of the tree always maintains the current distance matrix between the clusters (or objects, in terms of Algorithm 1). Level $i+1$ contains one entry for each square of size 4 in level i that equals the minimum of the associated distances in level i. Thus, level $\log_2(\lceil n \rceil)$ contains one entry, which equals the minimum distance between any pair of clusters. If the distances involving one cluster are changed, this affects exactly one row and column on each of the levels; in the example in Figure 3.4, these are marked yellow. It can be easily seen that the total number of these values is in $O(n)$. Hence, if we assume that the new distances can be computed in constant time, all updates can be performed in time linear in the number of objects n. Merging two clusters A and B can now be realized by deleting the rows and columns corresponding to A and B and adding a new row and column to store the distances to the cluster resulting from the merge. Hence, the total number of updates needed is linear, which leads to a total running time in $O(n^2)$ for n merge operations.

As mentioned above, CNM can be interpreted as a SAHN algorithm and hence, the idea of Eppstein [70] can be used to implement it with $O(n^2)$ time and space complexity, compared to $O(n^2 \log n)$ time in the worst case and linear space for the heap based approach. To the best of our knowledge, alternative SAHN algorithms for modularity based clustering have not yet been considered. However, as there are faster and more accurate algorithms for modularity maximization available by now, this is probably of little practical interest, at least in the standard scenario where the goal is to obtain one clustering of a static graph with high quality with respect to modularity.

3.3.2 Greedy Vertex Moving (Louvain)

To assess the running time of the Louvain method, i.e., GENERIC GREEDY VERTEX MOVING with modularity as objective, we first consider the complexity of one round of the GENERIC LOCAL MOVING algorithm. Similar to the gain when joining two clusters, the improvement in modularity when moving a vertex v from its current cluster A to

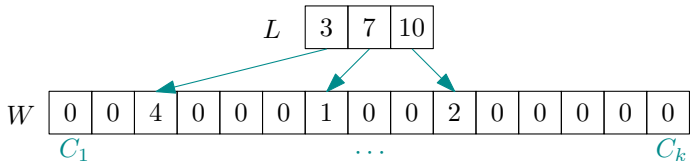


FIGURE 3.5: Bucket sort like method to collect the number of edges connecting v to the clusters $\{C_1, \ldots, C_k\}$

cluster B can be easily calculated as

$$\mathrm{mod}(\mathcal{C}_{v\to B}) - \mathrm{mod}(\mathcal{C}) = \frac{e_{B,\{v\}} - e_{A\setminus\{v\},\{v\}}}{m} + \frac{[v_{A\setminus\{v\}} - v_B] \cdot \deg(v)}{2m^2}$$

If A is considerably larger than B, it is possible that moving v to B increases modularity even if v is not linked with any vertex in B. It can however easily be seen that in this case isolating v has an even greater benefit with respect to modularity. As above, when computing the best cluster for a vertex, it is sufficient to consider all clusters it is linked with and, additionally, the possibility to isolate it.

In contrast to the implementation of the CNM algorithm, we do not contract clusters during the local moving procedure but always work on the original graph. For each vertex, we store the id $\mathrm{id}(v)$ of the cluster it currently belongs to. These values are initialized such that $\mathrm{id}(v_i) = i$ in the beginning. In the rare case that a vertex becomes isolated, we assign to it the id of a cluster that is currently empty, thus guaranteeing that all ids remain in $[1, n]$. Furthermore, we have to maintain the volume of each cluster during the execution of the algorithm.

To decide where to move a vertex v, we have to determine the number of edges linking it to every neighboring cluster. This can be done efficiently in time $O(\deg(v))$ by using a simple trick similar to bucket sort. At the beginning of the algorithm we initialize a vector W of size n with zeros at each position. For each neighbor of v, we increment $W[i]$ by one. If $W[i]$ was zero before, we additionally store i in a list L of neighboring clusters (see Figure 3.5). Now, the best neighboring cluster can be determined by going through L and considering the corresponding entry in W. In the end, we have to reset all values of neighbors to zero in W in order to be able to reuse W for the next vertex. Altogether, in one round of the algorithm, we iterate over the neighbors of all vertices, thereby considering each edge twice; this leads to a total running time in $O(m + n)$. This approach has for example been taken in the very fast implementation by Blondel et al. [32].

After local moving has converged on some level, we contract the resulting clusters to obtain a higher level graph where edge weights represent the number of edges linking the clusters that are represented by the incident vertices. For that, we can first construct for each non-empty cluster C in the graph a list containing its vertices by using for example bucket sort. The number of edges linking C with every other cluster and therefore all of its adjacencies can now be determined by iterating over all edges incident to clusters in v and using the above trick. Hence, contraction is also easily possible in linear time.

An asymptotic upper bound on the total running time of the algorithm is thus given by $O(r \cdot (n+m))$, where r is the total number of rounds spent by the local moving procedure. To the best of our knowledge, the best upper bound known for r for unweighted graphs is the trivial bound derived by the observation that the objective gain associated with any

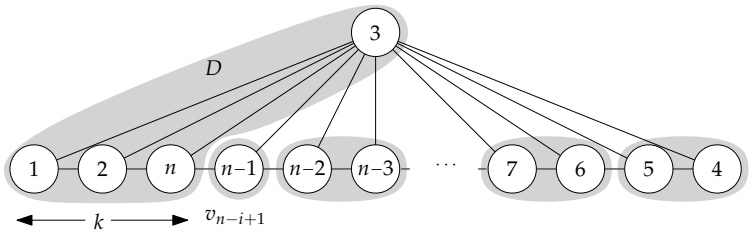


FIGURE 3.6: Graph family where the Louvain method needs $\Omega(n)$ rounds, if vertices are considered in the order indicated by the indices. The clustering shows the intermediate clustering obtained after one round of local moving, starting from the singleton clustering. The number of vertices n in the graph is assumed to be odd.

performed move equals $\text{mod}(\mathcal{C}_{v\to B}) - \text{mod}(\mathcal{C}) = a/(2m^2)$ for some $a \in \mathbb{Z}^+$. Hence, as the modularity of any clustering is in $[-1/2, 1]$ (cf. [35]) and modularity improves by at least $1/(2m^2)$ in each step, the algorithm performs at most $O(m^2)$ moves in total, which results in a total of $O(m^2)$ rounds in the worst case. In degenerate cases, $\Omega(n)$ rounds can be necessary, if we assume a worst case order in which the vertices are considered. An example for this is the graph family illustrated in Figure 3.6.

Proposition 3.1. *For the graph family depicted in Figure 3.6, the number of rounds performed by the Louvain algorithm on the lowest level of the hierarchy is in* $\Omega(n)$.

Proof. We first recall that the Louvain algorithm never moves vertices to clusters they are not linked with; thus, we do not have to consider this possibility when determining the best move for each vertex. We will show that, as long as the cluster of vertex 3 is small enough, exactly one vertex joins this cluster in each round of the algorithm. We have $m = 2n - 3$ and hence, the improvement of moving a vertex v from cluster A to cluster B can be expressed as

$$\frac{\overbrace{e_{B,\{v\}} - e_{A\setminus\{v\},\{v\}}}^{=:\alpha}}{2n-3} + \frac{\overbrace{[v_{A\setminus\{v\}} - v_B]\cdot \deg(v)}^{=:\beta}}{8n^2 - 24n + 18}$$

If $\alpha > 0$, β is constant and n is large enough, this value is larger than 0, which means that the associated move always improves modularity.

In the first step, we justify why the clustering after the first round of local moving is the one illustrated in Figure 3.6. Vertex 1 has two neighboring clusters $\{2\}$ and $\{3\}$, both of which share one edge with vertex 1. As the volume of $\{2\}$ is smaller than the volume of $\{3\}$, moving vertex 1 to $\{2\}$ is always more beneficial than moving it to $\{3\}$. Furthermore, as both volumes are constant, this move improves modularity for large n. Hence, vertex 1 is moved to $\{2\}$. With similar arguments, it can be seen that vertex 2 does not change its cluster membership. If n is large enough, moving vertex 3 to $\{1, 2\}$ yields an improvement of $\frac{2}{2n-3} - \frac{5n-5}{8n^2-24n+18}$, which is both larger than zero and the improvement $\frac{1}{2n-3} - \frac{2n-2}{8n^2-24n+18}$ of moving it to $\{4\}$. Moving vertex 3 to any other neighboring cluster is even worse, hence, it is moved to $\{1, 2\}$. With similar arguments as we applied for vertex 1 and vertex 2, it can be seen that vertex 4 moves to $\{5\}$ and vertex 5 does not change cluster membership. The same applies to the vertices 6 to $n - 1$. Vertex n is moved from $\{n - 1, n\}$ to cluster $\{1, 2, 3\}$ if n is large enough, as the associated improvement $\frac{1}{2n-3} - \frac{3n+3}{8n^2-24n+18}$ is larger than 0. This yields the claimed clustering after the first round.

We now prove by induction that for each $i \in \{2, \dots, \lfloor \frac{1}{10} n \rfloor\}$, exactly one vertex move occurs in round i, the move of vertex $n - i + 1$ to the cluster D containing vertex 3. Hence, the Louvain algorithm performs at least $\lfloor \frac{1}{10} n \rfloor$ rounds on the lowest level of the hierarchy and the statement of the proposition follows.

Let $i = 2$. We already know that the initial clustering looks like the one depicted in Figure 3.6. Analogous to the arguments used in the first round, it can be verified that the vertices $1, \dots, n-2$ do not change their cluster membership, vertex $n-1$ moves to D, and vertex n remains in D.

For $i > 2$, it can be easily seen that vertex 1 and vertex 2 do not get isolated from D, as otherwise the rightmost vertex on the lower path that is contained in D would not have been moved to D in the previous round.

For vertex 3, we claim that if we would first isolate it, moving it back to cluster D is always at least as good as moving it to another cluster or leaving it isolated, which is equivalent to the statement that it remains in D. Let k be the number of vertices in D on the path. The improvement in modularity when moving vertex 3 back to D can be expressed as

$$\frac{k}{2n-3} - \frac{(n-1)\cdot(3k-1)}{8n^2-24n+18} = k \cdot \underbrace{\left(\frac{1}{2n-3} - \frac{3n-3}{8n^2-24n+18}\right)}_{>0 \text{ for large } n} + \underbrace{\frac{n-1}{8n^2-24n+18}}_{>0 \text{ for large } n}$$

Hence, moving vertex 3 to D always improves modularity and, as the improvement increases with k and no cluster on the path contains more vertices than $D \setminus \{3\}$, the claim follows.

It is not hard to see that vertex 4 to vertex $n - i$ remain in their cluster, as essentially nothing has changed for them since the previous round. For vertex $n-i+1$ we have to distinguish two cases, either it is contained in a cluster of size 2 or currently isolated.

In the first case, it does not become isolated, otherwise it would have been already isolated in the previous round. Hence, it either stays in its cluster or is moved to cluster D. The improvement of the latter with respect to modularity can be expressed as

$$\frac{1}{2n-3} + \frac{3\cdot[3-(n-1+3k-1)]}{8n^2-24n+18} = \frac{1}{2n-3} - \frac{3n+9k-15}{8n^2-24n+18}$$

It is $k = i + 1 \leq \frac{1}{10} n + 1$ and hence

$$\frac{1}{2n-3} - \frac{3n+9k-15}{8n^2-24n+18} \geq \underbrace{\frac{1}{2n-3} - \frac{3.9n-6}{8n^2-24n+18}}_{>0 \text{ for large } n}$$

It follows that vertex $n - i + 1$ is moved to cluster D.

If vertex $n-i+1$ is currently isolated, moving it to the cluster of vertex $n-i$ improves modularity if n is large enough, as α is larger than 0 and β is constant. It remains to show that modularity increases even further if vertex $n-i+1$ is moved from the cluster of vertex $n-i$ to D; this is completely analogous to the first case.

It is not hard to verify that the vertices $n-i+2$ to n do not get isolated, as otherwise moving vertex $n-i+1$ to D would have decreased modularity. In summary, the only

vertex move that occurred was the move of vertex $n - i + 1$ to D, which concludes the induction step and shows the proposition. □

In summary, one round of the Louvain algorithm is linear in the size of the current level of the hierarchy, which is at most the size of the input graph. For unweighted graphs, we further gave a lower bound of $\Omega(n)$ and a trivial upper bound of $O(m^2)$ on the total number of rounds that are executed. There is still a very large gap between these bounds and other scenarios might be interesting as well. Up to now, we assumed a fixed vertex ordering; a simple modification of Louvain that has also been considered by its authors is to visit the vertices in a random order in each iteration, which raises the question of average time complexity over all possible orderings. Furthermore, the upper bound on the number of rounds only holds for unweighted graphs, whereas we do not know of any subexponential bound if edges are weighted.

From the theoretical point of view, this leaves some interesting open problems. From the practical point of view, the lower bound on the number of rounds needed in the worst case is very far away from the actual number of rounds observed on real world instances. For example, on the graph `uk-2002`[3], which is a webgraph with over 18 million vertices and 216 million edges, the algorithm converges after only 59 rounds. This makes the algorithm very fast and therefore popular in practice; as a heuristic to further decrease the running time, it is possible to set a limit on the number of rounds without sacrificing too much quality [32].

3.4 Related Work

The application of hierarchical clustering algorithms dates back to the 1950's, when Sneath proposed to apply single linkage clustering in the context of identifying taxonomic groups of bacteria [203]. Single linkage is a simple SAHN algorithm where in each step, the minimum distance between two objects in distinct clusters is computed and the corresponding clusters are linked. Since then, a lot of work has been invested in developing more efficient algorithms for general SAHN clustering or designed for specific distance update schemes [11, 26, 45, 59, 60, 70, 105, 221, 127, 133, 139, 143, 166, 167, 169, 185, 201]. We will describe the most general of these algorithms in detail in Section 4.4.3 and discuss their applicability in the context of graph clustering based on intracluster density and intercluster sparsity measures.

For the special case of single linkage, Ross and Gower [105] observe that it is possible to first construct a minimum spanning tree of the objects and then to iteratively merge clusters that are linked by the edges of the spanning tree ordered by increasing weight. Rohlf [185] proposes a modification of this approach that does not compute the full spanning tree but enough information to create the correct dendrogram in $O(n^2)$ time and in linear space. In Müllner's experiments, this variant turns out to be very fast in practice. Another algorithm with the same guarantees for single linkage is the SLINK algorithm by Sibson [201]. A similar algorithm exists for complete linkage [60]. A systematic overview on early work on SAHN algorithms can be found in the books of Jain and Dubes [125] and Anderberg [11], the related work discussed by Day and Edelsbrunner [59] and the survey by Murthag [168].

[3] `http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml`

It is worth to remark that in the context of vector quantization, a special SAHN linkage scheme minimizing the average distance of vertices to the centroid of their cluster is known as *pairwise nearest neighbor clustering* (PNN) [71, 197]. Fränti et al. consider fast algorithms to implement this linkage scheme exactly [85] and approximately [86]. The former is more or less equivalent to Anderberg's general SAHN algorithm based on a list of nearest neighbors for each cluster [11]. Another approximate PNN algorithm is described by Equitz [71].

A well-known clustering algorithm that includes a SAHN algorithm as one step of the clustering procedure is BIRCH [234]. CURE [110] is another clustering algorithm for points in k-dimensional space that can be seen as a compromise between centroid and single linkage. It is based on a variant of Anderberg's algorithm that uses a *k-d tree* for nearest neighbor queries whenever these are necessary. ROCK [111] can be seen as a graph clustering algorithm, although it aims to cluster attributed data. In a preprocessing phase, a *threshold graph* is built where edges link objects that are "sufficiently similar". Then, the similarity of two objects is defined as their number of common neighbors. Roughly speaking, the aim of the main clustering procedure is now to maximize the overall similarity of objects in the same cluster. The objects are then clustered using a similarity measure based on this global objective with the help of a SAHN algorithm using a hierarchy of binary heaps. Similarly, CHAMELEON [131] is a graph based clustering algorithm that applies hierarchical clustering in the second phase to obtain the final clustering. In the first phase, a *k nearest neighbor graph* is built for the objects, i.e., edges between each object and its k closest neighbors are created. Then, a graph partitioning algorithm is used to partition the objects into many small preclusters. In the second phase, a SAHN algorithm that uses a custom distance function taking both the graph structure and the distances between the original objects into account is applied to cluster the preclusters. The globally closest pair is here maintained in a binary heap. An alternative second phase consists in iteratively merging all pairs of clusters whose distance is below a certain threshold.

In the context of graph clustering, Fortunato [83] gives an overview on similarity measures between the vertices of a graph. Commonly used measures include similarities based on the number of common neighbors [43, 222] or the *commute time* between the vertices in a random walk [228] and related concepts. He points out, that in principle any similarity measure for vertices could be used with any non-metric linkage scheme as an algorithm for graph clustering. This approach has for example be taken by Harel and Koren, who use distance measures based on random walks [116]. Carrasco et al. [46] use a modified version of single linkage to cluster vertices of a bipartite advertiser-keyword graph based on the similarity of their neighborhoods. Hopcroft et al. [121] combine similarities of vertices in citation networks derived from the set of commonly cited articles with an agglomerative algorithm similar to centroid linkage. Donetti and Munoz [67] first project the vertices in k-dimensional space based on the eigenvectors of the Laplacian. In the second step, they use single and complete linkage to cluster the resulting points, but restrict merging operations to clusters that are connected in the original graph. The final clustering is then chosen as the clustering with maximum modularity in the dendrogram. A well-known hierarchical graph clustering algorithm is the CNM algorithm by Clauset et al. [53] that greedily optimizes modularity. We have described this algorithm in detail in Section 3.3.1. Several modifications of this algorithm exist, either targeted at decreasing the running time or improving the quality of the resulting clustering [68, 189, 208, 220]. Görke et al. [95] consider null model based graph clustering objectives inspired by modularity that are based on a different tradeoff between the

actual and expected number of intracluster edges in a graph. In contrast to modularity, there is no obvious way to fit the Generic Greedy Merge algorithm greedily optimizing these measures into the framework of SAHN algorithms. Using a data structure that maintains the convex hull of a fully dynamic point set P and that allows for quick tangent queries, they show that it is nonetheless possible to achieve a worst case running time of $O(n^2 \log n)$ for the merging procedure.

Local search algorithms for clustering and partitioning problems based on vertex moves abound. Many of them apply a multilevel scheme in which the graph is first coarsened by the iterative contraction of small subclusters and then optionally partitioned by a potentially expensive algorithm on the resulting smaller graph. In the following refinement phase, the contractions of the coarsening phase are iteratively undone and the clustering is further improved on each level using different heuristics. In this scheme, vertex moves are most often used in the refinement phase, as they allow for fine grained modifications of the clusters and are usually very cheap. Early refinement techniques based on vertex moves from the field of graph partitioning include the Kernighan-Lin [134] and KL/FM heuristic [77]. For more information on the use of multilevel algorithms for graph partitioning, we refer to the recent overview article of Buluç et al. [42].

Clustering algorithms also often rely on iteratively improving the current clustering by moving single objects between the clusters. A very well-known algorithm for k-means clustering is for example Lloyd's two phase algorithm [154] that greedily minimizes the within-cluster sum of squares of a clustering. One of these phases can be seen as an instantiation of Generic Local Moving, with the difference that the input is a set of points instead of a graph.

In the context of modularity-based graph clustering, refining a clustering based on greedy vertex moves has been first proposed by Schuetz and Caflisch [193]. In Section 3.3.2, we already discussed the Louvain algorithm [32], which uses greedy vertex moves for the coarsening phase in a multilevel scheme and therefore fits the description of Generic Greedy Vertex Moving. Inspired by algorithms for graph partitioning, Rotta and Noack [189] evaluate different possibilities for the coarsening and refinement phase in a general multilevel scheme optimizing modularity. The different variants considered for the coarsening phase depend on greedily moving or merging clusters based on *join prioritizes*, which are variants of the modularity increase that basically skew the merge process in favor of balanced dendrograms, similar to the idea of Wakita and Tsurumi [220]. They further consider a vertex moving procedure that is based on always performing the *globally best* vertex move instead of considering all vertices in rounds; in their experiments, this variation turns out to be far slower while yielding worse quality. In terms of local search heuristics [120], Generic Local Moving can be seen as a hill-climbing algorithm in which the neighborhood of a clustering is defined by vertex moves. Among other local search metaheuristics applied to modularity based clustering is *simulated annealing* [112, 161], which leads to good quality but higher running times.

Karypis et al. [132] first use a variation of *group average* linkage to obtain an initial clustering. In a following multilevel scheme, this clustering is further improved by first coarsening the graph contracting matchings subject to the constraint that each contracted edge links vertices in the same cluster. In the refinement phase, greedy vertex moving with different objective functions is used to improve the clustering on all levels of the resulting hierarchy. Similarly, Conan-Guez and Rossi [55] use a coarsening phase based on merging clusters such that an expansion based objective function is greedily

optimized. In the refinement phase, they further improve this objective functions by greedy vertex moves on a selection of the levels. Dhillon et al. [63] aim at optimizing different cut based objective functions for graph clustering in their tool Graclus. Similar to many algorithms for graph partitioning, they use a *base clustering*[4] phase between coarsening and refinement. The idea is to stop the coarsening procedure at some point to perform a more expensive algorithm on the significantly smaller graph after the contractions. Graclus uses spectral clustering for this step. The coarsening phase contracts matchings with high weight, where weight corresponds to the improvement in the objective function used. Weighted kernel k-means is used for refinement on all levels.

[4] In terms of graph partitioning, this corresponds to the initial partitioning phase.

Chapter 4

Density-Constrained Graph Clustering

The guiding intuition in the field of graph clustering is "intracluster density vs. intercluster sparsity". Mathematical formalizations thereof abound, most of which, however, incorporate both aspects into a single criterion, which then serves as a quality measure for graph clusterings. However, balance between the two aspects is a fine line. Treating them separately offers possibilities to individually adapt the tradeoff between them: High intracluster density typically corresponds to finer and low intercluster density to coarser clusterings. While the recent literature on graph clustering has mainly been focusing on large data sets and on single criteria such as Modularity [173], Kannan et al. [129] propose to minimize the cut between clusters, subject to a guaranteed conductance within them and show that this approach avoids the drawbacks of many simpler measures. This stepping stone in bicriterial graph clustering inspired Flake et al. [79], who gave an algorithm with provable, but interdependent bounds on both intra- and a variant of intercluster expansion. Brandes et al. [38] were the first to use a notion of intercluster conductance to experimentally evaluate clustering algorithms.

A range of well-known measures for quantifying the sparsity of a cut exist, many of which are indisputable. Among these are conductance [129], expansion and density, all of which suggest the adaption to measuring clustering with respect to both aspects. However, not all of them have so far been coined and used. We systematically collect such measures and set our main focus on scrutinizing their behavior in the light of the question which combinations of intracluster density and intercluster sparsity measure enable efficient greedy agglomeration (cf. Algorithm 2 in Chapter 3), putting aside other algorithmic approaches [83].

Outline. We start by defining different intercluster sparsity and intracluster density measures; the former are based on the sparsity of the associated cut and the latter on the density of the induced subgraphs. In Section 4.2, we formally define the generic problem Density-Constrained Clustering, which aims at optimizing intercluster sparsity while retaining guarantees on the intracluster density of the resulting clustering, followed by a discussion of related approaches from the literature. Furthermore, we discuss traits of intracluster density and intercluster sparsity measures that aim at capturing our intuition that the former strives towards fine and the latter towards coarse clusterings, and prove or disprove these traits for each established measure. This yields

qualitative insights into their behavior with respect to greedy agglomeration. We proceed by reviewing efficient SAHN algorithms from the literature and identify for each of these algorithms key properties of objective functions and constraints that are sufficient in order to apply them to implement greedy agglomeration. Section 4.4.4 and Section 4.4.5 then contain an almost complete classification of the proposed measures with respect to these properties.

Systematic experiments evaluating how well these measures conform to human intuition and how well the proposed algorithms discover existing clusterings are beyond the scope of this chapter. To some extent, these questions are covered in the experiments in Chapter 5.

4.1 Quality Measures for Clusterings

The bicriteria measures we construct and use in this work build upon the conductance, expansion and density of cuts in the graph. Each of these formalizes a varied view of the paradigm of graph clustering and can be cast into either measures for intracluster density or for intercluster sparsity. The *conductance* of a cut (S, T) measures the bottleneck between S and T, defined as

$$\mathsf{conductance}(S,T) = \frac{e_{S,T}}{\min\{v_S, v_T\}}$$

Expansion substitutes volume by cardinality:

$$\mathsf{expansion}(S,T) = \frac{e_{S,T}}{\min\{n_S, n_T\}}$$

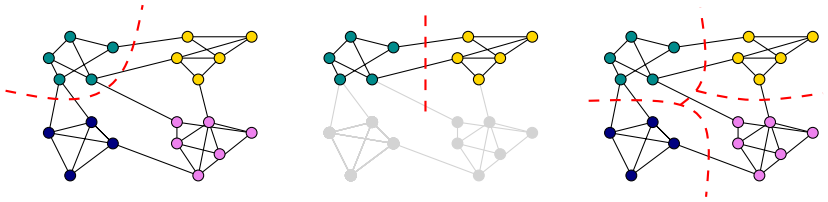
The *density* of a cut is defined as

$$\mathsf{density}(S,T) = \frac{e_{S,T}}{n_S n_T}$$

In the following, we describe how these criteria translate into concrete measures that can be used to assess the quality of a whole clustering.

4.1.1 Intercluster measures

For intercluster measures, we distinguish two ways of measuring cuts: between pairs of clusters (*pairwise*), or cutting off a cluster (*isolating*). Isolated measures assess how well a cluster is separated from the remainder of the graph and pairwise measures how well the clusters are separated from each other. For an illustration, see Figure 4.1 (a) and (b). The isolated view yields a sparsity value for each of the clusters, defined by the cut induced by the cluster. In contrast to that, the pairwise view considers cuts between pairs of clusters in the subgraph induced by their union; in total, this yields $\binom{|\mathcal{C}|}{2}$ values for the clustering $\mathcal{C}$. To have these quantities express the quality of an entire clustering, we can either construct a worst-case measure by considering the *minimum* or *maximum* of the cut values of the whole clustering or an *average* measure, which builds upon the average of the cut values. In addition to that, density lends itself to the natural idea of adding up all values before normalization. In terms of graph cuts, we can think of this measure as the density of the k-way cut induced by the clustering.



(a) isolated view (b) pairwise view (c) global view

FIGURE 4.1: Illustration for different ways to use cut-based measures to evaluate the intercluster sparsity of a clustering.

TABLE 4.1: Density and counting

intracluster density		
global	gid	$\frac{\sum_{C\in\mathcal{C}} e_C}{\sum_{C\in\mathcal{C}} \binom{n_C}{2}}$
minimum	mid	$\min_{C\in\mathcal{C}} \frac{e_C}{\binom{n_C}{2}}$
average	aid	$\frac{1}{\lvert\mathcal{C}\rvert} \sum_{C\in\mathcal{C}} \frac{e_C}{\binom{n_C}{2}}$

intercluster edges		
global	nxe	$\sum_{\{A,B\}\in\binom{\mathcal{C}}{2}} e_{A,B}$

intercluster sparsity		
global	gxd	$\frac{\text{nxe}}{\sum_{A\neq B\in\mathcal{C}} n_A n_B}$
max. pw.	mpxd	$\max_{A\neq B\in\mathcal{C}} \frac{e_{A,B}}{n_A n_B}$
max. is.	mixd	$\max_{C\in\mathcal{C}} \frac{x_C}{n_C n_{V\setminus C}}$
av. pw.	apxd	$\binom{\lvert\mathcal{C}\rvert}{2}^{-1} \sum_{\{A,B\}\in\binom{\mathcal{C}}{2}} \frac{e_{A,B}}{n_A n_B}$
av. is.	aixd	$\frac{1}{\lvert\mathcal{C}\rvert} \sum_{C\in\mathcal{C}} \frac{x_C}{n_C n_{V\setminus C}}$

We call this the *global* point of view (see Figure 4.1 (c)). A simple global alternative to global intercluster density is to count the total number of intercluster edges.

Thus, in accordance with the above discussion, we define the twelve intercluster sparsity measures resulting from the isolated and pairwise view listed in Tables 4.1 and 4.2, plus the two measures with a global nature: gxd and nxe (cf. Table 4.1). Given a clustering, all of these measures can be efficiently evaluated. Adhering to our abbreviations, we denote individual clusters' contributions by

$$\text{ixd}(C) := \frac{x_C}{n_C n_{V\setminus C}}, \text{ixc}(C) := \frac{x_C}{\min\{v_C, v_{V\setminus C}\}}, \text{ and } \text{ixe}(C) := \frac{x_C}{\min\{n_C, n_{V\setminus C}\}}$$

Analogously,

$$\text{pxd}(\{A,B\}) := \frac{e_{A,B}}{n_A n_B}, \text{pxc}(\{A,B\}) := \frac{e_{A,B}}{\min\{v_A, v_B\}}, \text{ and } \text{pxe}(\{A,B\}) := \frac{e_{A,B}}{\min\{n_A, n_B\}}$$

In the special case that $\mathcal{C}$ is the all clustering, i.e., $\mathcal{C}$ only contains one cluster, we define all these values to be 0, which corresponds to the optimum. Any other choice is counterintuitive on trivial examples such as a clique.

TABLE 4.2: Intercluster measures based on conductance and expansion

intercluster conductance		
maximum pairwise	mpxc	$\max_{A \neq B \in \mathcal{C}} \frac{e_{A,B}}{\min\{v_A, v_B\}}$
maximum isolated	mixc	$\max_{C \in \mathcal{C}} \frac{e_{C,V \setminus C}}{\min\{v_C, v_{V \setminus C}\}}$
average pairwise	apxc	$\frac{1}{\binom{\lvert\mathcal{C}\rvert}{2}} \sum_{\{A,B\} \in \binom{\mathcal{C}}{2}} \frac{e_{A,B}}{\min\{v_A, v_B\}}$
average isolated	aixc	$\frac{1}{\lvert\mathcal{C}\rvert} \sum_{C \in \mathcal{C}} \frac{e_{C,V \setminus C}}{\min\{v_C, v_{V \setminus C}\}}$
intercluster expansion		
maximum pairwise	mpxe	$\max_{A \neq B \in \mathcal{C}} \frac{e_{A,B}}{\min\{n_A, n_B\}}$
maximum isolated	mixe	$\max_{C \in \mathcal{C}} \frac{e_{C,V \setminus C}}{\min\{n_C, n_{V \setminus C}\}}$
average pairwise	apxe	$\frac{1}{\binom{\lvert\mathcal{C}\rvert}{2}} \sum_{\{A,B\} \in \binom{\mathcal{C}}{2}} \frac{e_{A,B}}{\min\{n_A, n_B\}}$
average isolated	aixe	$\frac{1}{\lvert\mathcal{C}\rvert} \sum_{C \in \mathcal{C}} \frac{e_{C,V \setminus C}}{\min\{n_C, n_{V \setminus C}\}}$

4.1.2 Intracluster measures

Intracluster density measures quantify how well the vertices within a cluster are interconnected. A natural way to extend the notion of sparse cuts to the (inner) density of the individual clusters is to compute the conductance, expansion or density of the subgraph induced by the cluster. In this context, the conductance, expansion or density of a graph is defined as the minimum value over all possible cuts of the graph. However, the very evaluation of both the conductance and the expansion of a graph is $\mathcal{NP}$-hard [148, 202]; the same holds for the *sparsest cut problem* as formalized in [163], which generalizes the problem of finding a cut of minimum density to graphs with edge weights. While there are many ways to deal with this, it generally discourages the use of these functions as intracluster measures.[1]

Density can also be used in another way by defining the density of a graph as the number of edges divided by the maximum possible number of edges, which equals the number of vertex pairs in the graph. We refer to the density of a cluster in the sense of the density of the subgraph induced by it as

$$\mathsf{id}(C) = \frac{2e_C}{n_C(n_C - 1)}$$

As in the case of isolated measures for intercluster sparsity, this yields as many density values as clusters; taking the minimum of these values leads to *minimum intracluster density*, whereas the average corresponds to *average intracluster density*. Similar to intercluster density, we can also take a more global view and regard the density of the whole clustering as the number of intracluster edges divided by the total number of vertex pairs that are contained in the same cluster. This yields *global intracluster density*. In the degenerate case that a cluster C only contains one vertex, we define its intracluster density to be maximal, i.e., $\mathsf{id}(C) := 1$.

[1] Note that a bottom-up approach cannot take advantage of the approximation results used in [129].

4.1.3 Qualitative Observations

While all proposed intra- and intercluster measures are based on the same intuition, there are fundamental differences in the way they assess particular clusterings. One important point is whether balanced clusterings, i.e., homogeneous cluster sizes, are rewarded or penalized. As an example aid, as an intracluster measure, has a tendency to favor unbalanced clusterings, as singletons yield optimum values and it is easy to compensate the existence of a large cluster with poor intracluster density with an appropriate number of singletons. In contrast to that, mid rewards balanced clusterings, as clusters that are larger than the average are more likely to have low intracluster density and thus to be the qualitative bottleneck. Gid ranges somewhere between these extremes. Using the number of intercluster edges to measure intercluster quality clearly favors unbalanced clusterings, as cutting off small portions of the vertex set from the remainder of the graph usually cuts far fewer edges than partitioning the graphs in two blocks of roughly equal size. To some extent this effect can be compensated by combining nxe with an appropriate intracluster measure.

In the context of intercluster measures, another interesting aspect is how vertices that are only loosely connected to the remainder of the graph are handled. For example, singletons with degree one have a low intercluster density of $1/(|V|-1)$ but maximum intercluster conductance of one. Thus, algorithms minimizing intercluster conductance are prone to put "outsiders" in the clusters of their neighbors, while algorithms minimizing intercluster density will tend to consider these vertices as singletons. Both views can be motivated, depending on the desiderata: If a vertex is linked to just a single vertex of a larger group, it can be hardly considered as an integral part of this group and should thus be treated separately. On the other hand, this vertex has no links to other groups and thus, from its point of view, it clearly has a strong affiliation to the group of its neighbor.

4.2 Problem Statement and Complexity

In the following we narrow down the myriad formalizations for combining intra- and intercluster quality and state the problem we focus on. Not only do these two aspects capture different properties of a clustering, they even tend to oppose each other: Fine clusterings easily achieve high intracluster density but poor intercluster quality, while the converse is true for coarse clusterings. In the light of a bottom-up strategy, intercluster sparsity aspires a coarse clustering and starts out poorly, which suggests using it as the driving objective function. By contrast, intracluster density starts out with the optimum value, which suggests it as a suitable constraint. We thus formalize our problem statement as follows, an exemplary instance and its solution are given in Figure 4.2.

Problem 1 (Density-Constrained Clustering (DCC))**.** *Given a graph $G = (V, E)$, among all clusterings with an intracluster density of no less than α, find a clustering $\mathcal{C}$ with optimum intercluster quality.*

Note that this problem definition is parameterized in the sense that each combination of intra- and intercluster quality function can be used to derive a concrete optimization problem in the context of Density-Constrained Clustering. If we consider all 14 measures for intercluster quality and the three measures for intracluster density defined

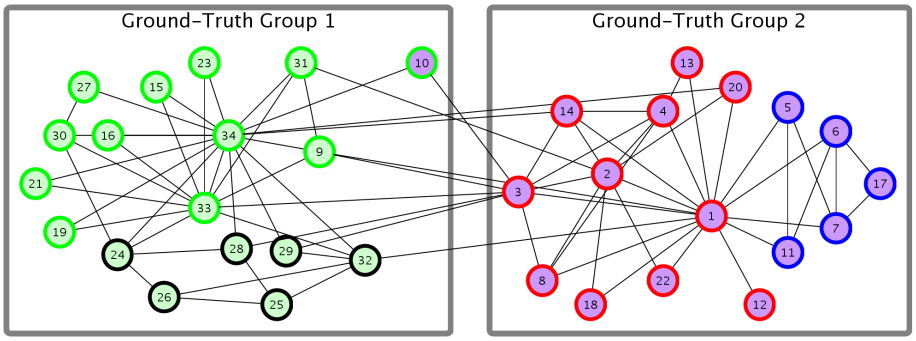


FIGURE 4.2: Zachary's karate club [232] represents a social network and is traditionally used for a test of feasibility in the graph clustering literature. Groups represent the split of the network in reality, fill colors depict the optimal solution to our problem statement using nxe constrained by mid with $\alpha = 0.25$. Reviewing an optimal solution (see Appendix A) helps judging the measure's usefulness independently of an algorithmic approach. For comparison, border colors indicate a Modularity-optimal clustering [173]. By contrast, aid yields the all clustering with the exception of one singleton vertex (12), pointing out its undesirable tendency to allow degenerately imbalanced clusterings.

above, this leads to a family of 42 optimization problems. An exhaustive study of hardness results for all combinations of intra- and intercluster measures is beyond the scope of this thesis. We derive $\mathcal{NP}$-hardness exemplarily for DENSITY-CONSTRAINED CLUSTERING combining mid, aid or gid with nxe and conjecture $\mathcal{NP}$-hardness for all remaining combinations. We use that, if we set $\alpha = 1$, the decision variant of our problem is equivalent to the following problem.

Problem 2 (CLUSTER DELETION)**.** *Given a graph $G = (V, E)$ and a parameter $r \in \mathbb{N}$, is there a subset $E' \subseteq E$ with $|E'| \leq r$ such that the connected components in $G' = (V, E \setminus E')$ are cliques?*

This problem is similar to both the classic problem PARTITION INTO CLIQUES [88], which instead minimizes the *number* of cliques and the edge-maximizing variant of the $\mathcal{K}_r$-PACKING PROBLEM [49], which differs in that it only allows cliques with bounded size. Shamir et al. [196] show that there is some constant $\varepsilon > 0$ such that it is NP-hard to approximate CLUSTER DELETION to within a factor of $1 + \varepsilon$; for an independent proof of the weaker statement that a problem equivalent to CLUSTER DELETION is NP-complete, see also our technical report [102]. As CLUSTER DELETION is a special case of the problem we are interested in, we get the following corollary.

Corollary 4.1. *There is some constant $\varepsilon > 0$ such that it is NP-hard to approximate* DENSITY-CONSTRAINED CLUSTERING *combining* mid, aid *or* gid *with* nxe *to within a factor of $1 + \varepsilon$.*

4.3 Related Work

The first part of this section deals with clustering based on dense subgraphs, whereas the second part is devoted to algorithms based on sparse cuts.

4.3.1 Dense Clusters

Although all notions of good clusterings are implicitly or explicitly based on the assumption that clusters are *dense subgraphs* in the sense that the corresponding vertices are connected by many edges, the precise definition of what a good cluster is varies among the literature. A common property of all established definitions of dense clusters is that *isolated cliques*, that is, cliques whose vertices are not connected to the remainder of the graph, are considered as perfect clusters in some sense.

Local definitions of clusters usually only depend on the subgraph induced by the cluster and, possibly, its connections to the other vertices. Early definitions of communities include the definition as *maximum cliques* in the graph [157]. Although this might seem appealing at first glance, this criterion is usually considered as too strict and relaxations of this concept are used. One possibility to do this is based on the observation that the diameter of cliques is always 1. The notions of *n-clique* [3, 156], *n-clan* and *n-club* [164] relax this property in the sense that they consider communities as subgraphs with diameter at most n, where n is "small".[2]

Closer to our definition of clusters as dense subgraphs are relaxations of cliques with respect to the number of neighbors a vertex must have in its community. Roughly, these relaxations can be classified in two categories. The first category considers communities that have the property that each vertex within the induced subgraph has a minimum number of neighbors inside the community. If this number k is constant and only maximal subgraphs with respect to this property are considered, one retrieves the definition of a *k-core* [194]. Similarly, a *k-plex* [195] is a maximal subgraph with the property that each vertex is connected to all but at most k vertices within the subgraph. Rather than requiring the degree of each vertex to be larger than a constant, Matsuda et al. [162] define a subgraph to be *p-quasi complete* if each of its vertices is connected to at least a pth fraction of the vertices within the subgraph. In the data mining community, p-quasi complete subgraphs are also known as *quasi-cliques* [153, 178, 233].

Another possibility to relax the constraint on the within-cluster degree is to require that the average degree of each vertex in the cluster is above a certain threshold. If the minimum threshold on the average within-cluster degree of the vertices is set to be an α fraction of the number of possible neighbors within the cluster, one retrieves our definition of a cluster with density at most α, understood as the density of the induced subgraph. Clusters that are dense in this respect are also sometimes referred to as quasi-cliques [2, 40]. Hartuv and Shamir's [118] definition of a *highly connected subgraph* equals the definition of a cluster of density at least $1/2$. Brunato et al. [40] unify both variants of quasi-cliques by requiring that both the minimum within-cluster degree of each vertex is higher than a threshold λ and the average within-cluster degree is higher than another parameter γ, with $\gamma \geq \lambda$.

Complexity and Enumeration Algorithms. Finding large dense subgraphs with respect to one of the definitions above or similar ones is a well-known problem that has been considered frequently in the literature, often in the context of clustering biological data. Usually, the corresponding decision problems are $\mathcal{NP}$-complete, starting with the classical problem CLIQUE that asks if a graph contains a clique of a given size [88]. Another example is the problem considered by Holzapfel et al. [119], which consists

[2] In contrast to the definition of n-clan and n-club, n-cliques also take into account paths running through vertices outside the community.

in deciding whether a given graph contains a subgraph of size k such that each of its vertices has an average within-cluster degree of $\gamma(k)$ for some function γ. They show that the problem is $\mathcal{NP}$-complete for $\gamma = 2 + \Omega(1/k^{1-\epsilon})$ and polynomially solvable for $\gamma = 2 + O(1/k)$. Feige et al. [76] give an approximation algorithm for the problem of finding a subgraph of size k with a maximum number of edges. Abello et al. [2] and Brunato et al. [40] use local search algorithms to find a large quasi-clique in a given graph. As these algorithms are randomized and are therefore able to find different quasi-cliques with different random seeds, they can be also used to find sets of quasi-cliques. Similarly, Bu et al. [41] identify quasi-cliques in protein interaction networks and evaluate their usefulness in the prediction of protein functionality. Other studies focus on the enumeration of all maximum quasi-cliques in a graph [153, 214] or across a set of different graphs [178, 233] using branching combined with different techniques to prune the search space. Matsuda et al. [162] address the problem of covering a given graph with a minimum number of quasi-cliques and propose a greedy algorithm for this problem. All of these approaches differ from Density-Constrained Clustering in that they either find sets of overlapping dense clusters or single clusters instead of a partition of the vertex set.

Partitioning Algorithms. Basu et al. [21] consider the problem to partition the graph into a minimum number of quasi-cliques and propose a greedy algorithm for this problem. In contrast to Density-Constrained Clustering, the number of edges linking the resulting clusters is not taken into account. Hartuv and Shamir [118] partition the graph into *highly connected components*, i.e., quasi-cliques in the sense that each vertex must be linked to at least half the vertices in its cluster. For that purpose, they iteratively split clusters that violate this constraints according to a minimum cut in the cluster. Hüffner et al. [122] formalize this as the optimization problem to minimize the number of edges cut by a clustering under the constraint that each cluster is a highly connected component. They show that this problem is $\mathcal{NP}$-hard and evaluate exact and heuristic algorithms for it, both with respect to the number of edges cut and the usefulness in classifying proteins. This can be considered as a special case of Density-Constrained Clustering combining mid with nxe, except for the difference that the minimum within-cluster vertex degree must be larger than a certain threshold instead of the average degree within each cluster.

Dengraph and SCAN. Other density-based graph clustering algorithms include Dengraph [73] and Scan [226]. The term density is here derived from the common base algorithm [72] that is used to identify dense regions in metric space. This intuition of density is quite different from our definition of a cluster as a dense subgraph and there is no apparent lower bound on the density of the resulting clusters in this sense.

4.3.2 Clustering based on sparse cuts

Terminology. Before reviewing some of the vast literature on sparse cuts in the context of clustering problems, we would like to spend a few words on terminology. In Section 4.5, we adopted the terms expansion and conductance that have been used for example by Kannan et al. [130] and many other authors. We chose the word density for the variant that normalizes the number of cut edges by the product of the sizes of the two cut sides as it is very close to the definition of the density of a cluster as the density of the induced subgraph. This corresponds to the definition of intercluster density used by Fortunato [83] in the context of graph clustering.

Due to the fact that these measures were proposed and rediscovered in different communities and application contexts, they are also known under different names that are often ambiguous. Lang and Rao [146] state that expansion is also known as *quotient cut*, whereas for example Spielman and Teng use the term *ratio cut* for the same objective. The minimal expansion of any cut in the graph is often called the *Cheeger constant* or *isoperimetric number* of the graph. Hagen and Kahng [115] and Chan et al. [48] use the name ratio cut not for expansion but for our definition of the density of a cut. Matula and Shahroki [163] use the term density for a generalization of density that takes demands into account[3] and call the associated optimization problem the *sparsest cut problem*.

Cut-based Clustering Measures. A common way to use conductance as a measure to evaluate the intercluster sparsity of a whole clustering is the following definition, which is often called *normalized cut* or *Ncut* [63, 219]:

$$\mathrm{Ncut}(\mathcal{C}) = \sum_{C \in \mathcal{C}} \frac{e_{C,V \setminus C}}{v_C}$$

This definition is equivalent to our notion of average isolated intercluster conductance in case the volume of each cluster is smaller than half the total vertex degree in the graph and the number of clusters is fixed. A similar generalization of expansion to whole clusterings is known as ratio cut [63, 219] or *RCut*. Interestingly, the ratio cut value of a clustering consisting of two clusters is equivalent to the density of the associated cut as defined in Section 4.1:

$$\mathrm{RCut}(\{A, B\}) = \frac{e_{A,B}}{n_A} + \frac{e_{A,B}}{n_B} = n \cdot \frac{e_{A,B}}{n_A \cdot n_B}$$

Approximating Sparse Cuts. There has been a substantial amount of research on approximation algorithms for the problem of finding sparse cuts. For example, Arora et al. [16] give $O(\sqrt{\log n})$ approximation algorithms for both the expansion and the conductance of a graph. As can be seen by the above conversion relating RCut to density, the density of a cut is bounded below by $1/n$ times its expansion, and bounded above by $2/n$ times its expansion. Hence, this also yields an $O(\sqrt{\log n})$ approximation algorithm for the problem of finding a cut of minimum density. For more information on approximation results, we refer to the related work mentioned by Madry [160].

Spectral Clustering. A very common approach to graph clustering, especially in the field of image segmentation, is the use of *spectral clustering*. Spectral clustering partitions the graph using the eigenvectors corresponding to the largest eigenvalues of some variant of the Laplacian matrix associated with the graph [219]. A very good explanation of the relation between spectral clustering and sparse graph cuts is given by von Luxburg [219]. Early examples of spectral algorithms explicitly designed to find sparse cuts include the spectral heuristic by Hagen and Kahng [115], which intends to find a cut of low density. This algorithm has been modified to k-way clustering by Chan et al. [48]. Shi and Malik's [199] spectral algorithm is designed to find cuts that are sparse with respect to the normalized cut objective. A detailed review on the vast literature on spectral clustering is beyond the scope of this chapter. A good overview

[3] This means that the term $n_S \cdot n_T$ in the denominator of density is replaced by $\sum_{s \in S, t \in T} D(s,t)$, where $D(s,t)$ is called the *demand* between s and t and can be arbitrarily chosen.

on earlier work in this context can be found in the article by Spielman and Teng [207], whereas von Luxburg [219] discusses some more recent articles.

Approximate Clustering Algorithms. Kannan et al. [130] show that the spectral algorithm of Shi and Malik yields a clustering with a conductance of at most $\sqrt{2 \cdot \mathrm{OPT}}$, where OPT is the minimum conductance over all cuts in the graph. They use this algorithm to analyze the performance of recursive spectral clustering in the context of the bicriterial problem minimizing both the minimum conductance of cuts within clusters and the number of edges cut by the clustering. Another variant of *iterative conductance cutting* they consider replaces spectral clustering with an approximation algorithm for conductance by Leighton and Rao [148]. Cheng et al. [51] propose to use the spectral variant of iterative conductance cutting to obtain a hierarchy of clusters or, in terms of the description in Chapter 3, a dendrogram of the data. For many objective functions like k-means or min-diameter, a good clustering can now be found in the second phase by computing the optimum clustering respecting the dendrogram efficiently. *Geometric MST Clustering* [37] first embeds the vertices in Euclidean space using the eigenvectors associated with the largest eigenvalues of the normalized adjacency matrix and uses the resulting distances to weigh the edges of the graph. Then, a weighted minimum spanning tree T is computed according to these edge weights. In the last step, a clustering is chosen according to some quality function among all clusterings that result from deleting the k heaviest edges in T, for $0 \leq k \leq n-1$. Flake et al. [79] use properties of minimum cut trees [93] to obtain a clustering algorithm with guarantees on both intra- and intercluster quality: The *intracluster expansion* of the resulting clusters, as defined by Kannan et al. [130], is larger than an input parameter α and each cluster C satisfies $x_C/|V \setminus C| \leq \alpha$. Louis and Makarychev [155] consider the *sparsest k-partitioning* problem, which generalizes the problem of splitting a graph into k clusters such that the maximum intercluster conductance or expansion is minimized. They give a bicriteria approximation algorithm that finds a clustering with at least $(1-\epsilon) \cdot k$ clusters whose conductance or expansion is at most $O_\epsilon(\sqrt{\log n \cdot \log k})$ times the optimum, using an SDP-relaxation.

Refinement Algorithms. The *Max Flow Quotient Cut Improvement (MQI)* algorithm by Lang and Rao [146] takes as input an arbitrary cut and returns a cut of lower conductance or expansion whose smaller side is contained in the smaller side of the original cut, if such a cut exists. The algorithm finds this cut by solving a flow problem in a modified graph. A combination of the graph partitioner Metis with MQI compares favorably to spectral algorithms in their experiments. Carrasco et al. [46] use the same approach to obtain a complete dendrogram of advertiser-keyword graphs. In their experiments, clusterings chosen from the dendrogram with few clusters show good behavior with respect to additional meta information on the data. Wei and Cheng [225] use a modification of the heuristic by Fiduccia and Mattheyses [77], which has been applied very successfully in the context of graph partitioning, to find a cut of low density.

Agglomerative and Multilevel Algorithms. Conan-Guez and Rossi [55] use a criterion similar to average isolated intercluster expansion in a greedy merge algorithm following the implementation of Müllner [166]. In contrast to our definition of intercluster expansion, the numerator of the quantity for each cluster is not equal to the number of edges leaving the cluster but the number of edges within the cluster; the goal is hence to maximize this measure. In a second phase, the resulting clustering is refined by a multilevel scheme based on greedy vertex moves, as described in Chapter 3. Karypis et al. [132] use essentially the same objective in the refinement phase of a multilevel

Algorithm 5: GREEDY MERGE DCC(GM)

Input : graph $G = (V, E)$, intracluster density measure i, intercluster sparsity measure x, α
Output: clustering $\mathcal{C}$ of G
$\mathcal{C} \leftarrow$ singletons
$\mathcal{A} \leftarrow \{\{A, B\} \in \binom{\mathcal{C}}{2} \mid i(\mathcal{C}_{\{A,B\}}) \geq \alpha \text{ and } x(\mathcal{C}_{\{A,B\}}) \leq x(\mathcal{C})\}$
while $|\mathcal{C}| > 1$ *and* $\mathcal{A} \neq \emptyset$ **do**
  $M \leftarrow \arg\min_{M \in \mathcal{A}} \{x(\mathcal{C}_M)\}$
  $\mathcal{C} \leftarrow \mathcal{C}_M$
  $\mathcal{A} \leftarrow \{\{A, B\} \in \binom{\mathcal{C}}{2} \mid i(\mathcal{C}_{\{A,B\}}) \geq \alpha \text{ and } x(\mathcal{C}_{\{A,B\}}) \leq x(\mathcal{C})\}$
return $\mathcal{C}$

algorithm. A second variant of this algorithm uses our definition of global intercluster density in the refinement phase. In the contraction phase of their algorithm, a SAHN algorithm is applied that uses as similarity measure between two clusters A and B the following function $f(A, B)$ that depends on an input parameter θ:

$$f(A, B) = \frac{e_{A,B}}{(n_A + n_B)^\theta - n_A^\theta - n_B^\theta}$$

For $\theta = 2$, this is similar to our definition of average or maximum pairwise intercluster density; however, it can be easily seen that f is not equivalent to greedily maximizing either of these measures. Dhillon et al. [63] consider the problem to minimize the sum of the intercluster expansion or conductance of all clusters subject to the constraint that the clustering contains exactly k clusters. In the coarsening phase of their multilevel algorithm, they compute greedy matchings based on the pairwise conductance or expansion of the cut separating the clusters represented by the vertices. The subsequent refinement phase is based on greedily moving vertices according to the objective function used.

4.4 Generic Greedy Agglomeration

In this section, we discuss greedy agglomeration algorithms that try to optimize an objective function in the presence of constraints on the feasibility of a clustering. In our context, the objective function will be one of our measures for intercluster sparsity, and the feasibility of a clustering depends on its intracluster density, i.e., we are interested in agglomerative greedy algorithms for DCC. The general structure of such an algorithm is given in Algorithm 5. It follows the generic scheme of Algorithm 2 discussed in Chapter 3 with intercluster sparsity as objective function except for the restriction to merge operations that lead to *feasible clusterings*, i.e., clusterings with an intracluster density larger than a given threshold α. We will identify key properties of intracluster density and intercluster sparsity measures that affect the efficiency and quality of greedy agglomeration and classify the measures proposed in Section 4.1 with respect to these properties.

4.4.1 Properties of Intercluster Sparsity and Intracluster Density Measures

This section aims to give a high-level overview on the properties of intercluster sparsity and intracluster density measures we define in this chapter; formal definitions and a classification of the measures we proposed in Section 4.1 with respect to these can be found in Section 4.4.2, Section 4.4.3, Section 4.4.4 and Section 4.4.5.

The first two properties are related to the nature of the resulting clusterings; they yield qualitative insights in the behavior of different measures in the context of greedy agglomeration, as well as insights into the measures themselves. The first property tries to formalize our intuition that intercluster sparsity strives towards coarse clusterings and therefore effectively drives greedy agglomeration. The second property relates to the question whether a particular intracluster measure favors fine clusterings and therefore does not impose an additional impediment to agglomeration. Informally, this translates to the following definitions.

- *unbounded merge behavior*: Informally, an intercluster sparsity function x exhibits unbounded merge behavior, if a greedy agglomeration algorithm that uses it as objective function and does not constrain the set of feasible merges always finds an improving merge, until it reaches the all clustering.
- *merge consistent*: An intracluster density measure i is merge consistent, if for any density threshold α, any clustering $\mathcal{C}$ and any refinement $\mathcal{D}$ of $\mathcal{C}$ that are both feasible, there is a sequence of merge operations transforming $\mathcal{D}$ into $\mathcal{C}$ such that all intermediate clusterings are feasible as well.

The second set of properties are relevant if we aim for efficient implementations of Algorithm 5. As already mentioned in Chapter 3, greedy merge algorithms are closely related to SAHN algorithms which are used to find groups of similar objects. In Section 4.4.3, we will review some algorithms that implement SAHN clustering and discuss how these algorithms can be adapted to the clustering of graphs in the context of Density-Constrained Clustering. It turns out that the applicability of the different algorithms depends on the following properties of the objective functions and constraints used in combination with Algorithm 5. More information on their relation to SAHN algorithms can be found in Section 4.4.3, thereby clarifying the different possibilities to implement greedy algorithms for particular instantiations of Density-Constrained Clustering.

- *efficiently comparable*: An intercluster sparsity measure is efficiently comparable if the comparison of the benefit of merge operations in Algorithm 5 can be carried out in constant time.
- *efficiently decidable*: An intracluster density measure is efficiently decidable if the feasibility of a merge operation in Algorithm 5 can be decided in constant time.
- *local*: Roughly speaking, an intercluster sparsity measure x is local if the benefit two merge operations have with respect to x can be compared without considering the remainder of the clustering. This admits for example the efficient maintenance of a set of possible merges in a priority queue.

TABLE 4.3: Summary of the properties of all intercluster measures

measure	abbr.	unbounded	eff. comparable	local	eff. loc. comp.	enf. conn. merges	reducible
number of intercluster edges	nxe	Y	Y	Y	Y	Y	N
global intercluster density	gxd	Y	Y	N	N	Y	N
max. is. intercluster density	mixd	Y	Y	Y	Y	N	N
avg. is. intercluster density	aixd	N	Y	Y	Y	N	?
max. pw. intercluster density	mpxd	Y	?	N	N	N	N
avg. pw. intercluster density	apxd	N	?	N	N	N	N
max. is. intercluster conductance	mixc	Y	Y	Y	Y	N	?
avg. is. intercluster conductance	aixc	Y	Y	Y	Y	N	?
max. pw. intercluster conductance	mpxc	N	?	N	N	N	N
avg. pw. intercluster conductance	apxc	N	?	N	N	N	N
max. is. intercluster expansion	mixe	Y	Y	Y	Y	N	?
avg. is. intercluster expansion	aixe	Y	Y	Y	Y	N	?
max. pw. intercluster expansion	mpxe	N	?	N	N	N	N
avg. pw. intercluster expansion	apxe	N	?	N	N	N	N

TABLE 4.4: Summary of the properties of all intracluster measures

measure	abbr.	efficiently decidable	merge consistent	context insensitive
minimum intracluster density	mid	Y	Y	Y
global intracluster density	gid	Y	Y	N
average intracluster density	aid	Y	N	N

- *context insensitive*: Similarly to the property locality in the context of intercluster sparsity, context insensitivity tries to formalize the notion if the feasibility of a merge can be decided locally, without considering the remainder of the clustering.
- *enforces connected merges*: Informally, an intercluster sparsity measure x enforces connected merges if the merge with the largest benefit with respect to x does not consist of unconnected clusters, i.e., Algorithm 5 only has to consider cluster pairs that are connected by at least one edge.
- *reducible:* Roughly speaking, an intercluster measure x is reducible, if merging cluster $A \cup B$ with another cluster C is never better than merging A or B with C.

An overview on the properties of all measures for intracluster density and intercluster sparsity introduced in Section 4.1 can be found in Table 4.3 and Table 4.4.

4.4.2 Merge Behavior

In this section, we give formal definitions of the properties properties unbounded and context insensitive and classify the intercluster sparsity and intracluster density measures from Section 4.1 with respect to these properties.

Merge Behavior of Intercluster Measures

Definition 4.2. An intercluster sparsity measure x has *unbounded merge behavior*, if for any clustering $\mathcal{C}$ with at least two clusters, there exist clusters $A \neq B \in \mathcal{C}$ such that $x(\mathcal{C}_{A,B}) \leq x(\mathcal{C})$.

We elucidate the merge behavior of each proposed intercluster measure, either by proving its unboundedness or by giving a graph together with a clustering that constitutes a local minimum. All proofs for unboundedness are constructive in that they point out how to find a non-increasing merge.

We start with the simple measures nxe, the number of intercluster edges in a clustering. Obviously, nxe can never increase when two clusters are merged and thus has unbounded merge behavior.

Observation 1. Nxe *has unbounded merge behavior.*

Most of the remaining proofs for unboundedness are based on the following simple observations on the results of adding the numerator and denominator of two fractions separately.

Observation 2. *Given $N_i, D_i \in \mathbb{Z}_0$ such that $D_i > 0$, let $N = \sum_{i=1}^{k} N_i$ and $D = \sum_{i=1}^{k} D_i$. Then, exactly one of the following cases holds:*

1. *There exist $\ell, s \in \{1, \dots, k\}$ with* $\dfrac{N_\ell}{D_\ell} > \dfrac{N}{D} > \dfrac{N_s}{D_s}$. *(strictly larger/smaller)*
2. *For all $i \in \{1, \dots, k\}$, it is* $\dfrac{N_i}{D_i} = \dfrac{N}{D}$. *(complete equality)*

Proof. We rewrite $\frac{N}{D}$ as a convex combination of its contributors $\beta_i := \frac{N_i}{D_i}$:

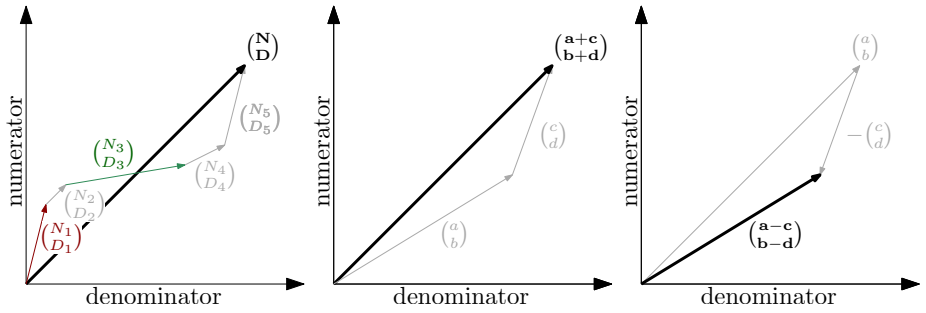
$$\frac{N}{D} = \frac{\sum_{i=1}^{k} N_i}{D} = \frac{\sum_{i=1}^{k} \frac{N_i}{D_i} \cdot D_i}{D} = \frac{\sum_{i=1}^{k} \beta_i \cdot D_i}{D} = \sum_{i=1}^{k} \beta_i \cdot \frac{D_i}{D}$$

Since $\sum_{i=1}^{k} D_i = D$, this is indeed a convex combination. Thus, if not all β_i are equal to $\frac{N}{D}$, we have $\frac{N}{D} \in (\min_{s=1}^{k} \beta_s, \max_{\ell=1}^{k} \beta_\ell)$, which immediately yields the claim. □

This immediately yields the following observations.

Observation 3. *Let $a, b, c, d > 0$ and $\frac{a}{b} \leq \frac{c}{d}$.*

(i) Then, $\frac{a+c}{b+d} \geq \frac{a}{b}$.



(a) Observ. 2 (b) Observ. 3(i) (c) Observ. 3(ii)

FIGURE 4.3: Illustration for the geometric interpretation of Observation 2 and Observation 3. For $a, b, c, d > 0$, $\frac{a}{b}$ can be interpreted as the slope of the vector $\binom{a}{b}$ and $\frac{a+c}{b+d}$ as the slope of $\binom{a}{b} + \binom{c}{d}$. Now, Observation 2 corresponds to the fact that the slope of the sum of a set of vectors is always within the range of the slopes of the individual vectors (a) and Observation 3 can be illustrated by the effect of adding or subtracting vectors with a higher slope (see (b) and (c)).

(ii) If $b > d$, $\frac{a-c}{b-d} \leq \frac{a}{b}$.

Proof. (i) Follows directly from Observation 2.

(ii) It is $\frac{a}{b} = \frac{(a-c)+c}{(b-d)+d}$ and hence, with Observation 2, $\frac{a-c}{b-d} \leq \frac{a}{b}$.

□

The above observations also have a geometric interpretation illustrated in Figure 4.3. The idea is to interpret the numerator and denominator of fractions as the coordinates of a two dimensional vector. Summing up the numerators and denominators of two fractions separately and computing the resulting fraction corresponds to determining the slope of the sum of the two vectors associated with the initial fractions.

Recall that gxd is defined as $\frac{\sum_{A \neq B \in \mathcal{C}} e_{A,B}}{\sum_{A \neq B \in \mathcal{C}} n_A n_B}$. Observation 2 tells us that there exists at least one pair of clusters (C, D) such that $\frac{e_{C,D}}{n_C n_D} \geq \mathsf{gxd}(\mathcal{C})$. In terms of gxd, merging the clusters C and D is equivalent to "vectorially subtracting" from $\frac{\sum_{A \neq B \in \mathcal{C}} e_{A,B}}{\sum_{A \neq B \in \mathcal{C}} n_A n_B}$ the contributor $\frac{e_{C,D}}{n_C n_D}$. By Observation 3(ii), we get $\mathsf{gxd}(\mathcal{C}_{\{C,D\}}) \leq \mathsf{gxd}(\mathcal{C})$, a non-increase. This yields the following corollary.

Corollary 4.3. Gxd *has unbounded merge behavior.*

Proposition 4.4. Mixc *has unbounded merge behavior.*

Proof. We focus on the worst cluster $B = \arg\max_{C \in \mathcal{C}} \mathsf{ixc}(C)$ (which need not be unique), and distinguish three cases by the volume v_B of B. In each case we will either constructively show that a non-increasing merge is possible or show that the absence of such a possibility leads to a contradiction. Note that it suffices to identify two clusters C and D that satisfy $\mathsf{ixc}(C \cup D) \leq \mathsf{ixc}(B) = \mathsf{mixc}(\mathcal{C})$, as we are only interested in the cluster with maximum intercluster conductance and merging two clusters does not affect the

conductance of the remaining clusters. Without loss of generality we may assume that $|\mathcal{C}| \geq 3$.

$v_B \geq m$: We express $\beta := \mathsf{ixc}(B)$ as the "vectorial sum" of other clusters' contributions:

$$\beta = \frac{x_B}{2m - v_B} = \frac{\sum_{C \neq B} e_{C,B}}{\sum_{C \neq B} v_C}$$

By Observation 2, there must either exist a cluster A in $\mathcal{C}$ with $\frac{e_{A,B}}{v_A} > \beta$, or we have for all $C \neq B$: $\frac{e_{C,B}}{v_C} = \beta$. Since $\mathsf{ixc}(C) \geq \frac{e_{C,B}}{v_C}$ for all $C \neq B \in \mathcal{C}$, the former case yields $\mathsf{ixc}(A) > \beta$, contradicting our choice of B; for the same reason, the latter case either also yields such a cluster A, or we have for all $C \neq B : \mathsf{ixc}(C) = \beta$. But then merging any $C, D \neq B$ is non-increasing: $\mathsf{ixc}(C \cup D) = \frac{x_C + x_D - 2e_{C,D}}{v_C + v_D} \leq \frac{x_C + x_D}{v_C + v_D}$. By Observation 2, this is exactly $\beta = \mathsf{mixc}(\mathcal{C})$.

$v_B < m$ and for all $C \neq B : v_B + v_C \geq m$: For an arbitrary merge $\{C, B\}$, it is

$$\mathsf{ixc}(B \cup C) = \frac{x_B + x_C - 2e_{C,B}}{2m - v_B - v_C} = \frac{\sum_{D \neq B,C} e_{D,B \cup C}}{\sum_{D \neq B,C} v_D}.$$

Hence, by the above arguments, there must either exist a cluster A with $\mathsf{ixc}(A) > \mathsf{ixc}(C \cup B)$, or we have for all $D \neq C, B : \mathsf{ixc}(D) = \mathsf{ixc}(C \cup B)$. In either case merge $\{C, B\}$ is non-increasing.

$v_B < m$ and there exists $A \neq B$ such that $v_A + v_B < m$: Merging A and B yields $\mathsf{ixc}(A \cup B) = \frac{x_A + x_B - 2e_{A,B}}{v_A + v_B} \leq \frac{x_A + x_B}{v_A + v_B}$. Since $\frac{x_A}{v_A} \leq \frac{x_B}{v_B} = \beta$, by Observation 3(i) we can see that $\frac{x_A + x_B}{v_A + v_B} \leq \beta = \mathsf{mixc}(\mathcal{C})$; hence, merging A and B does not increase mixc. □

Proposition 4.5. Aixc *has unbounded merge behavior.*

Proof. Let $\mathcal{C}$ be an arbitrary clustering and $\beta := \mathsf{aixc}(\mathcal{C})$. We have to show that there exists a merge that does not increase the objective function. For clusters $C, D \in \mathcal{C}$ with $v_C + v_D \leq m$, we will use that, by Observation 2, merging them yields

$$\mathsf{ixc}(C \cup D) = \frac{x_C + x_D - 2e_{C,D}}{v_C + v_D} \leq \frac{x_C + x_D}{v_C + v_D} \leq \max\{\mathsf{ixc}(C), \mathsf{ixc}(D)\} \tag{4.1}$$

We divide the clusters of an arbitrary clustering $\mathcal{D}$ into "bad" and "good" clusters, depending on β: $\mathcal{B}_\mathcal{D} = \{B \in \mathcal{D} \mid \mathsf{ixc}(B) > \beta\}$ and $\mathcal{G}_\mathcal{D} = \{H \in \mathcal{D} \mid \mathsf{ixc}(H) \leq \beta\}$, respectively. Now we can link the quality of D with β as follows:

$$|\mathcal{D}| \cdot \mathsf{aixc}(\mathcal{D}) = |\mathcal{D}| \cdot \beta + \underbrace{\textstyle\sum_{B \in \mathcal{B}_\mathcal{D}} (\mathsf{ixc}(B) - \beta)}_{:= b_\mathcal{D}} - \underbrace{\textstyle\sum_{H \in \mathcal{G}_\mathcal{D}} (\beta - \mathsf{ixc}(H))}_{:= g_\mathcal{D}}$$

It follows directly that $\mathsf{aixc}(\mathcal{D}) \leq \beta = \mathsf{aixc}(\mathcal{C})$ iff $b_\mathcal{D} \leq g_\mathcal{D}$. Hence, it remains to find two clusters $B_1 \neq B_2$ in $\mathcal{C}$ such that $b_{\mathcal{C}_{\{B_1,B_2\}}} \leq g_{\mathcal{C}_{\{B_1,B_2\}}}$. If there exist $B_1 \neq B_2 \in \mathcal{B}_\mathcal{C}$ with $v_{B_1} + v_{B_2} \leq m$, $\mathcal{G}_{\mathcal{C}_{\{B_1,B_2\}}} = \mathcal{G}_\mathcal{C}$ and hence $g_{\mathcal{C}_{\{B_1,B_2\}}} = g_\mathcal{C}$. Without loss of generality, let us assume that $\mathsf{ixc}(B_1) > \mathsf{ixc}(B_2)$. As explained above, it is $\mathsf{ixc}(B_1 \cup B_2) \leq \max\{\mathsf{ixc}(B_1), \mathsf{ixc}(B_2)\}$. Hence, merging B_1 and B_2 reduces $b_\mathcal{C}$ by

$$\mathsf{ixc}(B_1) + \mathsf{ixc}(B_2) - 2\beta - \mathsf{ixc}(B_1 \cup B_2) + \beta = \underbrace{\mathsf{ixc}(B_1) - \mathsf{ixc}(B_1 \cup B_2)}_{\geq 0} + \underbrace{\mathsf{ixc}(B_2) - \beta}_{> 0} > 0,$$

which implies that $\mathsf{aixc}(\mathcal{C}_{\{B_1,B_2\}}) < \mathsf{aixc}(\mathcal{C})$. Therefore, assume that for all $B_1 \neq B_2 \in \mathcal{B}_\mathcal{C} : v_{B_1} + v_{B_2} > m$. We distinguish five cases according to $|\mathcal{B}_\mathcal{C}|$:

$|\mathcal{B}_\mathcal{C}| \geq 4$: Since $\sum_{C\in\mathcal{C}} v_C = 2m$, there must exist B_1, B_2 with $v_{B_1} + v_{B_2} \leq m$, which contradicts our assumption.

$|\mathcal{B}_\mathcal{C}| = 3$: We sort $\mathcal{B}_\mathcal{C}$ by $\mathsf{ixc}(B_1) \geq \mathsf{ixc}(B_2) \geq \mathsf{ixc}(B_3)$. Merging B_1 and B_2 yields

$$\mathsf{ixc}(B_1 \cup B_2) = \frac{x_{B_1\cup B_2}}{\sum_{C\neq B_1,B_2} v_C} = \frac{\sum_{C\neq B_1,B_2} e_{B_1\cup B_2,C}}{\sum_{C\neq B_1,B_2} v_C} \leq \frac{\sum_{C\neq B_1,B_2} x_C}{\sum_{C\neq B_1,B_2} v_C}$$

Among the set of clusters not containing B_1 and B_2, B_3 is the cluster with the highest isolated intercluster conductance, as all other clusters belong to $\mathcal{G}_\mathcal{C}$. Hence, with Observation 2, it follows that

$$\mathsf{ixc}(B_1 \cup B_2) \leq \frac{\sum_{C\neq B_1,B_2} x_C}{\sum_{C\neq B_1,B_2} v_C} \leq \mathsf{ixc}(B_3).$$

Therefore, the merge $\{B_1, B_2\}$ reduces $b_\mathcal{C}$, leaving $\mathcal{G}_\mathcal{C}$ untouched, which yields an improving merge.

$|\mathcal{B}_\mathcal{C}| = 2$: Analogous to case $|\mathcal{B}_\mathcal{C}| = 3$, by defining B_3 as a cluster in $\mathcal{G}_\mathcal{C}$ with maximum isolated intercluster conductance, we can see that $\mathsf{ixc}(B_1 \cup B_2) \leq \mathsf{ixc}(B_3) \leq \beta$.

$|\mathcal{B}_\mathcal{C}| = 1$: If, for some $H \in \mathcal{G}_\mathcal{C}$ we have $\mathsf{ixc}(B \cup H) \leq \beta$, the merge $\{B, H\}$ does not increase aixc, as after the merge the isolated intercluster conductance of every cluster is at most $\beta = \mathsf{aixc}(\mathcal{C})$; thus, assume that

$$\forall H \in \mathcal{G}_\mathcal{C} : \mathsf{ixc}(B \cup H) > \beta. \tag{$*$}$$

If, for some $H \in \mathcal{G}_\mathcal{C}$ we have $v_B + v_H > m$, we can use the same arguments as in the case $|\mathcal{B}_\mathcal{C}| = 3$, replacing B_1 by B, B_2 by H and B_3 by the cluster in $\mathcal{G}_\mathcal{C}$ having the highest isolated intercluster conductance to see that $\mathsf{ixc}(B \cup H) \leq \beta$, which contradicts $(*)$. Hence, $v_B + v_H \leq m$ for all $H \in \mathcal{G}_\mathcal{C}$. For a merge $\{B, H\}$, we denote by $\Delta_b(H) := x_B/v_B - x_{B\cup H}/v_{B\cup H}$ the change of $b_\mathcal{C}$, and the change of $g_\mathcal{C}$ by $\Delta_g(H) := \beta - x_H/v_H$; the crucial observation is that any merge $\{B, H\}$ for which $\Delta_b(H) > \Delta_g(H)$ holds, improves aixc; but such a merge must exist:

$$\begin{aligned}
\underbrace{(|\mathcal{G}_\mathcal{C}| + 1)}_{=|\mathcal{C}|}\beta &= \frac{x_B}{v_B} + \sum_{H\in\mathcal{G}_\mathcal{C}} \frac{x_H}{v_H} + = \sum_{H\in\mathcal{G}_\mathcal{C}} \left(\frac{x_H}{v_H} + \frac{e_{B,H}}{v_B}\right) \leq \sum_{H\in\mathcal{G}_\mathcal{C}} \left(\frac{x_H}{v_H} + \frac{2e_{B,H}}{v_B}\right) \\
&= \sum_{H\in\mathcal{G}_\mathcal{C}} \left(\frac{x_H}{v_H} + \frac{x_B}{v_B} - \frac{x_B - 2e_{B,H}}{v_B}\right) \leq \sum_{H\in\mathcal{G}_\mathcal{C}} \left(\frac{x_H}{v_H} + \frac{x_B}{v_B} - \underbrace{\frac{x_B - 2e_{B,H} + x_H}{v_B + v_H}}_{=\mathsf{ixc}(B\cup H)}\right)
\end{aligned}$$

To see the second inequality, observe the following: By $(*)$ $\mathsf{ixc}(B \cup H) > \beta$ and since $H \in \mathcal{G}_\mathcal{C}$, $\mathsf{ixc}(H) \leq \beta$. Hence $\frac{x_H}{v_H} = \mathsf{ixc}(H) < \mathsf{ixc}(B \cup H)$ and Observation 2 yields that

$$\mathsf{ixc}(B \cup H) = \frac{x_B - 2e_{B,H} + x_H}{v_B + v_H} \leq \frac{x_B - 2e_{B,H}}{v_B}.$$

Since we have only $|\mathcal{G}_\mathcal{C}|$ summands, at least one summand exceeds β. More precise, there exists $H_0 \in \mathcal{G}_\mathcal{C}$ such that

$$\frac{x_{H_0}}{v_{H_0}} + \frac{x_B}{v_B} - \underbrace{\frac{x_B - 2e_{B,H_0} + x_{H_0}}{v_B + v_{H_0}}}_{=x_{B\cup H_0}/v_{B\cup H_0}} > \beta$$

It follows that $\Delta_b(H_0) > \Delta_g(H_0)$.

$|\mathcal{B}_\mathcal{C}| = 0$: It is $\mathsf{ixc}(C) = \beta$ for all $C \in \mathcal{C}$. Let $C_1, C_2 \in \mathcal{C}$ be the two clusters with the smallest volume. If $v_{C_1} + v_{C_2} \leq m$, again, Observation 2 yields that $\mathsf{ixc}(C_1 \cup C_2) \leq \max\{\mathsf{ixc}(C_1), \mathsf{ixc}(C_2)\} = \beta$. Otherwise, $|\mathcal{C}| \leq 3$ and as we assumed that $|\mathcal{C}| \geq 3$, $|\mathcal{C}| = 3$. Hence, $\mathsf{ixc}(C_1 \cup C_2)$ equals the isolated intercluster conductance of the third cluster, which is β. In both cases, $\mathsf{aixc}(\mathcal{C}_{C_1,C_2}) \leq \beta$. □

A careful inspection of the proofs of Proposition 4.4 and Proposition 4.5 reveals, that if we substitute any occurrence of volume v_C by the size n_C of a cluster and always distinguish by $n_C \geq n/2$ instead of $v_C \geq m$, the conclusions remain correct, but translate the results from conductance to expansion. We summarize this result in the following corollary on the measures mixe and aixe:

Corollary 4.6. Mixe *and* aixe *have unbounded merge behavior.*

Proposition 4.7. Mixd *has unbounded merge behavior.*

Proof. For a worst cluster $B = \arg\max_{C\in\mathcal{C}} \mathsf{ixd}(C)$ (which need not be unique) with $\mathsf{ixd}(B) = \mathsf{mixd}(\mathcal{C}) =: \beta$, we show how to find a non-increasing merge. Merging any cluster C with B yields

$$\mathsf{ixd}(C \cup B) = \frac{x_C + x_B - 2e_{C,B}}{n_C(n - n_C) + n_B(n - n_B) - 2n_C n_B} \ ,$$

which is the "vectorial addition" of $\mathsf{ixd}(C)$, $\mathsf{ixd}(B)$ and the *correction* $\kappa_C := \frac{2e_{C,B}}{2n_C n_B}$. If we find a cluster A such that $\kappa_A \geq \mathsf{ixd}(B)$, by Observation 3 (ii), it is

$$\frac{x_B - 2e_{A,B}}{n_B(n - n_B) - 2n_A n_B} := \alpha \leq \beta$$

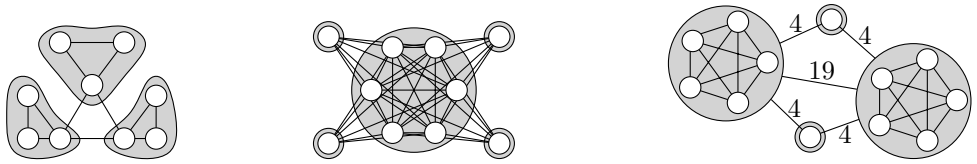
and, by Observation 2, $\mathsf{ixd}(A\cup B) \leq \max\{\mathsf{ixd}(\mathcal{C}), \alpha\} \leq \beta$. Hence, merging A and B does not increase mixd. To see that such a cluster A must exist, we decompose

$$\mathsf{ixd}(B) = \frac{\sum_{C\neq B} 2e_{C,B}}{\sum_{C\neq B} 2n_C n_B}$$

into such corrections κ_C; the claim now follows from Observation 2. □

Proposition 4.8. Mpxd *has unbounded merge behavior.*

Proof. Let $\{A, B\} = \arg\max_{M\in\mathcal{M}} \mathsf{pxd}(M)$. If we merge A and B, one of the pairs with maximum pairwise intercluster density is not present in the clustering any more. The



(a) mpxc, mpxe (b) apxd, apxc, apxe (c) aixd (edges are summarized)

FIGURE 4.4: These instances illustrate bounded merge behavior. Given clustering $\mathcal{C}$ (gray), no further merge is non-increasing for the measures pointed out.

following inequality shows that the merge $\{A, B\}$ also cannot cause new, worse values $\mathsf{pxd}(\{A \cup B, C\})$, which shows that it does not increase $\mathsf{mixd}(\mathcal{C})$:

$$\text{For all } C \neq A \cup B : \mathsf{pxd}(\{A \cup B, C\}) = \frac{e_{A\cup B,C}}{n_{A\cup B} n_C} = \frac{e_{A,C} + e_{B,C}}{n_A n_C + n_B n_C}$$
$$\leq \max\{\mathsf{pxd}(\{A, C\}), \mathsf{pxd}(\{B, C\})\} \leq \mathsf{pxd}(\{A, B\})$$

□

We have now seen that the basic measures nxe, gxd, as well as the isolating measures of conductance and of expansion mixc, mixe, aixc, aixe and the maximum measures of density mixd, mpxd exhibit unbounded merge behavior. From our set of fourteen objective functions, the remaining six do not have unbounded merge behavior, but can instead get stuck in local minima, such that no further merge is non-increasing. Thus, even without a constraint, the all clustering cannot be reached. In Figure 4.4 we give specific instances which are local minima of mpxc and mpxe (a), apxd, apxc and apxe (b), and of aixd (c). The common intuition for average measures is that a merge must not reduce the number of beneficial clusters (or pairs thereof) too dearly. As an example, we discuss the counterexample in Figure (b) for the pairwise measures. Here, a clique of size six is surrounded by four satellites such that the satellites are not connected among each other, but fully connected to the clique. In the context of average pairwise measures, we therefore have four very bad contributors (clique with satellite) and four very good contributors (satellite with satellite). As expected, merging satellite with satellite increases all measures. Somehow unexpected, merging a satellite with the clique has the same effect, although the resulting cluster is again a clique. The reason for that is that the number of good pairwise cuts is reduced by three, whereas the number of bad cuts just decreases by one. One might argue that in this example the satellites have less connections than the vertices in the clique and it is therefore justified to leave them unclustered. However, the actual density of the cluster in the middle does not matter for intercluster measures, hence replacing the clique by a less dense subgraph does not change anything.

Mpxc and mpxe are prone to local minima near balanced clusterings. Roughly speaking, this is due to the case distinction in the denominator of their base measures ruining arguments analogous to those in Proposition 4.8 for mpxd. As an example, Figure (a) shows three clusters with pairwise expansion of 1/3. If we merge any two clusters, the expansion of the resulting cut does not count the number of vertices in the merged cluster in the denominator anymore, but the number of vertices in the other cluster, leading to an expansion of 2/3.

Corollary 4.9. Mpxc, mpxe, apxd, apxc, apxe, *and* aixd *have bounded merge behavior.*

Merge Behavior of Intracluster Measures

Definition 4.10. An intracluster measure is *merge consistent*, if for any clustering $\mathcal{D}$ that is feasible with respect to the density constraint and any feasible refinement $\mathcal{C}$ of $\mathcal{D}$, there is a sequence of feasible merge operations that transforms $\mathcal{C}$ into $\mathcal{D}$.

This directly implies that each feasible clustering can possibly be reached by Algorithm 5, provided that the objective function chooses the right merge in each step.

Lemma 4.11. *Given density α, a clustering $\mathcal{C}$ and a subset $\mathcal{D} \subseteq \mathcal{C}$ with $|\mathcal{D}| \geq 2$ such that for all $D \in \mathcal{D}$ we have $\mathsf{id}(D) \geq \alpha$. If $U = \bigcup_{D \in \mathcal{D}} D$ is feasible, i.e., $\mathsf{id}(U) \geq \alpha$, then there exist $A \neq B \in \mathcal{D}$ such that $\mathsf{id}(A \cup B) \geq \alpha$.*

Proof. Assume to the contrary that all merges of clusters in $\mathcal{D}$ are infeasible, i.e., the density of the resulting cluster is less than α. From this, it follows directly that for all $A, B \in \mathcal{D}$, it is $e_{A \cup B} < \alpha \cdot \binom{n_{A \cup B}}{2}$. Furthermore, the intracluster edges (pairs) in $A \cup B$ can be decomposed into intracluster edges (pairs) of A and B and edges (pairs) linking both clusters. Hence,

$$\alpha = \frac{\sum_{\{A,B\} \in \binom{\mathcal{D}}{2}} \alpha \cdot \binom{n_{A \cup B}}{2}}{\sum_{\{A,B\} \in \binom{\mathcal{D}}{2}} \binom{n_{A \cup B}}{2}} > \frac{\sum_{\{A,B\} \in \binom{\mathcal{D}}{2}} e_{A \cup B}}{\sum_{\{A,B\} \in \binom{\mathcal{D}}{2}} \binom{n_{A \cup B}}{2}} = \frac{\sum_{\{A,B\} \in \binom{\mathcal{D}}{2}} e_A + e_B + e_{A,B}}{\sum_{\{A,B\} \in \binom{\mathcal{D}}{2}} \binom{n_A}{2} + \binom{n_B}{2} + n_A n_B}$$

Note that the number of intracluster edges (pairs) of each cluster appears exactly $|\mathcal{D}| - 1$ times in the above sum. If we use this and the fact that the number of intracluster edges (pairs) in U can be decomposed as above, we get

$$\frac{\sum_{\{A,B\} \in \binom{\mathcal{D}}{2}} e_A + e_B + e_{A,B}}{\sum_{\{A,B\} \in \binom{\mathcal{D}}{2}} \binom{n_A}{2} + \binom{n_B}{2} + n_A n_B} = \frac{e_U + \sum_{A \in \mathcal{D}} (|\mathcal{D}| - 2) \cdot e_A}{\binom{n_U}{2} + \sum_{A \in \mathcal{D}} (|\mathcal{D}| - 2) \cdot \binom{n_A}{2}}$$

At this point, we can use that the density of U as well as the density of each cluster in $\mathcal{D}$ is at least α:

$$\frac{e_U + \sum_{A \in \mathcal{D}} (|\mathcal{D}| - 2) \cdot e_A}{\binom{n_U}{2} + \sum_{A \in \mathcal{D}} (|\mathcal{D}| - 2) \cdot \binom{n_A}{2}} \geq \frac{\alpha \cdot \binom{n_U}{2} + \sum_{A \in \mathcal{D}} (|\mathcal{D}| - 2) \cdot \alpha \cdot \binom{n_A}{2}}{\binom{n_U}{2} + \sum_{A \in \mathcal{D}} (|\mathcal{D}| - 2) \cdot \binom{n_A}{2}} = \alpha$$

In summary, we have proven that $\alpha > \alpha$ if the assumption holds, leading to a contradiction. □

Note that this result can be directly transferred to mid, since this constraint classifies each merge individually, unlike aid and gid.

Corollary 4.12. Mid *is merge consistent.*

For gid and aid the contributions of other unrelated clusters matter, a fact, which has further consequences on computational complexity, as discussed in the next section. However, for gid, it is possible to use the following Lemma.

Lemma 4.13. *Given a parameter α, two clusterings $\mathcal{C}$ and $\mathcal{C}'$ with $\mathsf{gid}(\mathcal{C}) \geq \alpha$ and $\mathsf{gid}(\mathcal{C}') \geq \alpha$ such that $\mathcal{C}$ is a (proper) refinement of $\mathcal{C}'$. Then, there exists a pair of clusters A and B in $\mathcal{C}$ such that $\mathcal{C}_{A,B}$ is a refinement of $\mathcal{C}'$ with $\mathsf{gid}(\mathcal{C}_{A,B}) \geq \alpha$.*

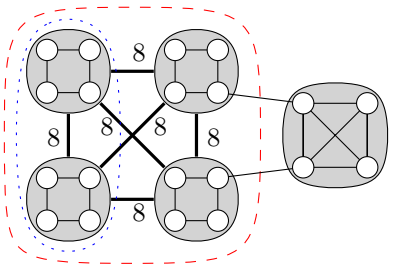


FIGURE 4.5: Example showing that aid is not merge consistent. If the constraint aid > 0.73 is used, the blue, dotted merge is not allowed whereas merging all clusters indicated by the red, dashed line is feasible. Fat lines indicate aggregated edges between clusters.

Proof. Let $D_1, \ldots, D_r$ denote the clusters in $\mathcal{C}'$ that contain more than one cluster in $\mathcal{C}$ and $\mathcal{D} := \{\mathcal{D}_1, \ldots, \mathcal{D}_r\}$ the set of the corresponding subsets of clusters in $\mathcal{C}$. The global intracluster density of $\mathcal{C}'$ can be written as a vectorial sum of the densities of the clusters in $\mathcal{C}$ and the density of the pairwise cuts between clusters in the same set $\mathcal{D}_i$:

$$\mathsf{gid}(\mathcal{C}') = \frac{\sum_{C\in\mathcal{C}} e_C + \sum_{\mathcal{D}_i\in\mathcal{D}} \sum_{\{A,B\}\in\binom{\mathcal{D}_i}{2}} e_{A,B}}{\sum_{C\in\mathcal{C}} \binom{n_C}{2} + \sum_{\mathcal{D}_i\in\mathcal{D}} \sum_{\{A,B\}\in\binom{\mathcal{D}_i}{2}} n_A n_B} \geq \alpha \tag{4.2}$$

Among all pairs of clusters in $\mathcal{C}$ that belong to a common set in $\mathcal{D}$, let A', B' be the pair such that the associated cut has maximum density, i.e. $e_{A',B'}/(n_{A'}n_{B'})$ is maximal. Assume that this merge is not feasible, i.e.,

$$\frac{\sum_{C\in\mathcal{C}} e_C + e_{A',B'}}{\sum_{C\in\mathcal{C}} \binom{n_C}{2} + n_{A'}n_{B'}} < \alpha$$

As $\mathcal{C}$ is feasible, it follows that $e_{A',B'}/(n_{A'}n_{B'}) < \alpha$ and with Observation 2 it can be seen that the global density of the cuts of the remaining pairs of clusters contained in a common set, i.e. the vectorial sum of their contribution to $\mathsf{gid}(\mathcal{C}')$, is also less than α. Hence, including these pairs in the vectorial sum cannot increase the global intracluster density to a value of at least α, which contradicts Equation (4.2). □

Iterative application of Lemma 4.13 yields the following corollary.

Corollary 4.14. Gid *is merge consistent.*

In contrast to that, aid is not merge consistent, as the example in Figure 4.5 illustrates. Similar to some proofs in the preceding section, the idea behind this counterexample is that besides their density, the actual number of bad contributors matters. In the original clustering, we have four clusters with a density of 2/3, mutually separated by cuts of density 1/2, and one cluster with an optimal density of 1. Merging only two clusters in the left part leads to a new cluster with worse density while not reducing the number of bad contributors enough. In contrast to that, merging all clusters leads to a cluster with even worse quality, but this is compensated by the stronger influence of the cluster to the right, so that the average density over all clusters is feasible again.

Corollary 4.15. Aid *is not merge consistent.*

Table 4.5: Sufficient conditions for implementing variants of Greedy Merge DCC

algorithm	worst case running time	add. space	eff. comparable	local	eff. loc. comp.	enf. conn. merges	reducible	eff. decidable	cont. insensitive
Straightforward	$O(n^3)$	$O(1)$	√	-	-	-	-	√	-
Straightforward Sparse	$O(nm)$	$O(1)$	√	-	-	√	-	√	-
Nearest Neighbor List	$O(n^3)$	$O(n)$	√	√	√	-	-	√	√
Nearest Neighbor List Sparse	$O(nm)$	$O(n)$	√	√	√	√	-	√	√
Heap	$O(n^2 \log n)$	$O(n^2)$	√	√	√	-	-	√	√
Heap Sparse	$O(md \log n)$	$O(m)$	√	√	√	√	-	√	√
Quadtree	$O(n^2)$	$O(n^2)$	√	√	√	-	-	√	√
Conga Line	$O(n^2 \log^2 n)$	$O(n)$	√	√	√	-	-	√	√
Nearest Neighbor Chaining	$O(n^2)$	$O(n)$	√	√	√	-	√	√	√

4.4.3 Transferring SAHN Algorithms to DCC

In this section, we will review some basic algorithmic approaches to implement SAHN clustering (cf. Algorithm 1 in Chapter 3). Some of these have been already discussed in Section 3.3.1 in the context of modularity based graph clustering. For each of these approaches, we will scrutinize under which circumstances they can be transferred to Greedy Merge DCC. More specific, we discuss what properties the intercluster sparsity and intracluster density measure have to fulfill such that these algorithms can be used to implement Algorithm 5. Table 4.5 summarizes the insights from this discussion.

When we speak of the running time and space requirement of each algorithm, we assume that we always maintain the current cluster graph as an adjacency list in main memory, as described in Section 3.3.1 in the context of modularity based clustering. The weight of the edges in the cluster graph corresponds to the number of edges between the associated clusters in the original graph. Additionally, each vertex in the cluster graph stores the number of vertices n_C, the volume v_C and the number of intracluster edges e_C of the cluster C it represents. As justified in Section 3.3.1, the cluster graph can be maintained in $O(n^2)$ time during the whole agglomeration process, which incurs no additional overhead as the fastest of the algorithms described below have a worst case complexity in $O(n^2)$. We further assume that the informations from the cluster graph can be retrieved in constant time. Technically, this not entirely correct as retrieving the weight of an edge in an adjacency list entails parsing the edge list of an incident vertex. It can however be verified that all algorithms access this information locally in the sense that all neighbors of a cluster are evaluated at the same time, which leads to amortized constant time for each neighbor.

Straightforward. Probably the first generic view on hierarchical clustering independent from any particular distance update schemes dates back to the article of Ward [221] from 1963. Technically speaking, his description does not fit into the context of SAHN

clustering as he does not speak of objects with pairwise distances but considers hierarchical clustering as a means to greedily optimize an objective function while producing a complete dendrogram. Hence, what he describes corresponds to the Generic Greedy Merge algorithm introduced in Chapter 3. In line with the traditional distance-based view on SAHN algorithms, he assumes that the objective function is optimal for the singleton clustering and in each step, the merge that decreases the objective function the least is executed. As an example he considers the total within-cluster variance, which is often referred to as *Ward's criterion* [166], but he also briefly discusses the use of other objective functions. Furthermore, he describes a straightforward generic implementation of Generic Greedy Merge determining the best merge in each iteration by a linear scan over all pairs of clusters, which leads to a running time in $O(n^3)$ for the most common objective functions, which we term Straightforward.

It is easy to see that this bound on the running time applies to Greedy Merge DCC if we can in constant time both decide the feasibility of a merge and compute the intercluster sparsity of the resulting clustering. The former property can be cast into the following definition.

Definition 4.16. An intracluster density measure i is *efficiently decidable*, if for each threshold $\alpha \geq 0$ and for each clustering $\mathcal{C}$ with $i(\mathcal{C}) \geq \alpha$ containing a merge M, $i(\mathcal{C}_M) \geq \alpha$ can be decided in constant time using the information from the cluster graph and other easy to maintain properties of the clustering[4].

Unfortunately, in the context of Density-Constrained Clustering, the value of a clustering after a merge is hard to determine for some notions of intercluster sparsity. Especially in combination with maximum isolated measures, it is not clear how to efficiently update the current objective if we merge clusters A and B and one of these clusters corresponds to a cut with maximum density, expansion or conductance. As we will see in Section 4.4.5, it is however possible to use a relation $\leq_{x,\mathcal{C}}$ to order the merge operations which can be evaluated in constant time. The intuition is to require that $\leq_{x,\mathcal{C}}$ almost behaves like ordering the set of merges by their benefit for the objective function, but permits clever tie-breaking.

Definition 4.17. An intercluster sparsity measure x is *efficiently comparable*, if for each clustering $\mathcal{C}$ there exists a relation $\leq_{x,\mathcal{C}}$ on $\binom{\mathcal{C}}{2} \times \binom{\mathcal{C}}{2}$ such that for $M_1, M_2, M_3 \in \binom{\mathcal{C}}{2}$, it holds that

$$
\begin{aligned}
&M_1 \leq_{x,\mathcal{C}} M_2 \text{ or } M_2 \leq_{x,\mathcal{C}} M_1 && && \text{totality}\\
&M_1 \leq_{x,\mathcal{C}} M_2 \text{ and } M_2 \leq_{x,\mathcal{C}} M_3 &&\implies M_1 \leq_{x,\mathcal{C}} M_3 && \text{transitivity}\\
&M_1 \leq_{x,\mathcal{C}} M_2 &&\implies x(\mathcal{C}_{M_1}) \leq x(\mathcal{C}_{M_2}) && \text{consistency with } x
\end{aligned}
$$

and further $M_1 \leq_{x,\mathcal{C}} M_2$ can be decided in constant time from the information in the cluster graph and other easy to maintain properties of the clustering.

In summary, to achieve $O(n^3)$ running time for Straightforward, it suffices that i is efficiently decidable and x efficiently comparable.

Müllner [166] implements this algorithm in the context of SAHN clustering as a baseline which can be used to assess the improvement obtained by enhanced algorithms. Recall

[4] For gid, this will be the total number of intracluster edges and intracluster pairs and for aid the number of clusters and the current intracluster density value.

that the implicit assumption in [166] is that the distances of a newly merged cluster $A \cup B$ to another cluster C only depend on the size of A, B and C and their pairwise distances. In each iteration, the set of all pairs of clusters is fully traversed, distances are recalculated in case the pair involves the newly merged cluster, and the pair with minimum distance is selected and merged, which can be done in $O(n)$ if the old distances and the size of each cluster are maintained by the algorithm. Hence, for SAHN algorithms according to the update scheme from Müllner, the naïve implementation always has a running time in $O(n^3)$.

In the context of graph clustering, a simple modification of this algorithm is interesting, where in each iteration, only pairs of clusters are considered that are linked by at least one edge. As we have $O(n)$ iterations and in each iteration, at most m pairs of clusters are connected, this leads to a total running time in $O(mn)$. We call this modification Straightforward Sparse. Again, it is easy to see that this modification leads to a correct dendrogram if the intercluster sparsity measure fulfills the following requirement.

Definition 4.18. An intercluster sparsity measure x *enforces connected merges* if for any pair of clusters $C \neq D \in \mathcal{C}$ with $e_{C,D} = 0$ and $x(\mathcal{C}) - x(\mathcal{C}_{\{C,D\}}) > 0$ (i.e., an improving, disconnected merge), there exist clusters $A \neq B \in \mathcal{C}$ such that $e_{A,B} > 0$ and $x(\mathcal{C}_{\{C,D\}}) > x(\mathcal{C}_{\{A,B\}})$ (i.e., a better, connected merge).

Nearest Neighbor List. Anderberg [11] discusses a modification of the naïve implementation called Nearest Neighbor List, where each object or cluster always stores its closest neighbor. When two clusters A and B are merged, the distances of the resulting cluster to all other clusters has to be computed and it has to be checked if $A \cup B$ is the closest neighbor of some other clusters. Furthermore, for each cluster C that had A or B as closest neighbor before, the closest neighbor has to be recomputed from scratch. This leads to a worst case running time in $O(n^3)$ and a best case running time in $O(n^2)$ for SAHN algorithms following the update scheme described by Müllner. The main benefit of this algorithm compared to competing approaches with better runtime guarantees is that it only needs memory that is linear in the current number of clusters. Furthermore, it is very easy to implement and has little constant overhead. Müllner [166] proposes two modifications of Anderberg's approach in order to make it more efficient in practice. The first modification is to store the nearest neighbors of all vertices in a priority queue in order to find the minimum in each iteration faster than by the linear search of the base algorithm. Second, if A was the closest neighbor of some cluster C prior to merging A and B and $d(A, C) < d(A \cup B, C)$, the algorithm sets an invalid flag for the corresponding entry in the queue but uses as key the old distance $d(A, C)$. This is guaranteed to be a lower bound on the distance to the actual closest neighbor. Whenever an invalid entry is detected as the minimum of the queue, the closest neighbor of the represented cluster is recomputed from scratch and the key of the entry is updated. The latter heuristic has also been applied by Cardinal and Eppstein [45] and by Kaukoranta et al. [133] in the context of vector quantization. Müllner's experiments confirm that this modified algorithm compares very well to alternative SAHN algorithms [166] and to widely used library implementations [167] with respect to centroid and median linkage. Kriege et al. [139] propose a heuristic modification of Müllner's algorithm that works for metric distance functions by using approximate nearest neighbor queries.

Nearest Neighbor List builds upon the property of SAHN algorithms that the distance between two clusters never changes if some of the remaining clusters are merged. Hence, the closest neighbor of a cluster only has to be recalculated if the previously

closest neighbor disappears by a merge and the new cluster has a larger distance than the old closest neighbor. The same property guarantees that the priority queue used by Müllner to store all nearest neighbors always yields a pair of clusters with minimum distance.

If we want to apply Anderberg's algorithm to implement Greedy Merge DCC, the intercluster sparsity measure x has to fulfill certain requirements. Intuitively, x should allow us to decide, without knowledge about the remainder of the clustering, which of two given merges is more beneficial to it. If the benefit exclusively depends on the participating clusters, as for nxe alone, this is immediate. However, as above, for maximum isolated measures, a merge can be non-improving at some point of the algorithm and then become improving again. As in the definition of efficient comparability, we make use of a relation $\leq_x$ that serves as a comparator, but this time, the relation is independent from the remainder of the clustering. To serve as a comparator in a priority queue, $\leq_x$ should closely resemble a total preorder on the set of all possible merges.[5] Informally, we call an objective function *local*, if it admits such a relation. More formally, we make the following definition. Let us denote by $\mathcal{M}$ the set of all possible merges, i.e., the set of all unordered pairs of subsets of V.

Definition 4.19. An intercluster sparsity measure x is *local*, if there exists a relation $\leq_x$ on $\mathcal{M} \times \mathcal{M}$ such that for any clustering $\mathcal{C}$ and for all M_1, M_2, M_3 in $\mathcal{M}$ with $M_1 \cup M_2 \cup M_3 \subseteq \mathcal{C}$, the following holds:

$$
\begin{aligned}
&M_1 \leq_x M_2 \text{ or } M_2 \leq_x M_1 && && \approx \text{totality} \\
&M_1 \leq_x M_2 \text{ and } M_2 \leq_x M_3 \implies && M_1 \leq_x M_3 && \text{transitivity} \\
&M_1 \leq_x M_2 && \implies x(\mathcal{C}_{M_1}) \leq x(\mathcal{C}_{M_2}) && \text{consistency with } x
\end{aligned}
$$

If $M_1 \leq_x M_2$ can additionally be decided in constant time using the information from the cluster graph and other easy to maintain properties of the clustering, x is called *efficiently locally comparable*.

It is not hard to see that if x is efficiently locally comparable, it is also efficiently comparable, as we can substitute $\leq_x$ for $\leq_{x,\mathcal{C}}$ for any clustering $\mathcal{C}$.

Using constraints potentially impedes quick agglomeration, as we do not only have to determine the merge that improves the objective function the most, but have to restrict ourselves to the set of feasible merges. If the feasibility of a merge is independent of the remainder of the clustering, this does not incur a penalty with respect to running time. More formally, the intracluster measure used should exhibit the following property.

Definition 4.20. An intracluster measure i is *context insensitive*, if for each merge M and each threshold value α, either all clusterings $\mathcal{C}$ with $i(\mathcal{C}) \geq \alpha$ and $M \subseteq \mathcal{C}$ fulfill $i(\mathcal{C}_M) \geq \alpha$ or none of them. If i does not have this property, we call it *context sensitive*.

Together with the locality of the objective function, context insensitivity ensures that the closest feasible neighbor for each cluster only changes if the previous closest feasible neighbor is one of the clusters that are merged in the current iteration or the new closest neighbor corresponds to their union. In Müllner's modification, for each cluster that has no feasible neighbor, no entry has to be stored in the priority queue. Altogether,

[5] We do not need proper totality, as we never have to compare pairs of merges that cannot coexist in a clustering, e.g., because the clusters considered intersect.

Anderberg's algorithm can be applied to Greedy Merge DCC if the intercluster sparsity measure x is local and the intracluster density measure i context insensitive. If furthermore x is efficiently locally comparable and i efficiently decidable, it has a worst case running time in $O(n^3)$. In case x enforces connected merges, in the worst case, every cluster has to update its closest neighbor by considering all clusters linked with it, leading to a total running time in $O(nm)$ for all iterations. We call this modification Nearest Neighbor List Sparse.

Heap and Quadtree. Day and Edelsbrunner [59] describe the straightforward approach to store the distances in one binary heap per vertex, which leads to a worst case running time in $O(n^2 \log n)$. Independently, Kurita [143] proposes a very similar approach replacing the n binary heaps by one single heap storing all distances. Eppstein [70] replaces the binary heap by a quadtree, which yields the best algorithm for general SAHN linkage schemes with respect to worst case running time; it runs in $O(n^2)$. Both algorithms work for arbitrary distance functions and have been discussed in detail in Section 3.3.1 in the context of modularity maximization. The downside of these algorithms is that they require space that is quadratic in the number of objects to be clustered. However, in case the distance function is explicitly given and cannot be computed on demand, the size of the input is in $\Theta(n^2)$ anyhow and not much is lost. Some instantiations of SAHN allow for a reduced memory complexity, as for example it is known in advance that some cluster pairs will never be merged.

It is not hard to see that both the heap and quadtree based algorithms can be applied to Greedy Merge DCC if the intercluster sparsity measure x is local and the intracluster density measure i context insensitive. If x is furthermore efficiently locally comparable and i efficiently decidable, this leads to a running time in $O(n^2 \log n)$ for the heap based variants and in $O(n^2)$ for the approach based on a quadtree. Similar to Müllner's variant of the Nearest Neighbor List algorithm, if x enforces connected merges, it suffices to maintain only cluster pairs that are connected by at least one edge in the heap, reducing the space requirement to $O(n + m)$. As explained in Section 3.3.1 in the context of modularity maximization, an alternative runtime analysis then yields a running time in $O(md \log n)$, where d is the height of the dendrogram. In case the dendrogram is well balanced, this bound is better than the trivial upper bound. We call this variant Heap Sparse. It might be possible to exploit the fact that the objective enforces connected merges also in combination with the quadtree based approach; as this is non trivial, we leave it out of consideration.

Conga Line and variants. Eppstein [70] further describes a data structure that only needs $O(n)$ space and can be used to find the closest pair of a set of n objects in $O(n)$ time. He shows that deletions can be handled in $O(n \log^2 n)$ and insertions in $O(n \log n)$ amortized time, yielding a time complexity in $O(n^2 \log^2 n)$ for general SAHN algorithms. The idea behind this data structure is to maintain a linear number of edges distributed over a logarithmic number of graphs over the set of objects. At any time, it is guaranteed that one of these edges corresponds to a globally closest pair of objects. The resulting SAHN algorithm Conga Line and two slightly modified version with worse worst case complexity but better practical running time compete well with other SAHN algorithms in an experimental evaluation [70]. A follow-up work by Cardinal and Eppstein [45] includes another heuristic modification of conga line that is based on *lazy deletion*, similar to the modifications of Anderberg's algorithm.

To give more details, the data structure used in Conga Line consists of a partition of the set of objects into a logarithmic number of subsets together with a set of directed

paths associated with each subset. For each of these sets S, exactly one path is initially constructed such that every second vertex has to be an object in S and the successor of an object is always its closest neighbor adhering to this restriction. Object deletions and insertions result in the deletion of some edges on this path, therefore splitting the path in several subpaths, and the creation of new and merging of old subsets. Hereby, the invariant that one of the edges in the remaining paths corresponds to a globally closest pair of subsets always remains valid. Eppstein states that with this data structure, bottom-up clustering methods in which clusters are represented as objects and the distances between clusters can be computed in constant time can be performed in time $O(n^2 \log^2 n)$ and space $O(n)$ [70]. Similar to other SAHN algorithms, the algorithm relies on the property that the distance between two clusters is independent from the remainder of the clustering. However, as for the Nearest Neighbor List approach, it is sufficient that the closest feasible neighbor of a cluster remains the same if it does not involve the clusters merged in the current iteration. In the context of Greedy Merge DCC, this is guaranteed if x is local and i context insensitive. To achieve the running time above, again, x must be efficiently locally comparable and i efficiently decidable. Compared to Heap Sparse, which also needs only linear additional space, the worst case running time of Conga Line is slightly worse, but the objective does not have to enforce connected moves.

Nearest Neighbor Chaining. Besides algorithms for general SAHN linkage schemes, there exists a plethora of algorithms that can either be used for distance functions fulfilling certain requirements [26, 127, 169] or are bound to a single distance function or linkage scheme [60, 105, 185, 201]. An elegant algorithm that is still fairly general is the *nearest-neighbor chain* algorithm developed and implemented by Benzécri [26] and Juan [127]. The algorithm iteratively builds a path of clusters such that the cluster at the $i+1$-th position is the closest neighbor of the cluster at position i. As soon as a circle of size 2 is found, i.e., the clusters at positions i and $i+1$ are mutual closest neighbors, these clusters are merged and deleted from the path and the process continues from position $i-1$. The algorithm is described in detail by Murthag [169], who also states that the algorithm is correct as long as the distance function d has the *reducibility property*, i.e.,

$$d(A,B) \leq \min\{d(A,C), d(B,C)\} \Rightarrow \min\{d(A,C), d(B,C)\} \leq d(A \cup B, C)\}.$$

Müllner observes that the structure of the resulting dendrogram interpreted as a binary tree is correct, but the order of merges may differ from the output of other SAHN algorithms. This can be easily solved by a postprocessing step, which orders all merge operations according to increasing distances. He also remarks that in the most general setting for SAHN algorithms, where the new distances are allowed to depend arbitrarily on the old distances involving the merged clusters, it is further required that the distance $d(A \cup B, C \cup D)$ is independent of whether A and B or C and D are merged first.[6] The same observation has been made previously by Gronau and Moran [107].

The worst case complexity of this algorithm is in $O(n^2)$ [169] and it is easy to see that the additional space to maintain the list of nearest neighbors is in $O(n)$. It is hence the fastest of the algorithms described in this section that only needs linear space.

Again, we can transfer the requirements of this method to obtain a concrete algorithm implementing Greedy Merge DCC. The idea behind the above reducibility property

[6] This usually holds, as the distance only depends on some properties of the clusters themselves.

is that merging the closest pair of clusters A and B does not change the closest neighbor $D \neq A, B$ of a cluster C. A look at Müllner's correctness proof [166] of the Nearest Neighbor Chain algorithm reveals that this is indeed all that is required. In the context of Generic Agglomeration With Constraints, this translates to the following property[7]:

Definition 4.21. An objective function f on clusterings is *reducible*, if it is efficiently locally comparable and the relation $\leq_x$ from Definition 4.19 satisfies the following property. Let A, B, C, D be mutually disjoint subsets of V and $\{A, B\} \leq_x \{A, C\} \leq_x \{B, C\}$. Then,

$$\{D, C\} \leq_x \{A, C\} \Rightarrow \{D, C\} \leq_x \{A \cup B, C\}.$$

In summary, Nearest Neighbor Chaining is correct and has a running time in $O(n^2)$, if x is efficiently locally comparable and reducible and i is efficiently decidable and context insensitive.

4.4.4 Properties of Intracluster Measures

In this section, we classify all intracluster density measures from Section 4.1 with respect to their impact on the running time of Greedy Merge DCC, i.e., with respect to the question whether they are context insensitive and efficiently decidable.

Context Insensitivity

As discussed above, using constraints potentially impedes quick agglomeration, as it does not suffice to determine the merge that improves the objective function the most. The good news is that if we use mid as a constraint, the feasibility of a merge only depends on the density of the merged cluster, which is clearly independent of the remainder of the clustering. This leads to the following simple corollary.

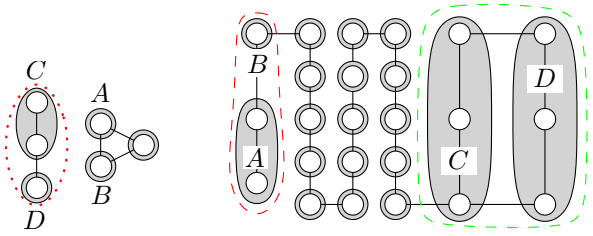
Corollary 4.22. Mid *is context insensitive.*

In contrast to mid, gid and aid are context sensitive, as the status of a merge can change from allowed to disallowed and back again. In Figure 4.6(a), starting from the gray clustering, merging the path to the left is not allowed if the constraint $\mathsf{gid}(\mathcal{C}) \geq 0.7$ is used. If singletons C and D are merged, this is allowed again, as the number of intracluster pairs increases.

Figure 4.6 shows a similar example for aid, which we are going to explain in more detail. Suppose that we use nxe as objective function, aid as constraint and an appropriate value of α. Starting with singletons, let us begin with a merge that yields cluster A. Merging A and B will decrease aid, any further decrease in the number of clusters will worsen this effect. Suppose we reach clustering $\mathcal{C}$ (gray), we even have $\mathsf{aid}(\mathcal{C}_{\{A,B\}}) < \mathsf{aid}(\mathcal{C}_{\{C,D\},\{A,B\}})$, as two "bad" clusters are summarized into a single one, which mildly lifts aid. Thus, if we set α to the average of these two values, the merge $\{A, B\}$ is infeasible in $\mathcal{C}$ but as soon as the merge $\{C, D\}$ is performed, regains feasibility. Obviously, it again loses feasibility as soon as we continue merging other singletons.

Corollary 4.23. Gid *and* aid *are context sensitive.*

[7] Requiring that $\{A, C\} \leq_x \{A \cup B\}$ seems more straightforward but is not suitable in our context; according to our definition of locality, these two merges do not have to be comparable.



(a) gid (b) aid

FIGURE 4.6: Counterexamples showing that gid and aid are context sensitive. (a) If constraint gid ≥ 0.7 is used, merging A and B in the gray base clustering is infeasible, but feasible after merging C and D. (b) If constraint aid ≥ 0.945 is used, merging A and B in the gray base clustering is infeasible, but feasible after merging C and D.

Efficient Decidability

To justify why all our notions of intracluster density are efficiently decidable, we have to show how the feasibility of a merge can be decided in constant time. Starting from a feasible clustering, using mid, the clustering remains feasible after merging clusters A and B if and only if $\mathsf{id}(A \cup B) \geq \alpha$. Hence, mid is efficiently decidable as $\mathsf{id}(A \cup B)$ can be easily computed with the information from the cluster graph:

$$\mathsf{id}(A \cup B) = \frac{e_A + e_B + e_{A,B}}{\binom{n_A + n_B}{2}}$$

For gid, we additionally keep track of the total number of intracluster edges $i_e(\mathcal{C}) := \sum_{C \in \mathcal{C}} e_C$ and the total number of intracluster pairs $i_p(\mathcal{C}) := \sum_{C \in \mathcal{C}} \binom{n_C}{2}$. With that, we can easily compute the intracluster density of the clustering after the merge and thereby its feasibility by

$$\mathsf{gid}(\mathcal{C}_{A,B}) = \frac{i_e(\mathcal{C}) + e_{A,B}}{i_p(\mathcal{C}) + n_A \cdot n_B}$$

Finally, the new value $\mathsf{aid}(\mathcal{C}_{A,B})$ can be computed by the following equation if we always keep track of the current value $\mathsf{aid}(\mathcal{C})$:

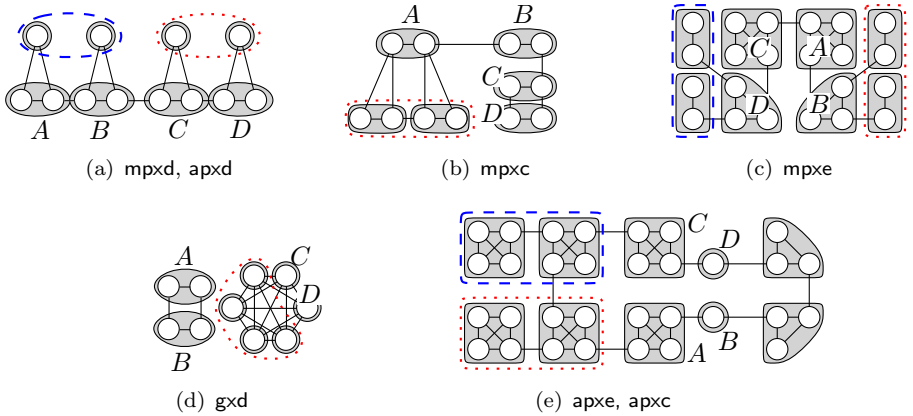
$$\mathsf{aid}(\mathcal{C}_{A,B}) = \frac{|\mathcal{C}|}{|\mathcal{C}|+1} \cdot \mathsf{aid}(\mathcal{C}) + \frac{1}{|\mathcal{C}|+1} \cdot \left(\underbrace{\frac{e_A + e_B + e_{A,B}}{\binom{n_a + n_B}{2}}}_{=\mathsf{id}(A \cup B)} - \underbrace{\frac{e_A}{\binom{n_a}{2}}}_{=\mathsf{id}(A)} - \underbrace{\frac{e_B}{\binom{n_B}{2}}}_{=\mathsf{id}(B)} \right)$$

Corollary 4.24. Mid, gid *and* aid *are efficiently decidable.*

4.4.5 Properties of Intercluster Measures

Locality and Efficient Comparability

In this section, we will state sufficient conditions for both non-locality (Lemma 4.25) and locality (Lemmata 4.27, 4.29), thereby clarifying locality for all our objective functions.



(a) mpxd, apxd (b) mpxc (c) mpxe

(d) gxd (e) apxe, apxc

FIGURE 4.7: Locality counterexamples: The base clustering consists of the gray clusters. In (a), (c), and (e), if the blue, dashed merge is performed, merge $\{C, D\}$ is better, if the red, dotted merge is performed, $\{A, B\}$ is better. In (b) and (d), in the base clustering, $\{C, D\}$ is better than $\{A, B\}$; if the red, dotted merge is performed, the opposite holds.

Lemma 4.25. *Let x be an intercluster sparsity measure. If there exists a graph with two clusterings $\mathcal{C}$ and $\mathcal{D}$ both containing two merges M_1 and M_2 such that $x(\mathcal{C}_{M_1}) < x(\mathcal{C}_{M_2})$ and $x(\mathcal{D}_{M_1}) > x(\mathcal{D}_{M_2})$, then x is not local.*

Proof. Assume x is local, let $\leq_x$ be a relation on $\mathcal{M} \times \mathcal{M}$ satisfying the definition of locality. From $x(\mathcal{C}_{M_1}) < x(\mathcal{C}_{M_2})$ it follows that $M_2 \leq_x M_1$ does not hold and $x(\mathcal{D}_{M_1}) > x(\mathcal{D}_{M_2})$ implies that $M_1 \leq_x M_2$ does not hold. This means that M_1 and M_2 are not comparable, contradicting the assumption on $\leq_x$. □

For mpxd, apxd, mpxc, mpxe and gxd, Figure 4.7 shows examples where the assumption of Lemma 4.25 holds, which implies that these measures are not local. The rough idea behind the examples in Figures 4.7(a)-4.7(c) is that the pairwise intercluster quality between two clusters can be improved or deteriorated by merging one of the partners with a third cluster. Figure 4.7(d) exploits that merging large clusters becomes more attractive if the global intercluster density is low. An instance proving the nonlocality of apxe and apxc is given in Figure 4.7(e).

Corollary 4.26. Mpxd, apxd, mpxc, apxc, mpxe, apxe *and* gxd *are not local.*

For nxe, it is easy to see that choosing $\leq_x$ such that $\{A, B\} \leq_x \{C, D\}$ is given by $e_{A,B} \geq e_{C,D}$ satisfies the definition of locality. The remaining objectives follow the isolated view on graph clusters and can be split into maximum and average measures. We start by a general lemma about maximum isolated measures.

Lemma 4.27. *Let x be an intercluster sparsity measure such that $x(\mathcal{C})$ can be expressed as $x(\mathcal{C}) = \max_{C \in \mathcal{C}} x'(C)$, with $x'(C)$ solely depending on C. Then, x is local.*

Proof. We prove the claim constructively by defining a relation $\leq_x$ and showing that it fulfills the required properties of the definition of locality.

Let $\mathcal{C}$ be a clustering and $\mathcal{D} = \{D_1, \ldots, D_k\} \subseteq \mathcal{C}$. We define $L(\mathcal{D}) := (x'(D_{i_1}) \geq \ldots \geq x'(D_{i_k}))$ to be the non-increasing sequence of values of the $D_i \in \mathcal{D}$. Let $\leq_\ell$ be the

lexicographical order on sequences of rational numbers. The first observation we make is that for arbitrary merges M_1 and $M_2 \subseteq \mathcal{C}$, $L(\mathcal{C}_{M_1}) \leq_\ell L(\mathcal{C}_{M_2})$ implies $x(\mathcal{C}_{M_1}) \leq x(\mathcal{C}_{M_2})$. Since the lexicographical order on sequences of rational numbers is a partial order, $\leq_\ell$ is transitive and reflexive. If we can find an equivalent relation $\leq_x$ on $\mathcal{M} \times \mathcal{M}$, i.e., a relation such that $M_1 \leq_x M_2$ iff $L(\mathcal{C}_{M_1}) \leq_\ell L(\mathcal{C}_{M_2})$, independent of $\mathcal{C} \setminus (M_1 \cup M_2)$, this yields the locality of x.

The crucial idea is to consider just that part of $L(\mathcal{C})$ which changes if either of the two merges $M_1 = \{A, B\}$ and $M_2 = \{C, D\}$ is performed. The affected entries of $L(\mathcal{C})$ are $\{x'(A), x'(B), x'(C), x'(D)\}$. Performing M_1 replaces these by $\{x'(A \cup B), x'(C), x'(D)\}$, and performing M_2 replaces them by $\{x'(A), x'(B), x'(C \cup D)\}$. Since the remaining part of the sequence, $L(\mathcal{C} \setminus \{x'(A), x'(B), x'(C), x'(D)\})$, remains unchanged in either case, it suffices to compare the two replacements $L(\{x'(A \cup B), x'(C), x'(D)\})$ and $L(\{x'(A), x'(B), x'(C \cup D)\})$ instead of $L(\mathcal{C}_{M_1})$ and $L(\mathcal{C}_{M_2})$. Thus, if we define $\leq_x$ such that $\{A, B\} \leq_x \{C, D\}$ iff $L(\{x'(A \cup B), x'(C), x'(D)\}) \leq_\ell L(\{x'(C \cup D), x'(A), x'(B)\})$, $\leq_x$ is a relation that satisfies the definition of locality. □

Corollary 4.28. Mixd, mixc *and* mixe *are local.*

A similar lemma gives us the locality of average isolated measures.

Lemma 4.29. *Let x be an intercluster sparsity measure such that $x(\mathcal{C})$ can be expressed as $x(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{C \in \mathcal{C}} x'(C)$, with $x'(C)$ solely depending on C. Then, x is local.*

Proof. Let $\mathcal{C}$ be an arbitrary clustering containing four clusters A, B, C and D. Since

$$x(\mathcal{C}_{A,B}) \cdot (|\mathcal{C}| - 1) = |\mathcal{C}| \cdot x(\mathcal{C}) - x'(A) - x'(B) + x'(A \cup B),$$

$x(\mathcal{C}_{A,B}) \leq x(\mathcal{C}_{C,D})$ is equivalent to

$$\underbrace{x'(A \cup B) - x'(A) - x'(B)}_{:= k_{A,B}} \leq \underbrace{x'(C \cup D) - x'(C) - x'(D)}_{:= k_{C,D}}$$

As for each cluster, x' is independent of the remainder of $\mathcal{C}$, this inequality shows that $\leq_x := \{(\{A, B\}, \{C, D\}) \mid k_{A,B} \leq k_{C,D}\}$ is a preorder on $\mathcal{M} \times \mathcal{M}$ such that $\{A, B\} \leq_x \{C, D\}$ implies $x(\mathcal{C}_{A,B}) \leq x(\mathcal{C}_{C,D})$. Thus, x is local. □

Corollary 4.30. Aixd, aixc *and* aixe *are local.*

We have now proven or disproven the locality of all intercluster measures (for a summary see Tab. 4.3). Note that all proofs of locality are constructive in that they induce comparators which can for example be used to efficiently maintain the set of all possible merges considered by Alg. 5 in a priority queue. For the maximum functions, triples of real numbers can be used as keys, with comparators as described in the proof of Lemma 4.27. When using averages, it suffices to store the values $k_{A,B}$ defined in the proof of Lemma 4.29. Similar to the arguments in Section 4.4.4, it can be verified that the expansion, conductance and density of two clusters A and B and their union $A \cup B$ can be computed in constant time using the information maintained in the cluster graph. Hence, all keys can be computed and compared in constant time, which yields the following corollary.

Corollary 4.31. Mixd, mixc, mixe, aixd, aixc, aixe *and* nxe *are efficiently locally comparable.*

As discussed in Section 4.4.3, this automatically entails that these measures are also efficiently comparable. Global intercluster density is not local, it is however efficiently decidable by the following simple argument. If we maintain the total number of intercluster edges $\mathsf{nxe}(\mathcal{C})$ and intercluster pairs $x_p(\mathcal{C}) := \sum_{A \neq B \in \mathcal{C}} n_a n_B$ during the algorithm, the intercluster sparsity of the clustering after merging clusters A and B can be easily computed by

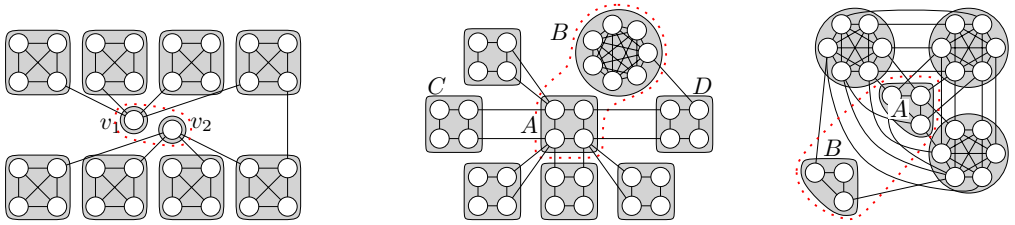
$$\mathsf{gxd}(\mathcal{C}_{A,B}) = \frac{\mathsf{nxe}(\mathcal{C}) - e_{A,B}}{x_p(\mathcal{C}) - n_A n_B}$$

Corollary 4.32. Gxd *is efficiently comparable.*

In contrast to that, using pairwise intercluster sparsity measures complicates the task to efficiently compare merge operations. Roughly speaking, merging two clusters A and B removes the contribution of $O(n)$ pairwise cuts from the objective and inserts $O(n)$ new cuts between $A \cup B$ and the remaining clusters. With the information stored in the cluster graph, the value of each of these cuts can be determined in constant time, which leads to a time complexity in $O(n)$ for the computation of the new objective and hence for the comparison of two merge operations. With this naïve approach, Straightforward has a time complexity in $O(n^4)$. We do not see a way to circumvent this problem, but have no formal proof that pairwise intercluster sparsity measures are not efficiently comparable. Hence, we leave the status of the remaining measures with respect to efficient comparability open.

Disconnected Merges

Whenever no single edge links two clusters, intuitively, merging them should not be beneficial to the clustering, or at least, such a merge should not be the best option. In the light of our bicriterial approach, an objective function which does encourage such a *disconnected merge* is naturally opposed by the separate mechanism of a constraint on the intracluster density. Superficially, this resolves the issue for non-degenerate instances; however, a more accurate assertion is algorithmically relevant: If we can rule out disconnected merges, this translates to algorithms that are more time and space efficient (cf. Section 4.4.3). In the following, we answer the question whether our objective functions enforce connected merges. In fact, only nxe and gxd have this property (see Tab. 4.3). Both measures never benefit at all from disconnected merges: The former does not change, and the latter even deteriorates. The circumstances under which all other measures potentially encourage disconnected merges are intuitively illustrated in Figure 4.8. If most clusters are reasonable, merging two clusters with a particularly ill contribution to the measure can be the best option (Figure 4.8(a)). For all pairwise measures, this is immediate, and it is also not hard to see for all average measures, as, roughly speaking, the number of bad contributors decreases. While the above arguments fail for mixc, mixe and mixd, a disconnected merge of a bad cluster with a very good one can be the best option for them (see Figure 4.8(b) and Figure 4.8(c)). Note that it is always possible to artificially restrict the set of allowed merges to connected ones, yielding a modified greedy algorithm; we chose this approach in the experiments in Chapter 5. We summarize our observations in the following corollary.



(a) apxc, apxe, apxd, aixc, aixe, aixd, mpxc, mpxe, mpxd (b) mixc, mixe (c) mixd

FIGURE 4.8: Given the gray clusterings, disconnected merges (red, dotted) yield the highest improvement for the objective functions pointed out. Thus, all objective functions, except nxe and gxd, potentially favor disconnected merges.

Corollary 4.33. *Of the intercluster sparsity measures introduced in Section 4.1.1, only* nxe *and* gxd *enforce connected merges.*

Reducibility

By definition, intercluster sparsity measures that are not local cannot be reducible, leaving nxe, mixd, mixc, mixe, aixd, aixc, and mixe as potential candidates for reducibility. Unlike locality, we will not obtain a complete classification of these measures with respect to reducibility, but show that some of them are not reducible and shortly discuss why constructing counterexamples for some of the remaining measures turns out to be difficult.

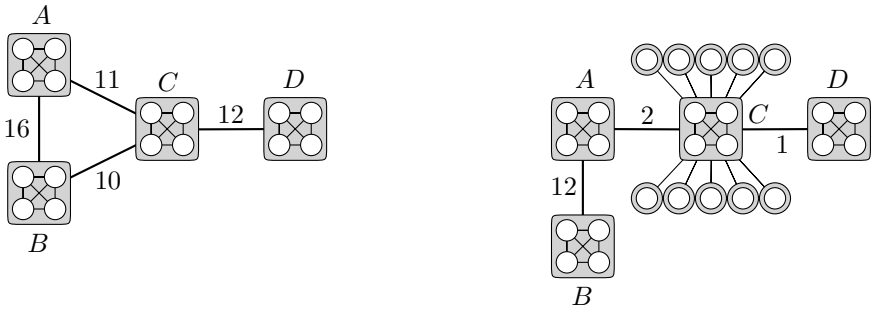
Similarly to locality, we start with a simple lemma giving sufficient conditions for nonreducibility that can be used to construct counterexamples.

Lemma 4.34. *Let x be an intercluster sparsity measure. If there exists a graph with a clustering $\mathcal{C}$ containing clusters A, B, C and D such that $x(\mathcal{C}_{A,B}) < x(\mathcal{C}_{A,C}) < x(\mathcal{C}_{B,C})$, $x(\mathcal{C}_{D,C}) < x(\mathcal{C}_{A,C})$, and $x((\mathcal{C}_{A,B})_{D,C}) > x((\mathcal{C}_{A,B})_{A\cup B,C})$, then x is not irreducible.*

Proof. Assume $\leq_x$ is a relation satisfying the properties from Definition 4.19. The consistency of $\leq_x$ with x immediately yields that $\leq_x$ does not fulfill the requirements of Definition 4.21. □

We start with a simple counterexample showing that nxe is not reducible (cf. Figure 4.9(a)). Clearly, merging A and B removes more intercluster edges than merging A and C and merging A and C is better than merging A and B. In the original clustering, D is strictly the best merge partner of C, as it shares the most edges with C. However, if A and B are merged, merging $A \cup B$ with C is strictly better than merging D with C. Thus, Lemma 4.34 tells us that nxe is not reducible.

The counterexample for mixd in Figure 4.9(b) is slightly more complicated. It is constructed such that both merging A and C and merging B and C produce a cluster that is worse than the current maximum value. In contrast to that, merging D and C leaves the objective function unchanged as the current worst contributor A stays untouched. However, if we first merge clusters A and B, the resulting cluster is much better than both A and B. This is why merging $A \cup B$ with the new worst contributor C is now able to reduce its value by a larger amount than merging C with D.



(a) nxe (b) mixd

FIGURE 4.9: Examples showing that nxe and mixd are not reducible. Heavier edges between the clusters summarize the number of edges between them.

Constructing a similar example for mixc and mixe turns out to be more difficult, as Observation 2 tells us that $A \cup B$ can be worse than A and B only if $n_A + n_B > n/2$ or $v_A + v_B > m$, respectively. Hence, to use a similar idea as above, it is necessary to exploit this border case repeatedly. We can even show that every counterexample for these measures using Lemma 4.34 has to depend on the case distinction in the denominator.

Observation 4. *Let x be an intercluster sparsity measure such that $x(\mathcal{C})$ can be expressed as $x(\mathcal{C}) = \max_{C \in \mathcal{C}} x'(C)$, with $x'(C)$ solely depending on C such that for all $A, B \in \mathcal{C}$, $x'(A \cup B) \leq \max\{x'(A), x'(B)\}$. Then, x cannot fulfill the conditions in Lemma 4.34.*

Proof. Let us assume that x fulfills the conditions in Lemma 4.34. From $x\big((\mathcal{C}_{A,B})_{D,C}\big) > x\big((\mathcal{C}_{A,B})_{A\cup B,C}\big)$, it follows that

$$\max\{x'(A \cup B), x'(D \cup C)\} > \max\{x'(A \cup B \cup C), x'(D)\}.$$

As, by assumption, $x'(D \cup C) \leq x'(D)$, we see that $x'(A \cup B)$ must be strictly larger than $x'(D)$, which in turn implies that $x'(A) > x'(D)$. On the other hand, from $x(\mathcal{C}_{D,C}) < x(\mathcal{C}_{A,C})$, it follows that $\max\{x'(D \cup C), x'(A)\} < \max\{x'(A \cup C), x'(D)\}$. Again, by assumption, $x'(A \cup C) \leq x'(A)$ and hence $x'(D) > x'(A)$, which leads to a contradiction. □

We therefore leave the reducibility of mixe and mixc, as well as the reducibility of all average isolated measures open. The following corollary summarizes the results from this section.

Corollary 4.35. Nxe *and* mixd *are not reducible.*

4.5 Concluding Remarks

Established measures for graph cuts lend themselves well for precisely expressing desiderata on graph clusterings. With a focus on finding graph clusterings that feature guaranteed intra- and high intercluster quality, we revived this approach and systematically formalized bicriteria quality measures based on the expansion, conductance and density of the cuts induced by the clusters.

The classification of these measures with respect to their behavior in the context of greedy agglomeration yields conditions that permit the use of efficient algorithms for SAHN clustering, incorporating constraints on the intracluster density of the clustering. Beyond the concrete question which combinations of intercluster sparsity and intracluster density measures render the use of SAHN clustering algorithms possible, this classification is interesting for two reasons. First, the use of greedy agglomeration as a straightforward and conceptually easy heuristic to solve variants of clustering and partitioning problems suggests itself; a well-known example is the CPM algorithm [53], which proved to be the first practical algorithm for modularity based clustering. The identification of key properties of objective functions and constraints summarized in Table 4.5 is independent of the measures considered in this chapter and can therefore be used as a guideline to obtain efficient implementations of greedy agglomeration algorithms for other problems in the context of graph clustering and graph partitioning. Second, some of these properties, especially locality and the question whether an intercluster sparsity measure enforces connected merges yield also qualitative insights into the measures themselves and might prove useful in other context, e.g., for the computation of optimal solutions.

We furthermore showed that most definitions of intercluster sparsity and intracluster density do not suffer from local minima in the context of greedy agglomeration, which confirms our intuition that the former strive towards coarse and the latter towards fine clusterings. Interestingly, although measures building upon the cut between pairs of clusters seem most suitable for agglomerative algorithms at first glance, most of them are prone to local minima, as well as the intuitive definition of intracluster density averaging the density of all clusters. Note that the properties unbounded and merge consistent carry over to other local search approaches that include the possibility to merge clusters, as for example the multilevel algorithm building upon vertex moves that is considered in Chapter 5. An overview of the classification of the considered measures with respect to these properties can be found in Table 4.3 and Table 4.4. As a small example, we illustrate the outcome of greedy agglomeration combining guaranteed intracluster density with high average isolated intercluster conductance on the email network of our department in Figure 4.10.

An experimental evaluation of density-constrained graph clustering and the adaption to local greedy optimization is subject to Chapter 5. Although the complexity of the problems considered in this chapter probably does not permit obtaining optimal solutions for large instances with good theoretical time bounds, it would be nonetheless interesting to compute optimal solutions for small graphs to investigate the influence the different measures have on the resulting clustering. An obvious approach for that is the use of sophisticated solvers for linear programming, as for example CPLEX [1] or gurobi [114]. Appendix A contains building blocks that can in principle be used to model all instantiations of Density-Constrained Clustering. Some of these however require prohibitively large sets of variables and constraints, which is why we did not use them in a systematic study; an example of an optimal clustering computed with this approach is contained in Figure 4.2. To obtain optimal solutions for interesting graph sizes, it is necessary to use other ideas, probably tailored to particular combinations of intercluster sparsity and intracluster density measures. A promising approach for this might be the use of column generation algorithms [10, 126, 122].

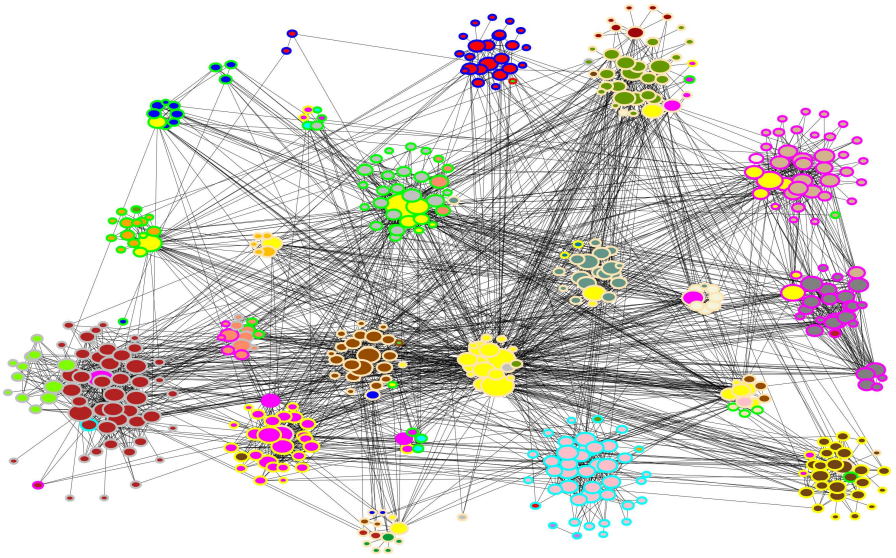


FIGURE 4.10: This graph is a three-month snapshot of the email traffic at KIT's CS department, groups represent chairs, which serve as a ground truth (vertices are scaled by degree, $n = 472, m = 2845$). We ran Alg.5 using mid with $\alpha = 0.25$ and aixc to arrive at the color-clustering. Border colors indicate a Modularity-based clustering [53].

Chapter 5

Experiments on Density-Constrained Graph Clustering

In Chapter 4, we systematically assembled a range of self-evident intracluster density and intercluster sparsity measures for clusterings, where the latter are based on conductance, expansion and density of the cuts induced by the clusters. We further formally stated the problem Density-Constrained Clustering (DCC), where the objective is to optimize intercluster sparsity with the constraint that the intracluster density must exceed a given threshold. As optimal polynomial-time algorithms for DCC are unknown, we investigated how different combinations of intercluster sparsity and intracluster density measure influence the efficiency of a greedy optimization strategy based on cluster merging. However, little is known about its qualitative behavior in practical scenarios, and an experimental evaluation of DCC is still missing. To some extent, this chapter aims at closing this gap.

Contribution. In this chapter, we provide a comprehensive study of the practical behavior of greedy graph clustering heuristics driven by cut-based objectives and constrained by intracluster density. We give evidence that, in general, greedy algorithms based on local vertex moves lead to better quality than the corresponding merge-based algorithm. We then compare the move-based algorithm to a set of reference algorithms from the literature, both with respect to the objective of DCC and their ability to reconstruct planted partitions in a family of synthetic graphs. We find that the greedy move algorithm compares favorably to most reference algorithms in the context of DCC, while a comparison with the modularity-based algorithm shows that optimizing modularity implicitly yields good results for some variants of DCC. Our main aim is however the comparison of the clusterings that can be found by optimizing different instantiations of DCC. Experiments with planted partition graphs suggest that certain combinations of inter- and intracluster measures are effective in finding the hidden clustering, while others clearly fail. Together with observations about the number of identified clusters and the skewness of the cluster size distribution, this yields valuable insights about the behavior of the respective intra- and intercluster density measures.

Algorithm 6: Greedy Vertex Moving DCC (GVM)

Input : graph $G = (V, E)$, inter, intra, α
Output: clustering $\mathcal{C}_0$ of G
$G^0 \leftarrow G$, $h \leftarrow 0$
repeat
  $\mathcal{C}^h \leftarrow \mathrm{LM}(G^h, \mathrm{Singletons}(G^h), \mathsf{intra}, \mathsf{inter}, \alpha)$; $G^{h+1} \leftarrow \mathrm{contract}(G^h, \mathcal{C}^h)$
  $h \leftarrow h + 1$
until *no more nontrivial contractions*
while $h \geq 0$ **do**
  $h \leftarrow h - 1$
  $\mathcal{C}^h \leftarrow \mathrm{project}(\mathcal{C}^{h+1}, G^h)$
  $\mathcal{C}^h \leftarrow \mathrm{LM}(G^h, \mathcal{C}^h, \mathsf{inter}, \mathsf{intra}, \alpha)$
return $\mathcal{C}^0$

To keep the number of different configurations managable, we limit ourselves to the evaluation of global and isolated intercluster sparsity measures here, dropping the notions of pairwise intercluster density, conductance, and expansion[1]; we deem these less interesting due to their counterintuitive behavior in some cases and the fact that they are prone to local minima. On the other hand, although it does not quite fit into our classification, we abuse modularity as an intercluster sparsity measure here and include experiments optimizing modularity under constraints on the intracluster density; this yields yet another instance of DCC.

5.1 Greedy Vertex Moving for Density-Constrained Graph Clustering

In Chapter 3, we already discussed a generic greedy algorithm based on moving vertices between clusters, which we termed Generic Local Moving (cf. Algorithm 3 in Section 3.2). Similar to the adaption of Generic Greedy Merge to Density-Constrained Clustering that yielded Greedy Merge DCC(GM), adapting the metaheuristic Generic Local Moving to our problem statement yields Greedy Vertex Moving DCC (GVM) as stated in Algorithm 6.

Its key ingredient is a subprocedure LM that greedily improves intercluster sparsity by letting vertices move to neighboring clusters (Algorithm 7). In contrast to Generic Greedy Vertex Moving, we restrict vertex moves to *feasible moves*, i.e., moves that lead to clusterings that adher to our density constraint. Note that we include here the optional refinement phase that potentially improves the objective by further vertex moves when projecting the clustering resulting from the coarsening phase to lower levels.

As discussed in Chapter 3, Generic Greedy Vertex Moving is closely related to multi-level algorithms in the context of graph partitioning and has previously been used for modularity-based clustering without constraints [32, 189]. We further discussed that in the case of modularity, moving a vertex to a cluster that contains none of its neighbors never improves the objective. Hence, it suffices to consider neighboring clusters. Together with the observation that the change in modularity can be determined

[1] To be consistent with Chapter 4, we keep however the i in the abbreviations, although we do not distinguish between pairwise and isolated measures here.

Algorithm 7: Local Moving (LM)

Input : graph $G = (V, E)$, clustering $\mathcal{C}_{\text{init}}$ of G, inter, intra, α
Output: clustering $\mathcal{C}$ of G
$\mathcal{C} \leftarrow \mathcal{C}_{\text{init}}$
repeat
  forall $v \in V$ **do**
    $\mathcal{A} \leftarrow \{C \in \mathcal{C} \mid \mathsf{intra}(\mathcal{C}_{v \to C}) \geq \alpha \text{ and } |E(v, C)| > 0\}$
    $N \leftarrow \arg\min_{C \in \mathcal{A} \cup \{\}} \{\mathsf{inter}(\mathcal{C}_{v \to C})\}$
    if $\mathsf{inter}(\mathcal{C}_{v \to N}) < \mathsf{inter}(\mathcal{C})$ **then** move(v, N)
until *no more changes*
return $\mathcal{C}$

in constant time for each move if the volume of the individual clusters and the number of edges between clusters is maintained, this yields a running time in $O(m)$ for each round in LM.

5.1.1 Comparing Moves Efficiently

For the following arguments, let us assume that we always maintain the volume, size and the number of outgoing edges of each cluster during the moving procedure. In addition to that, recall that we can determine the number of edges that link v with each of its neighboring clusters in $O(d_v)$, leading to amortized constant time for each neighbor, as we again consider all neighboring clusters at once (cf. Section 3.3.2). Furthermore, we only consider the lowest level in the hierarchy, i.e., vertex moves in the original graph; it is not hard to see that everything carries over to the aggregated graphs on higher levels in a straightforward way by taking vertex and edge weights, as well as loops into account. Analoguous to the discussion in Section 4.4.4, where we showed that all considered intracluster density measures are efficiently decidable, the feasibility of a clustering with respect to mid, gid and aid after moving a vertex to another cluster can be decided in constant time using this information and other easy to maintain properties of the clustering. Furthermore, it is not hard to see that the change in the number of intercluster edges after a move operation, and therefore the resulting value of nxe can be determined in constant time. For gxd, if we maintain nxe and the current number of intercluster pairs $x_p(\mathcal{C})$ throughout the algorithm, the value of gxd after moving a vertex v to cluster N can be computed as:

$$\mathsf{gxd}(\mathcal{C}_{v \to N}) = \frac{\mathsf{nxe}(\mathcal{C}) - e_{\{v\},\mathcal{C}(v)\setminus\{v\}} + e_{\{v\},N}}{x_p(\mathcal{C}) - n_{\mathcal{C}(v)} + 1 + n_N}$$

Regarding measures based on expansion, it is not hard to see that the expansion of $\mathcal{C}(v)$ without v can be expressed as

$$\mathsf{xe}(\mathcal{C}(v) \setminus \{v\}) = \frac{x_{\mathcal{C}(v)} - d_v + e_{\{v\},\mathcal{C}(v)\setminus\{v\}}}{\min\{n_C - 1, n - n_C + 1\}}.$$

A similar expression yields the expansion $\mathsf{xe}(N \cup \{v\})$ of the cut associated with v's new cluster. Using this, we can update the current value of aixe as follows:

$$\mathsf{aixe}(\mathcal{C}_{v \to N}) = \frac{1}{r} \cdot \left[|\mathcal{C}| \cdot \mathsf{aixe}(\mathcal{C}) + \mathsf{xe}(\mathcal{C}(v) \setminus \{v\}) + \mathsf{xe}(N \cup \{v\}) - \mathsf{xe}(\mathcal{C}(v)) - \mathsf{xe}(N) \right],$$

where $r = |\mathcal{C}|$ if both $\mathcal{C}(v) \setminus \{v\}$ and N are non-empty, $r = |\mathcal{C}| - 1$ if $\mathcal{C}(v) \setminus \{v\}$ is empty and $r = |\mathcal{C}| + 1$ if N is empty. Hence, aixe can be updated efficiently when moving vertices between clusters. It is not hard to see that the above equations carry over to aixc and aixd. In summary, the benefit of moving a vertex to another cluster can be computed in constant time for nxe, gxd, aixe, aixc and aixd and therefore, its new cluster can be determined in $O(n)$. By contrast, similar to the algorithm based on greedy merging, the maximum measures mixd, mixc and mixe are expensive to maintain.

5.1.2 Dealing with Maximum Measures

Another issue with a direct application of GVM to maximum-based measures is that iteratively traversing the whole vertex set is inefficient if only very few vertex moves potentially decrease the cut of the cluster with the currently worst value. Even worse, if this cluster is not unique, it is likely that the search gets stuck in a local minimum, as vertex moves generally can only improve the value for one of these clusters, not for all of them simultaneously. If we try to prevent this by allowing vertex moves that are not strictly improving, we somehow have to ensure that the algorithm terminates after a finite number of operations. We handle this in a similar way as proposed in Chapter 4 for GM by greedily optimizing the lexicographical order of the intercluster sparsity values of all the clusters. Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ and $L(\mathcal{C}) := \big(f(C_1), \ldots, f(C_k)\big)$ be the sequence of these values with decreasing intercluster density, i.e. $f(C_i) \geq f(C_{i+1})$ for $i \in \{1, \ldots, k-1\}$. Recall that we defined a clustering $\mathcal{C}$ to be *L-better* than $\mathcal{C}'$ if $L(\mathcal{C})$ is lexicographically less than $L(\mathcal{C}')$. We now determine for each vertex the set of clusterings that can be reached by moving it. If one of these clusterings is L-better than the current clustering, the move that results in the L-best sequence is performed. As we strictly improve the lexicographical order in each step, termination is guaranteed. This means that we greedily optimize the maximum value but are also allowed to improve the intercluster sparsity of clusters more locally, yielding better efficiency and the possibility to escape local minima.

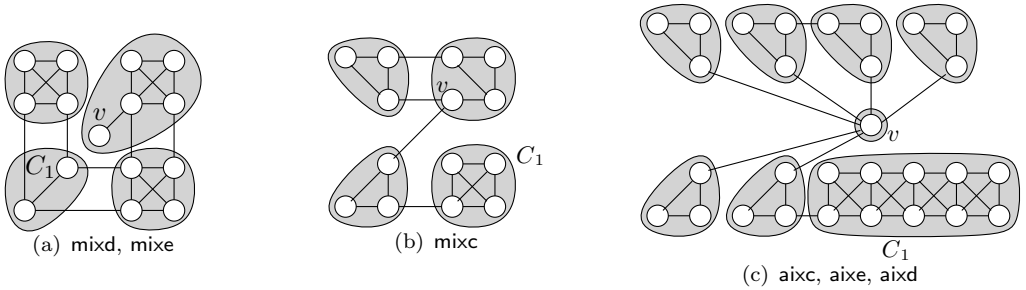
Using the following observation, it can be seen that, for any vertex v, any two clusterings resulting from either leaving v untouched or from moving v to a different (or new) cluster can be L-compared in constant time.

Observation 5. *For three distinct clusters C, A and B in $\mathcal{C}$ and $v \in C$ it holds that:*

(i) $\mathcal{C}_{v \to A}$ *is L-better than* $\mathcal{C} \Leftrightarrow \big\{C \setminus \{v\}, A \cup \{v\}\big\}$ *is L-better than* $\{C, A\}$

(ii) $\mathcal{C}_{v \to A}$ *is L-better than* $\mathcal{C}_{v \to B} \Leftrightarrow \big\{A \cup \{v\}, B\big\}$ *is L-better than* $\big\{B \cup \{v\}, A\big\}$

If the volume, size and number of out-going edges of the clusters A, B and C are maintained by the algorithm, the density/conductance/expansion of $C, A, B, C \setminus \{v\}, A \cup \{v\}$ and $B \cup \{v\}$ can be determined in constant time. Hence, the conditions on the right-hand side can be evaluated in constant time. If we compare clusterings with respect to this lexicographical ordering instead of by the value of inter in Algorithm 7, this can be used to determine the best move for a vertex efficiently.



(a) mixd, mixe (b) mixc (c) aixc, aixe, aixd

FIGURE 5.1: Examples illustrating that most measures considered do not enforce connected moves. Given the clusterings indicated by the gray areas, among all moves involving v, moving v to cluster C_1 yields the largest decrease in the objective function.

5.1.3 Connected Moves

Until now, we justified why for all measures considered, the best feasible move for a vertex can be determined in $O(n)$ operations. Similar to the discussion of the algorithm based on greedy merging, if it suffices to consider neighboring clusters, the algorithm is more efficient. More specifically, as discussed in the case of modularity optimization in Section 3.3.2, each round of the algorithm can be performed in linear time. This raises again the question whether the measures we consider *enforce connected moves*.

It is immediate that moving a vertex to a cluster it is not linked to can never decrease the number of intercluster edges (nxe). The equivalent does not hold for gxd. However, the following equation shows that isolating a vertex is always better than moving it to a cluster it is not linked with. Therefore, we never have to consider non-neighboring clusters when minimizing gxd. Let $v \in V$, $A := C(v) \setminus \{v\}$ and $B \in \mathcal{C}$ such that $e_{\{v\},B} = 0$, then:

$$\mathsf{gxd}(\mathcal{C}_{v\to\{\}}) = \frac{\sum_{\{C_i,C_j\}\in\binom{\mathcal{C}}{2}} e_{C_i,C_j} + e_{\{v\},A}}{\sum_{\{C_i,C_j\}\in\binom{\mathcal{C}}{2}} |C_i||C_j| + |A|} < \frac{\sum_{\{C_i,C_j\}\in\binom{\mathcal{C}}{2}} e_{C_i,C_j} + e_{\{v\},A} - \overbrace{e_{\{v\},B}}^{=0}}{\sum_{\{C_i,C_j\}\in\binom{\mathcal{C}}{2}} |C_i||C_j| + |A| - \underbrace{|B|}_{>0}} = \mathsf{gxd}(\mathcal{C}_{v\to B})$$

For all other intercluster density measures, moving vertices to clusters that contain none of their neighbors may be necessary, as can be seen in the examples in Figure 5.1. Thus, as might be expected, the list of measures enforcing connected moves is the same as the one enforcing connected merges.

From a practical point of view, we can argue that configurations like the above counterexamples are only expected in degenerate cases. Furthermore, the impact on efficiency on sparse graphs is large if we consider all possible clusters instead of only neighboring ones. As unconnected clusters are not desirable in the context of graph clustering anyway and we observed no negative impact on the quality, we chose to restrict the set of feasible moves to neighboring clusters, both for GM and GVM. Together with the possibility to compare different moves in constant time, we get a time complexity of $O(m)$ for each round of the local move procedure for each of the combinations considered. As with the modularity based version (cf. Section 3.3.2), we do not know of any good upper bound on the number of rounds. It can however be verified that the value of any clustering with respect to the considered intercluster sparsity measures is larger than 0 and at most n. Furthermore, the improvement when moving a vertex can be bounded below by $1/p$,

graph	n	m	graph	n	m
karate(N)	34	78	netscience(N)	1589	2742
dolphins(N)	62	159	power(N)	4941	6594
lesmis(N)	77	254	hep-th(N)	8361	15751
polbooks(N)	105	441	PGPgiantcompo(A)	10680	24316
adjnoun(N)	112	425	astro-ph(N)	16706	121251
football(N)	115	616	cond-mat(N)	16726	47594
jazz(A)	198	2742	as-22july06(N)	22963	48436
celegansneural(N)	297	2148	cond-mat-2003(N)	31163	120029
celegans_metabolic(A)	453	2039	cond-mat-2005(N)	40421	175693
polblogs(N)	1490	16718			

TABLE 5.1: List of the real world test instances ordered by increasing number of vertices. These are taken from the webpages of Arenas(A) [12] and Newman(N) [170] and are often used to compare clustering algorithms.

where p is a polynomial in n, which yields a very weak but polynomial upper bound on the number of rounds. However, it turns out that in practice, only few rounds are needed until the algorithm converges.

5.2 Experimental Comparison of Algorithms

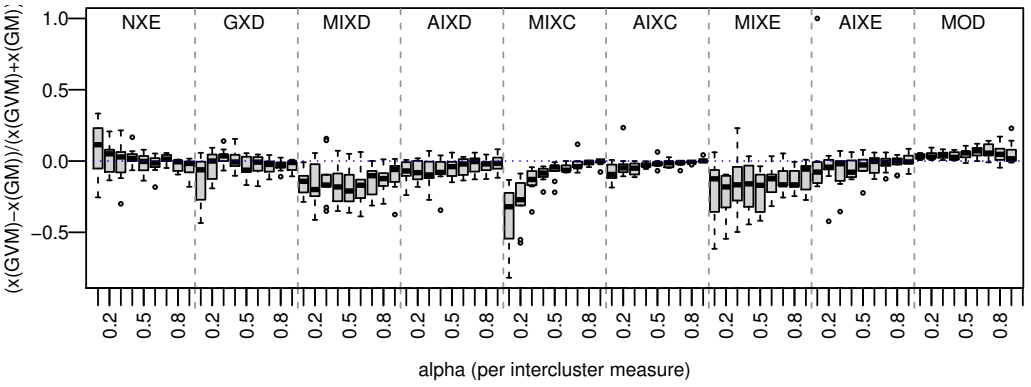
This section empirically evaluates GVM and GM with respect to their ability to optimize DCC, and with respect to the uniformity of the resulting cluster size distribution.

5.2.1 Qualitative Comparison of Greedy Merge and Greedy Vertex Moving

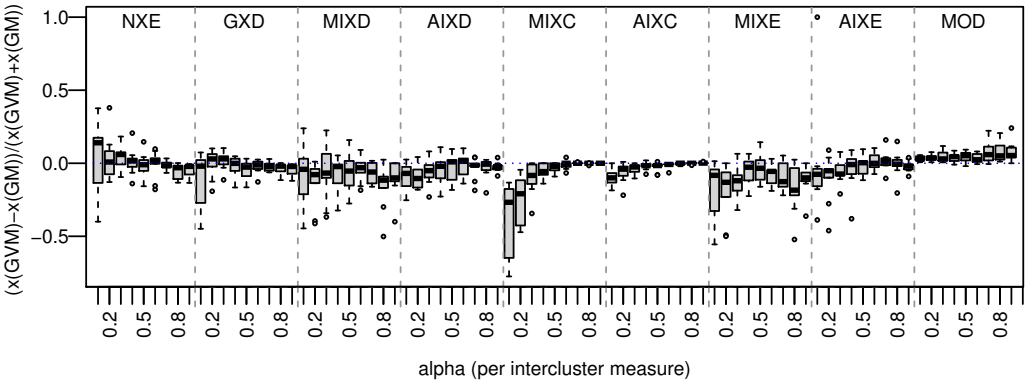
Our first experiments address the question which flavor of greedy algorithm is better suited for DCC. As test instances, we used all graphs listed in Table 5.1 with less than 1000 vertices; these are real-world networks taken from the websites of Mark Newman [170] and Alex Arenas [12]. For all proposed combinations of measures, Figure 5.2 shows the difference of the intercluster density obtained by using GVM and GM, normalized by the sum of these values. With the exception of mod, a value below 0 indicates that GVM yields better results than GM and vice versa. In contrast to more straightforward alternatives as plotting the ratio of x obtained with GVM and GM, this measure is only undefined if both values are 0; this only happens for the experiments including the instances `zachary` and `jazz` in combination with $\alpha = 0.1$. Moreover, the values are distributed more evenly to the plotting area.

For modularity, the median of these values is always greater than zero, confirming that local moving yields better results, regardless of the choice and strength of the constraint. In combination with gid and mid, this similarly holds for all other objectives in a large majority of parameter settings, except for nxe. Note that, in contrast to modularity, we aim to minimize these measures and therefore a value below zero means that GVM attains better results. For nxe, the outcome depends on the value of α.

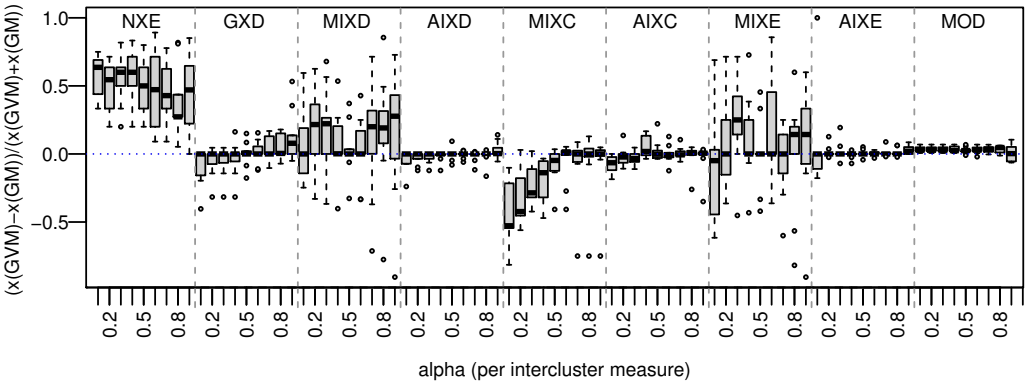
In combination with aid, the outcome is less clear, the results for nxe are significantly better when using GM instead of GVM. This can be explained by the observation that



(a) Constraint: global intracluster density, gid $\geq \alpha$



(b) Constraint: minimum intracluster density, mid $\geq \alpha$



(c) Constraint: average intracluster density, aid $\geq \alpha$

FIGURE 5.2: Qualitative comparison of GVM and GM in combination with different combinations of intercluster measure x, intracluster measure i and threshold α. For each configuration, the distribution of $\big(x(\mathrm{GVM}_{i,\alpha,x}) - x(\mathrm{GM}_{i,\alpha,x})\big)/\big(x(\mathrm{GVM}_{i,\alpha,x}) + x(\mathrm{GM}_{i,\alpha,x})\big)$ with respect to the test set described in Section 5.2.1 is shown. The box-whisker plots use the default settings of R [180]. The thick line represents the median of the data, the upper and lower bound of the box represent the upper and lower quartiles. The dots represent outliers, i.e., values that deviate by more than 1.5 times the inner quartile range from the median. The whiskers represent the maximum and minimum of the remaining values.

aid happily allows (and thereby encourages) unbalanced clusterings, as bad intracluster density values of large clusters can easily be compensated by a set of small and dense clusters. Consequently, optimal solutions for this configuration typically consist of one huge cluster and one or several singleton vertices with low degree. As GM has a tendency to grow one cluster without changing other parts of the clustering, low-degree vertices are likely to be unclustered in the resulting clustering. In contrast to that, GVM assigns these vertices to their anchor vertices in the first pass and, for nxe, will never isolate them again, which leads to worse results for this configuration. A more detailed evaluation of the skewness of the cluster size distribution produced by GM and GVM can be found in Section 5.2.3.

As degenerately unbalanced clusterings are usually not intuitive, we deem aid less suitable in the context of graph clustering. Disregarding aid for these reasons, in the vast majority of configurations, GVM outperforms GM.

5.2.2 Evaluating Balancedness using the Gini Coefficient

To gain further insights into the behavior of the two greedy algorithms, we additionally evaluate the balancedness of the clusterings they produce in terms of the *Gini coefficient* [90] of the resulting cluster size distribution. The Gini coefficient is a measure of inequality that originates from economics and is a common index to assess the inequality of the distribution of income or wealth. It is based on the *Lorenz-curve* of a sample X, which is a piecewise linear function that describes the proportion of the distribution assumed by the bottom x% of the values in X. Figure 5.3 shows the Lorenz-curve of a small artificial example, the sample $X = \{3, 4, 7, 10\}$. The smallest sample value is 3, which is 1/8 of the total sum of all values; hence, the Lorenz-curve includes the point $(1/4, 1/8)$. The next point is $(1/2, 7/24)$, as the two smallest values add up to 7, which is 7/24 of the total sum, and so on. The Gini coefficient is defined as twice the area between the Lorenz-curve and the line of perfect equality (A); it assumes values between 0 and 1. Intuitively, the larger the Gini coefficient, the skewer the distribution of values. To evaluate how balanced a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ is, we determine the Gini coefficient of its cluster size distribution. More precisely, we interpret cluster sizes as sample values, i.e., $\text{Gini}(\mathcal{C}) = \text{Gini}(\{|C_1|, \ldots, |C_k|\})$.

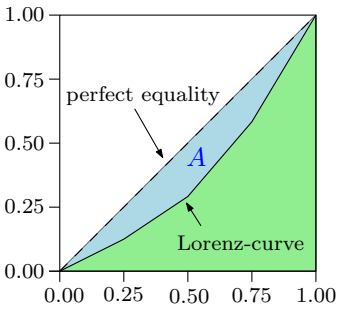


FIGURE 5.3: Lorenz-curve (example).

The Gini coefficient shows some peculiarities when the number of clusters is very small. For example, clusterings consisting of only two clusters cannot have a Gini coefficient larger than 0.5. On the other hand, clusterings that are perfectly balanced, i.e., in which each cluster has the same size, always have a Gini coefficient of 0, independent of the number of clusters. Another positive aspect is that the absolute values do not depend on the size of the graph. Furthermore, if vertices are moved from a larger to a smaller cluster, the Gini coefficient always decreases, no matter what the remaining clustering looks like; this seems to be a very desirable property when evaluating the balancedness of clusterings.

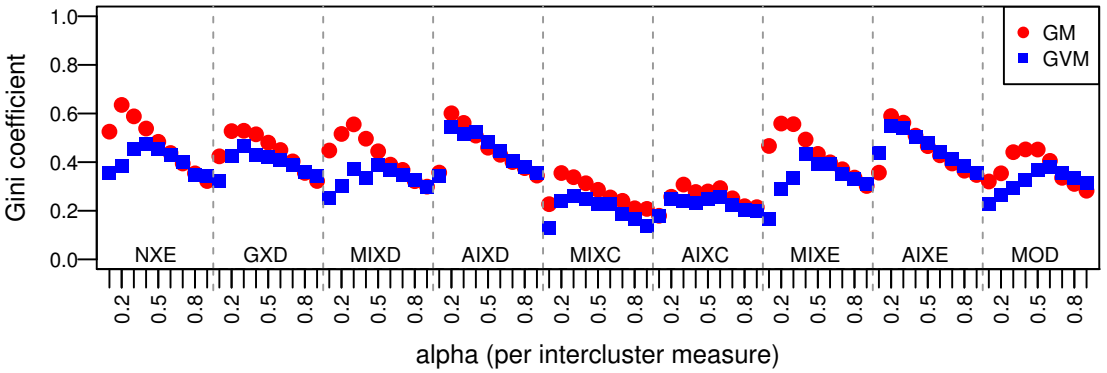


FIGURE 5.4: Gini coefficient, clusterings produced by using GVM and GM with constraints on gid.

Many objectives for graph clustering, including conductance or modularity, are implicitly based on a tradeoff between balancedness and the number of edges cut by the clustering. However, enforcing rigid balance constraints as in the context of graph partitioning [31] is usually considered as too strict; in contrast, there are several studies on real world networks that indicate cluster sizes following a power law distribution [13, 53, 175]. Our aim in comparing the Gini coefficient of the clusterings produced by GM and GVM is not to judge these clusterings with respect to their quality, but to gain some indication on the question which algorithm is better suited for particular objective functions or applications.

5.2.3 Cluster Size Distribution with GM and GVM

Figure 5.4 shows the mean Gini coefficient of the clusterings obtained by using GVM and GM with constraints on gid. Gini plots for the other intracluster density measures can be found in Appendix B.1. In general, the clusters produced by GVM are more balanced, especially for low values of α.

A possible explanation for this fact is the tendency of GM to grow the clusters one by one, as, especially in the early stages of the algorithm, the benefit of decreasing the number of intercluster edges often outweighs negative effects on the balancedness of the clustering. Balancedness, in turn, becomes more and more important as soon as the clusters grow to a certain size. This can cause the first clusters that start growing to become very large, which results in an especially skewed cluster size distribution. Of course, this effect depends on the objective function; in the context of modularity maximization, it empirically has a great impact on the quality of the resulting clustering and can be mitigated by modifying the objective in favor of balance [220]. In contrast to that, the round-based local moving scheme of GVM naturally favors balanced clusterings, as clusters start to grow simultaneously in all parts of the graph. As already mentioned in Section 5.2.1, this might be a reason for the observation that GM often yields better results in combination with nxe and aid, as both of them are more or less oblivious to the skewness of the cluster size distribution.

At first glance, the comparably low values of the Gini-coefficient obtained by both algorithms in combination with aid and low values of α seem to contradict our statement that these clusterings are degenerately unbalanced. This can be explained by the unintuitive behavior of the Gini coefficient when the sample size is very small; as already

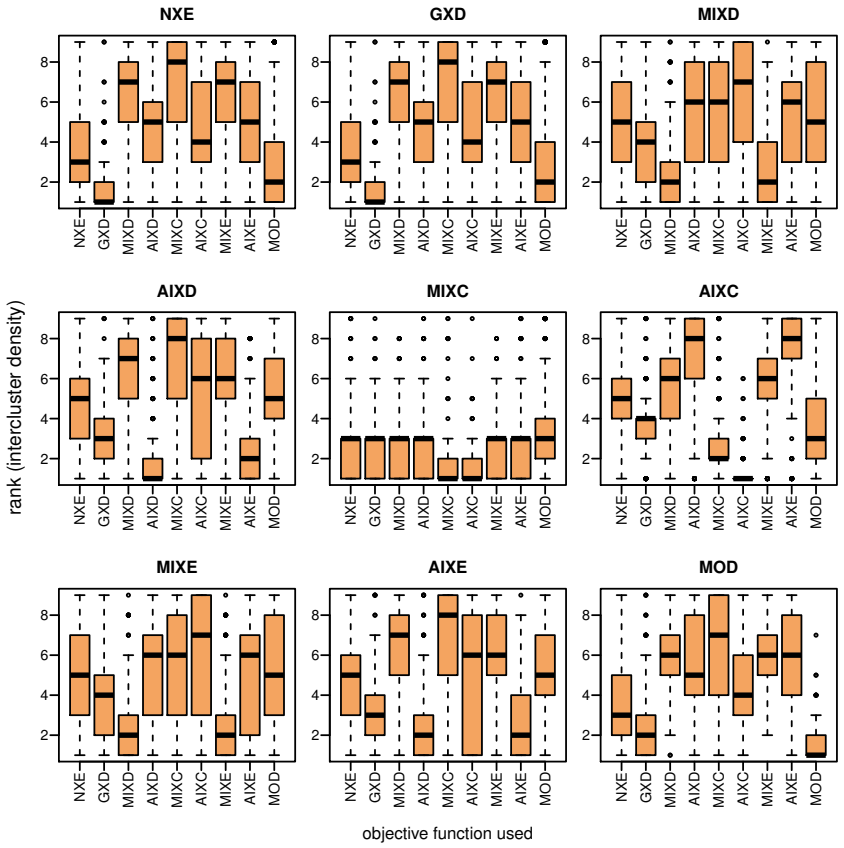


FIGURE 5.5: Effectiveness of different objective functions, one plot per intercluster measure, i.e., evaluated measure (see supertitles). In each plot, ranks for different intercluster density measures as objectives in the GVM-algorithm (indicated by the label of the x axis) using gid as constraint are shown. More precisely, an intercluster measure x obtains rank k with respect to another intercluster measure y, if optimizing x leads to a clustering with the k-best value of y.

mentioned, clusterings containing just two clusters, which are generated frequently with aid and $\alpha \leq 0.5$, cannot have Gini coefficients larger than 0.5, even if they are maximally unbalanced.

5.2.4 Effectiveness of Different Objective Functions

As discussed in Section 5.2.1, GVM most often yields better quality than GM. For tackling DCC in the following experiments, we thus solely use GVM, putting aside the algorithm based on greedy merging. The next question we pose is, if each of the intercluster density measures is effective in optimizing itself when used as inter in GVM. To answer this question, we conducted the following experiment on all graphs listed in Table 5.1. In the following, let $\text{GVM}_{i,\alpha,x}$ denote GVM incorporating the constraint $i(\mathcal{C}) \geq \alpha$ and the objective $x(\mathcal{C})$. For each setup of DCC, i.e. intracluster measure i, intercluster measure x and $\alpha \in \{0.0, 0.1, \ldots, 1.0\}$, we ranked the clusterings obtained by $\text{GVM}_{i,\alpha,y}$ by their performance with respect to x, using all possible objectives y for GVM. Figure 5.5 shows the distribution of these ranks over all configurations involving gid, grouped by x. The outcome of this experiment is less clear than what might be expected—none of the intercluster measures, not even modularity, scores the best quality with respect to

itself in all configurations. Nonetheless, in general, each objective optimizes itself quite well, except for nxe which is clearly dominated by gxd.

These observations also hold for mid, while for aid, the outcome is even less clear, as can be seen in Figures B.3, B.4 in Appendix B.2.

5.2.5 Reference Algorithms

For a more comprehensive assessment of GVM as a means to address DCC, we use the following reference algorithms:

- *Iterative Conductance Cutting (ICC)* [129]: This top-down algorithm iteratively splits the input graph into two subgraphs based on a cut with low conductance. The process stops when the conductance of the cut exceeds a given threshold, which we set to 0.4 in our experiments.
- *Markov-Clustering (MCL)* [215]: Emulating a random walk, the matrix of transition probabilities is alternately taken to the power of e and renormalized after taking each entry to the power of r, where e and r are input parameters. In our experiments, we set r and e to 2; this equals the default settings in the implementation of van Dongen and the parameter settings used in [37]. According to van Dongen, it is necessary to add (weighted) self loops to each vertex in the graph to prevent the result to reflect the bipartite characteristics of the input graph [216]. In the implementation used in the preliminary version [103], this is done by modifying the random walk matrix of the graph such that each entry on the diagonal receives a constant probability of 0.01 and the other entries are rescaled such that the matrix remains stochastic. Here, we use the exact implementation provided by van Dongen, which uses the following strategy. Each vertex receives an (unweighted) self loop prior to constructing the random walk matrix. As the two weight assignment schemes are not equivalent, the experimental results differ slightly from [103].
- *Geometric MST Clustering (GMC)* [37]: First, a spectral embedding of the graph in d-dimensional space is built. Then the algorithm constructs a Euclidean minimum spanning tree and successively deletes the heaviest edge. This defines a sequence of forests whose connected components induce a set of clusterings. Among these clusterings, the one with the best value according to some given objective function is chosen.
- *Multi-Level Modularity (ML-MOD)* [189]: This is the GVM-algorithm based solely on modularity without using any constraint. This algorithm has been shown to perform very well in the context of modularity optimization [189].

5.2.6 Comparison Based on Intracluster Density Found by Reference Algorithms

ICC, MCL and ML-MOD do not incorporate constraints on the intracluster density of the resulting clustering. Nonetheless, it is still possible to evaluate them with respect to those variants of DCC, where α is set to the intracluster density found by these

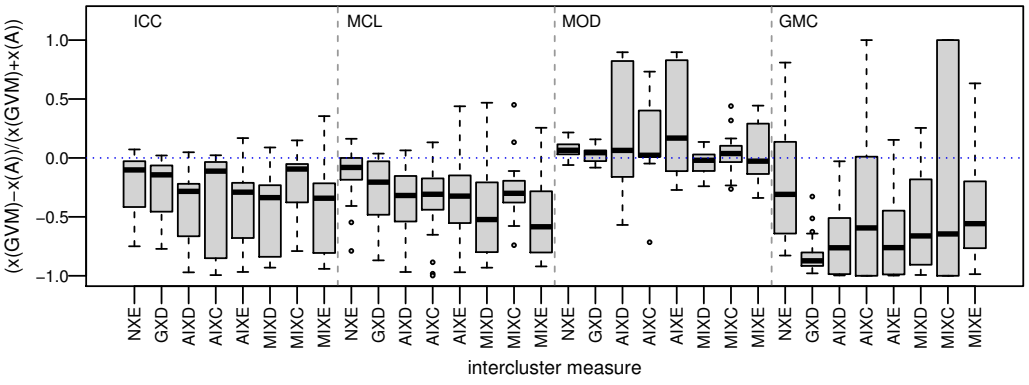
algorithms. In other words, given the same constraint a reference algorithm $\mathcal{A}$ implicitly adheres to, how well does GVM compare to $\mathcal{A}$ wrt. DCC?

We first ran ICC, MCL and ML-MOD on all test instances in Table 5.1 and recorded the intracluster density values of the resulting clusterings. Then, for each reference algorithm $\mathcal{A}$, constraint i, recorded corresponding intracluster density α and objective x, we compare the clustering obtained by $\text{GVM}_{i,\alpha,x}$ to the clustering of $\mathcal{A}$ with respect to x. For GMC the experiments slightly differ as GMC requires an objective function. We filled this degree of freedom by choosing $f(\mathcal{C}) = i(\mathcal{C}) - x(\mathcal{C})$ as the objective function for the experiments using i as intracluster and x as intercluster density measure. This seemed to be the fairest way of comparison and in almost all cases led to non-trivial clusterings. More specific, trivial clusterings only occurred in combination with aixc and mixc and are the reason for the increased number of ties in these scenarios.

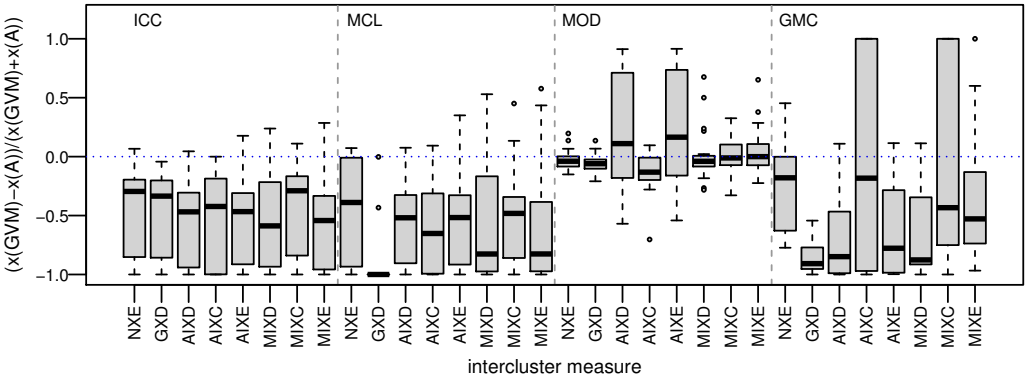
Table 5.2 shows the percentage of graphs where the greedy algorithm for x compares favorably. More detailed, Figure 5.6 shows the difference in x obtained with GVM and with the reference algorithm, normalized by the sum of these values. As we aim to minimize intercluster density, a value below zero indicates that the greedy algorithm yields better results than the reference algorithm and vice versa. Again, this measure is only undefined if both values are 0; in this rare case, we exclude the instance from the plot. Compared to ICC and MCL, GVM clearly yields better results. For GMC, GVM yields the same or better intercluster density in the majority of configurations, except for the combination of mixc with constraints on the average intracluster density. This can be explained by the fact that aid does not punish unbalancedness and GMC naturally leads to very unbalanced clusterings in most instances. The outcome of the comparison with the modularity-based algorithm is less clear. For aid, GVM performs better, which is not surprising as modularity strongly discourages unbalanced clusterings. For mid, GVM still beats ML-MOD in the majority of configurations, while for gid, this only holds for slightly less than half of the configurations. Furthermore, it is worth mentioning that especially for aixd and aixe there are instances where modularity minimizes these functions far better than the respective greedy algorithms. Altogether, the comparison with ICC, MCL and GMC suggests that GVM effectively addresses DCC, while the comparison with ML-MOD shows that optimizing modularity is similarly effective in minimizing cut-based intercluster sparsity measures.

TABLE 5.2: Comparison of GVM and reference algorithms. The first entry represents the percentage of graphs where GVM compares favorably, i.e., yields strictly better intercluster density than the reference algorithm. The second entry denotes the percentage of graphs where GVM and the reference algorithm yield the same intercluster density.

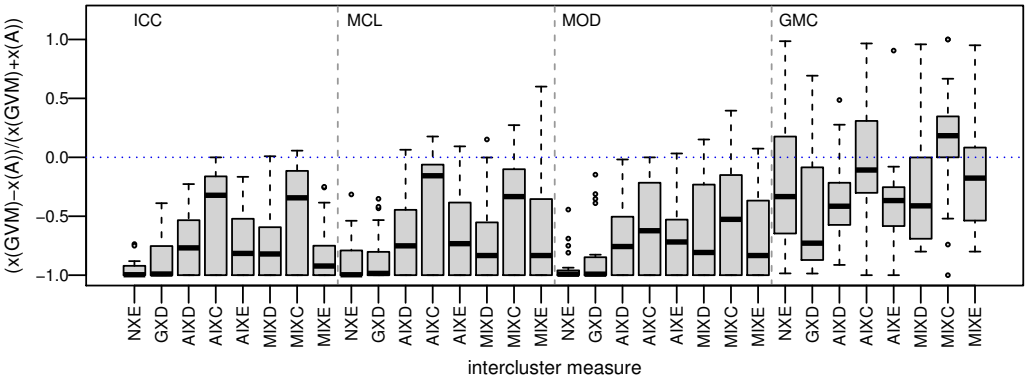
	gid				mid				aid			
	ICC	MCL	MOD	GMC	ICC	MCL	MOD	GMC	ICC	MCL	MOD	GMC
nxe	84/5	68/11	16/5	63/0	89/5	74/5	63/0	74/0	95/5	100/0	100/0	63/0
gxd	84/5	95/0	42/0	100/0	95/5	100/0	84/0	100/0	95/5	100/0	100/0	84/0
aixd	84/5	95/0	42/0	100/0	89/5	89/0	37/0	95/0	95/5	95/0	100/0	84/0
aixc	84/5	95/0	21/0	53/26	95/5	89/0	79/0	42/32	95/5	84/0	100/0	63/0
aixe	84/5	84/0	42/0	89/0	89/5	79/0	42/0	95/0	95/5	89/0	95/0	95/0
mixd	84/5	79/0	53/0	84/0	89/5	84/0	74/0	89/0	89/5	95/0	89/0	74/0
mixc	89/5	89/0	42/0	37/32	89/5	89/0	63/0	37/37	89/5	84/0	84/0	21/16
mixe	89/5	79/0	58/0	89/0	84/5	79/0	47/5	79/0	95/5	79/0	89/5	63/5



(a) gid



(b) mid



(c) aid

FIGURE 5.6: Comparison to reference algorithm $\mathcal{A} \in \{\mathrm{ICC}, \mathrm{MCL}, \mathrm{MOD}, \mathrm{GMC}\}$. For each configuration of intracluster measure i and intercluster measure x, the distribution of $\big(x(\mathrm{GVM}_{i,\alpha,x}) - x(\mathcal{A})\big) / \big(x(\mathrm{GVM}_{i,\alpha,x}) + x(\mathcal{A})\big)$ with respect to the graphs in Table 5.1 is shown. α is set to the corresponding value of i in the clustering produced by $\mathcal{A}$.

5.2.7 Implementation and Running Times

The algorithms ICC, GMC and GM are implemented in Java 1.6.0_22 using the graph library yFiles [231]. GVM (also incorporating ML-MOD as a special case) is implemented in C++ using version 1_42 of the Boost Graph Library[2] and compiled with gcc 4.5.2 with optimization level 4. For MCL, we used the highly optimized software provided by van Dongen[3], version 12-135. The focus of this evaluation is on the quality of the resulting clusterings, not on running times. However, to get a rough impression about the latter, clustering cond-mat-2005 on a 2.1 GHz AMD Opteron processor 6172 takes about 6 hours with ICC, 5 minutes with GMC, 46 seconds with MCL, and 3 to 15 seconds with GVM, depending on the parameter setting. The version of GVM corresponding to ML-MOD takes about 3.5 seconds. With our prototype implementation (not implementing any of the efficient SAHN algorithms from Chapter 4) of GM, clustering the much smaller celegans_metabolic takes over 2 minutes.

5.3 Comparison of Density and Sparsity Measures

To compare the different objective functions qualitatively, we evaluated how well the corresponding GVM-algorithms are able to reconstruct planted partitions in random graphs. As a comparison, we also give the results obtained by ML-MOD. Due to higher running times and large numbers of experiments, we omit a comparison with ICC, MCL and GMC.

5.3.1 Random Graphs

We use an adapted Erdős-Rényi-model, where, starting from a given reference partition, the probability that vertices in the same set (in different sets) are connected equals p_{in} (p_{out}). The number of vertices (n) and clusters (k) as well as the skewness of the distribution of cluster sizes (β) of the planted partition are input parameters. Setting β to 1.0 corresponds to uniform cluster sizes, values below and above 1 cause this distribution to be skewed, for more details see Chapter 8 and the technical report by Görke and Staudt [104]. As configurations, we fixed $n = 10000$ and chose p_{in} and p_{out} such that the average number of intracluster (intercluster) edges a vertex is incident to equals 5 (3). To determine the reference partition, we used all combinations of $k \in \{10, 100, 300\}$ and $\beta \in \{0.3, 1.0, 2.0\}$. For each configuration, we generated 100 instances and always averaged over the obtained values.

5.3.2 Distance Measures

To compare the clusterings obtained with the different algorithms to the reference clustering, we use the following graph-based distance measures taken from [61]:

- *Graph-based Rand Index (R_g)*: Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be clusterings and e_{11} (e_{00}) the number of edges which are intracluster (intercluster) wrt. both $\mathcal{C}_1$ and $\mathcal{C}_2$. Then, $R_g(\mathcal{C}_1, \mathcal{C}_2) = 1 - (e_{11} + e_{00})/m$.

[2] http://www.boost.org/
[3] http://micans.org/mcl/

- *Editing Set Difference (ESD)*: For a clustering $\mathcal{C}$, its editing set $F_{\mathcal{C}}$ is the set of edges requiring insertion or removal such that the clusters in $\mathcal{C}$ form disjoint cliques. Then, for clusterings $\mathcal{C}_1$ and $\mathcal{C}_2$, their editing set difference is defined as $ESD(\mathcal{C}_1, \mathcal{C}_2) = 1 - |F_{\mathcal{C}_1} \cap F_{\mathcal{C}_2}|/|F_{\mathcal{C}_1} \cup F_{\mathcal{C}_2}|$.

5.3.3 Parameters and Evaluation

As an exhaustive parameter search for all configurations would be far too expensive, we always set α to 75 percent of the expected global intra-cluster density p_{in}. We deemed taking the actual value of p_{in} too strict, as, especially for mid, even the reference clustering of the generator most likely does not meet this constraint. The previous experiments indicate that there are configurations where particular objective functions used in GVM do not score the best results with respect to themselves. As our goal was to compare good clusterings with respect to different combinations of i and x, independent of artifacts of GVM, we chose the following approach: For a combination i, α, x, we evaluated the clustering that, among all results obtained with GVM using $i \geq \alpha$ as constraint, is best with respect to x (as opposed to simply evaluating $\text{GVM}_{i,\alpha,x}$). Furthermore, preliminary experiments confirmed that constraining aid leads to very unintuitive and unbalanced clusterings, which is reflected in the fact that the corresponding versions of DCC are far less effective in finding the hidden clustering. For this reason, we excluded aid in the discussion of the results.

5.3.4 Results on Planted Partition Graphs

Figure 5.7 and Figure 5.8 show the results for selected configurations, the results for the whole set of experiments can be found in Appendix B.3. In the first plot of Figure 5.7 it can be seen that, in general, the clusterings that are ranked best with respect to mod, nxe and gxd are most similar to the reference.

Constraining modularity by mid *improves its results.* This especially holds for the experiments with high skewness ($\beta = 2$) and $k = 300$. In these experiments, modularity finds far less clusters than expected, partially due to its known resolution limit [84], which can be circumvented by steering the coarseness of the clustering by constraining the intracluster density. Another interesting fact is that ESD punishes the coarse clustering obtained by pure modularity far more than R_g.

aixe *and especially* aixd *identify many clusters.* Another striking observation is that the average number of clusters in clusterings found by aixd and aixe, indicated by the green $\times$-marks, is much higher than the average number of clusters in the reference. This especially stems from the experiments with few clusters. In the configuration with $\beta = 1$ and $k = 10$, it can also be seen that these measures differ the more, the coarser the expected clustering gets. This is not unexpected, as the denominator of aixd grows more slowly with the number of vertices in the cluster than the denominator of aixe, meaning that aixd is less eager to produce very large clusters. Additionally, in Chapter 4 it was proven that with the exception of aixd, all inter-cluster measures considered here can always be ameliorated by merging two existing clusters (unboundedness), which is also a hint that aixd is less likely to produce coarse clusterings than the other measures.

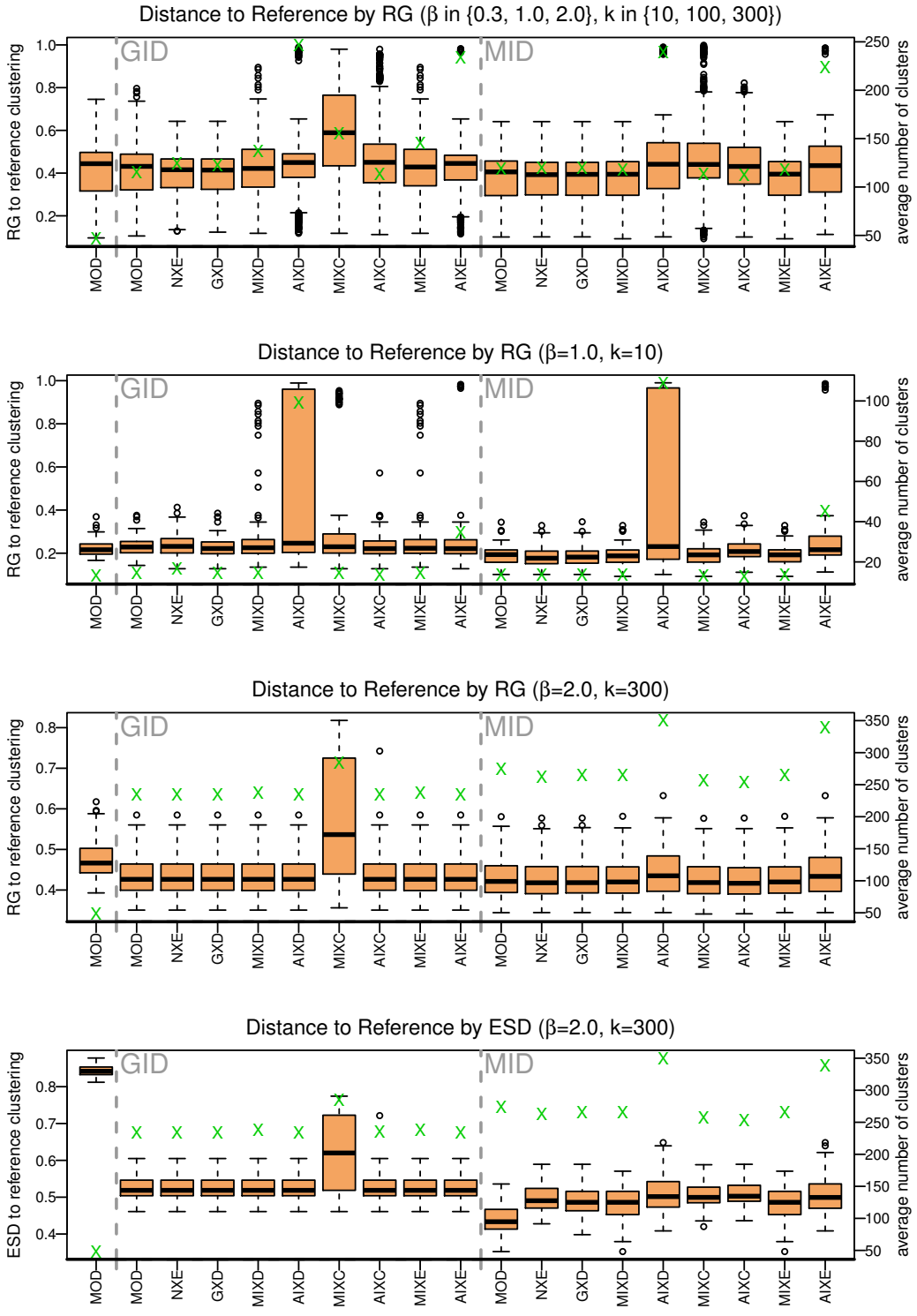


FIGURE 5.7: Distance to reference clustering (boxplots, left-hand y-axis) and number of clusters discovered in planted partition graphs (green $\times$-marks, right-hand y-axis), different configurations

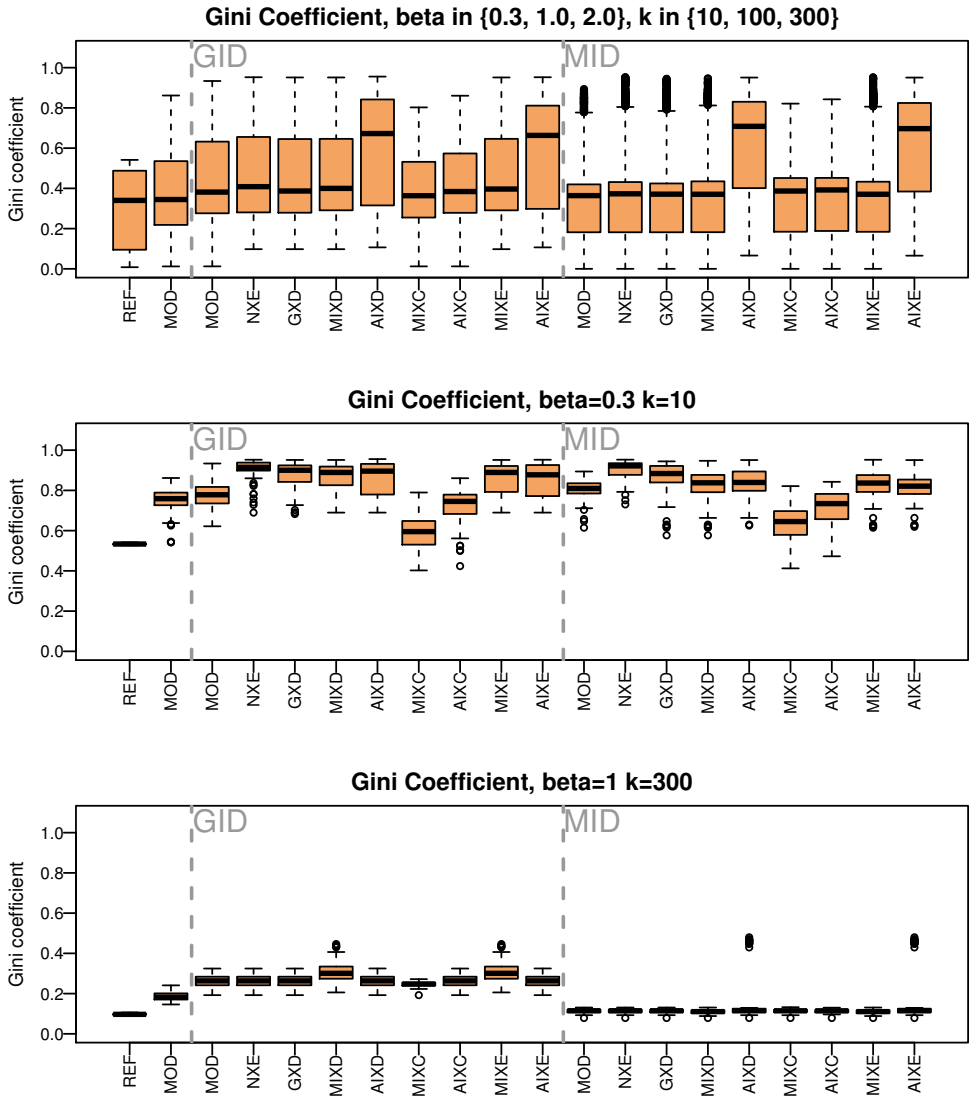
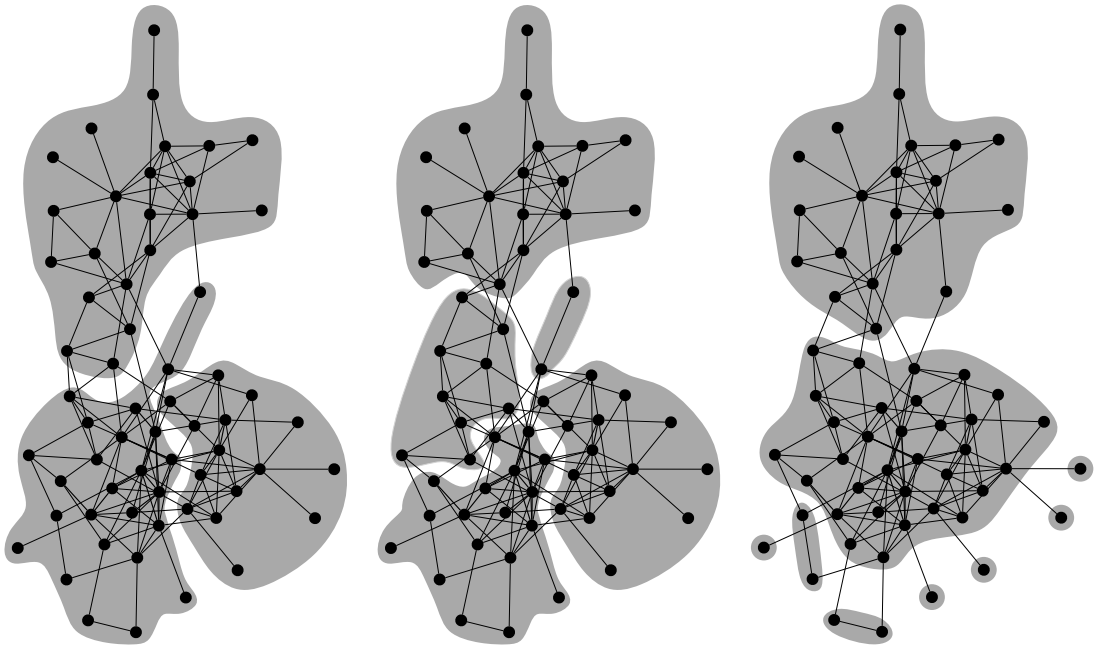


FIGURE 5.8: Gini coefficient of cluster size distribution, different configurations.

aixe *and especially* aixd *favor unbalanced clusterings.* This holds for a number of configurations with $k = 10$ or $k = 100$, and is also clearly visible in the plot showing a summary of all configurations (see Figure 5.8). This does not hold for their worst-case counterparts mixe and mixc.

mixc *yields significantly worse results in some configurations.* In our preliminary paper [103], we conjectured that, for configurations with fine reference clusterings, this can be explained by unbalanced clusterings. The corresponding plot for the Gini coefficient for the experiments with $k = 300$ and $\beta = 1$ reveals that this is not the case. In contrast to that, clusterings obtained by optimizing mixc are more balanced than the ones obtained by other measures in some configurations, especially in the experiments with $\beta = 0.3$ and $k = 10$. Interestingly, this leads to a very high distance to the reference clustering. By contrast, the intercluster expansion or density of such clusters is usually very low. Note that this effect seems to be a lot weaker when averaging the values with aixc.



(a) mixe (b) mod (c) aixd

FIGURE 5.9: Network of frequent associations between 62 dolphins in a community living of Doubtful Sound, New Zealand [159]. The clusterings displayed are obtained by optimizing the measures (a) mixe, (b) mod and (c) aixd with GVM under the restriction gid > 0.2.

Intra- and intercluster measures both influence the skewness of the resulting cluster size distribution. This influence depends strongly on the structure of the input graph. Roughly, with the exception of aid, the choice of the intercluster measure seems to have more influence on the balance, while in total, clusterings obtained by constraining mid are slightly more balanced than the ones obtained by constraining gid.

5.3.5 Selected clusterings on small example network.

Figure 5.9 demonstrates the differences between intercluster measures on a small network reflecting social interaction of a group of 62 dolphins [159]. As we did not want to introduce an artificial bias towards a particular clustering, the (force-directed) layout of the vertices does not use any information about the clustering. With the restriction gid > 0.2, aixd dominates nxe, gxd and mixd in the sense that the clustering obtained by optimizing aixd with GVM yields less intercluster edges and lower values of gxd and mixd than the corresponding clusterings obtained by optimizing these measures directly. Similarly, mixe dominates aixc and mixc, while aixc dominates aixe. Due to this and to retain visual clarity, we only give the clusterings obtained by aixd, mixe and mod. aixc is omitted because the respective clustering is very similar to the one obtained with mixe, they only differ in the assignment of few vertices connecting the upper with the lower part.

Compared to mixe, the clustering obtained by mod introduces two new clusters that consist of the vertices connecting the left and the right part. The main difference between these clusterings and the one obtained by aixd is the assignment of the nine vertices at

the bottom that are only sparsely connected to the remainder of the graph; mixe and mod assign them to the only clusters they are connected with while aixd essentially leaves them unclustered. Overall, all clusterings are rather similar in the sense that only few vertices are treated differently, all of them either connecting the two parts or being only loosely connected to the network; a human observer might argue in favor of any of the clusterings considered, as the group affiliation of these vertices seems ambiguous.

The reason why nxe, gxd and mixd are dominated by aixd is that the respective versions of GVM merge the sparsely connected vertices at the bottom with their anchor vertices in an early stage of the algorithm. Isolating these vertices later on is not possible, as this would decrease the respective objective function, although isolating these vertices and moving one of the vertices in the middle to the respective cluster would be feasible and improve the objective function.

5.4 Concluding Remarks

This chapter contains an experimental evaluation of algorithms optimizing the objective of Density-Constrained Clustering (DCC). We first evaluated two greedy heuristics, vertex moving and cluster merging, against each other and against algorithms from the literature. Vertex moving proved reliably superior to cluster merging and, in many cases, beats the results of the reference algorithms. Our results also show that a well-known modularity-based algorithm implicitly addresses DCC quite well, revealing similarities between cut-based intercluster sparsity measures and modularity. In the second part, we addressed the question whether different combinations of intracluster density and intercluster sparsity measures are suitable to guide algorithms in recovering planted partitions in random graphs. The results suggest that minimizing the average intercluster expansion or density of the clusters overestimates the number of clusters if the expected clustering is coarse, while maximum intercluster conductance fails to recognize the hidden clustering in a variety of configurations. Interestingly, for some cut measures, the choice of averaging values over all clusters or using the corresponding worst-case counterpart highly affects the nature of the resulting clustering. Additionally, it can be seen that the known resolution limit for modularity can be circumvented if the coarseness of the clustering is controlled by an additional constraint on the intracluster density of the clustering. Similar to other clustering measures involving an additional parameter that steers the coarseness of the resulting clustering, it is still open how to select the parameter α in case there is no prior knowledge on the expected density of the clusters.

Although all measures of intracluster density and intercluster sparsity we evaluated are based on the same paradigm, different combinations lead to different results with respect to characteristics like the skewness of the cluster size distribution or the treatment of satellites. Therefore, depending on the application, it is worthwhile identifying desirable features of the clustering and to choose a concrete optimization problem accordingly.

Chapter 6

Surprise - Complexity and Exact Solutions

In this chapter, we will turn our attention from the cut-based view back to clustering measures based on *null models*. This line of research recently gained a lot of popularity, the most prominent objective function in this context being the *modularity* of a clustering [173]. Roughly speaking, the idea behind this approach is to compare the number of edges within the same cluster to its expected value in a random graph that inherits some properties of the graph given as input (cf. Section 2.3).

In a wider sense, the measure called *surprise* that has recently been suggested as an alternative to modularity is also based on a null model, although, compared to modularity and its modifications [83], it uses a different tradeoff between the observed and expected number of edges within clusters. Surprise is used as a quality function in the tools UVCLUSTER and Jerarca to analyze protein interaction data [15, 4]. The authors' main arguments for using surprise instead of modularity is that it exhibits better behavior with respect to synthetic benchmarks and, empirically, it does not suffer to the same extent from the *resolution limit* of modularity [84], i.e., the tendency to merge small natural communities into larger ones [5, 7, 8]. However, these results are hard to assess, since a metaheuristic [9] is used instead of directly optimizing the measure. It chooses among a set of clusterings produced by general clustering algorithms the one that is best with respect to surprise. We will describe this metaheuristic in detail in Chapter 7, which is concerned with a heuristic algorithm for surprise maximization.

In this chapter, we take first steps towards a theoretical analysis of surprise. We show that the problem of finding a clustering with optimal surprise is $\mathcal{NP}$-hard in general and polynomially solvable on graphs with bounded tree width. Moreover, we formulate surprise as a bicriteria problem, which allows to find provably optimal solutions for small instances by solving a small number of integer linear programs. We further derive an objective function called SMod, which can be seen as a compromise between surprise and modularity, and illustrate its features on some small example instances.

6.1 Definition and Basic Properties

Let $\mathcal{C}$ be a clustering of a graph $G = (V, E)$ with i_e intracluster edges. Among all graphs labeled[1] with vertex set V and exactly m edges, we draw a graph $\mathcal{G}$ uniformly at random. The surprise $S(\mathcal{C})$ of this clustering is then the probability that $\mathcal{G}$ has at least i_e intracluster edges with respect to $\mathcal{C}$. The lower this probability, the more *surprising* it is to observe that many intracluster edges within G, and hence, the better the clustering. The above process corresponds to an urn model with $i_p(\mathcal{C})$ white and $p - i_p(\mathcal{C})$ black balls from which we draw m balls without replacement. The probability to draw at least i_e white balls then follows a hypergeometric distribution, which leads to the following definition[2]; the lower $S(\mathcal{C})$, the better the clustering:

$$S(\mathcal{C}) := \sum_{i=i_e}^{m} \frac{\binom{i_p}{i} \cdot \binom{p-i_p}{m-i}}{\binom{p}{m}}$$

Basic Properties. For a fixed graph, the value of S only depends on two variables, i_p and i_e. To ease notation, we will use the term $S(i_p, i_e)$ for the value of a clustering with i_p intracluster pairs and i_e intracluster edges. The urn model view yields some simple properties that lead to a better understanding of how surprise behaves, and that are heavily used in the $\mathcal{NP}$-hardness proof.

Lemma 6.1. *Let i_e, i_p, p and m be given by a clustering, i.e., $0 \leq i_e \leq i_p \leq p$, $i_e \leq m$ and $m - i_e \leq p - i_p$. Then, the following statements hold:*

(i) $S(i_p, i_e + 1) < S(i_p, i_e)$.

(ii) If $i_e > 0$, *then* $S(i_p - 1, i_e) < S(i_p, i_e)$.

(iii) If $p - i_p > m - i_e$, *then* $S(i_p + 1, i_e + 1) < S(i_p, i_e)$.

Proof. Statement (i) is obvious. Similarly, statement (ii) is not hard to see if we recall that $S(i_p - 1, i_e)$ corresponds to the probability to draw at least i_e white balls after replacing one white ball with a black one.

For statement (iii), we show that the number k_1 of m-element subsets of the set of all balls containing at least i_e white balls is larger than the number k_2 of m-element subsets containing at least $i_e + 1$ white balls after painting one black ball b white. Any subset A that contributes to k_2 also contributes to k_1, as at most one ball in A got painted white. On the other hand, every m-element subset not containing b that contains exactly i_e white balls contributes to k_1, but not to k_2. As there are at least i_e white balls, and $p - i_p > m - i_e$ implies that there are at least $m - i_e + 1$ black balls, there is at least one subset with these properties. Hence $k_1 > k_2$, which is equivalent to $S(i_p + 1, i_e + 1) < S(i_p, i_e)$. □

[1] This means, we consider graphs that are isomorphic to each other but differ in the concrete associations between vertices as two different graphs. For example, two graphs $G_1 = (\{v_1, v_2, v_3\}, \{\{v_1, v_2\}, \{v_2, v_3\}\})$ and $G_2 = (\{v_1, v_2, v_3\}, \{\{v_1, v_3\}, \{v_3, v_2\}\})$ are not identical, although they both represent a path of length 2.

[2] This is the definition used in the original version [15]; later on, it was replaced by maximizing $-\log_{10} S(\mathcal{C})$, which is equivalent with respect to optimum solutions.

In other words, the value of surprise improves the more edges and the less vertex pairs within clusters exist. Moreover, part (iii) shows that if we increase the number of intracluster edges such that the number of *intracluster non-edges*, i.e., vertex pairs within clusters that are not linked by an edge, does not increase, this leads to a clustering with strictly smaller surprise. This immediately yields some basic properties of optimal clusterings with respect to surprise. Part (i) of the following proposition is interesting as it shows that optimal clusterings always fulfill the assumptions of Lemma 6.1(ii)-(iii).

Proposition 6.2. *Let $G = (V, E)$ be a graph that has at least one edge and that is not a clique and $\mathcal{C}$ be an optimal clustering of G with respect to surprise. Then,*

(i) $i_e(\mathcal{C}) > 0$ *and* $p - i_p(\mathcal{C}) > m - i_e(\mathcal{C})$

(ii) $1 < |\mathcal{C}| < |V|$

(iii) $\mathcal{C}$ *contains at least as many intracluster edges as any clustering* $\mathcal{C}'$ *of* G *into cliques.*

(iv) Any cluster in $\mathcal{C}$ *induces a connected subgraph.*

Proof. (i): If $i_e(\mathcal{C}) = 0$, it can easily be seen that $S(\mathcal{C}) = 1$. This similarly holds if $p - i_p(\mathcal{C}) = m - i_e(\mathcal{C})$, as $p - i_p(\mathcal{C})$ is the number of black balls in the urn and the statement that at least i_e of the drawn balls are white is equivalent to the statement that at most $m - i_e(\mathcal{C})$ of them are black. On the other hand, let us consider a clustering $\mathcal{C}'$ where each cluster contains one vertex, except for one cluster that contains two vertices linked by an edge e. As $m < p$, there is at least one labeled graph on V with m edges that does not contain e.

(ii): If $|\mathcal{C}| = 1$, $p - i_p(\mathcal{C}) = 0 = m - i_e(\mathcal{C})$ and if $|\mathcal{C}| = |V|$, $i_e(\mathcal{C}) = 0$. The statement now follows from (i).

(iii): Let us assume that $i_e(\mathcal{C}) < i_e(\mathcal{C}')$. Lemma 6.1(ii) can be used to show that $S(\mathcal{C}) = S\big(i_p(\mathcal{C}), i_e(\mathcal{C})\big) \geq S\big(i_e(\mathcal{C}), i_e(\mathcal{C})\big)$ and from Lemma 6.1(iii), it follows that $S\big(i_e(\mathcal{C}), i_e(\mathcal{C})\big) > S\big(i_e(\mathcal{C}'), i_e(\mathcal{C}')\big) = S(\mathcal{C}')$.

(iv): Follows from Lemma 6.1(ii) and the fact that splitting a disconnected cluster into its connected components decreases the number of intracluster pairs and does not affect the number of intracluster edges. □

Bicriteria View. From Lemma 6.1, it follows that an optimal solution with respect to surprise is *Pareto optimal* with respect to (maximizing) i_e and (minimizing) i_p. Interestingly, this also holds for a simplification of modularity whose null model does not take vertex degrees into account and that was briefly considered by Reichardt and Bornholdt [181, 182], although the tradeoff between the two objectives is different. Hence, an optimal clustering can be found by solving the following optimization problem for all $0 \leq k \leq m$ and choosing the solution that optimizes surprise.

Problem 1 (minIP)**.** *Given a graph G and an integer $k > 0$, find a clustering $\mathcal{C}$ with $i_e(\mathcal{C}) = k$, if there exists one, such that $i_p(\mathcal{C})$ is minimal.*

The problem minIP closely resembles the Minimum Average Contamination problem considered by Li and Tang [151] in the context of information propagation or virus diffusion.

Problem 2 (MACP)**.** *Given a graph $G = (V, E)$ together with a weight function $w\colon V \to \mathbb{Q}_{\geq 0}$ on V and a parameter k. Find a clustering $\mathcal{C}$ of G such that $m - i_e(\mathcal{C}) = k$ and $\sum_{C\in\mathcal{C}} \left(\sum_{v\in C} w(v)\right)^2$ is minimal.*

In the context of surprise, we are interested in the special case that $w(v) = 1$ for all $v \in V$, the unweighted MACP. The following conversion shows that this is equivalent to minIP with respect to optimal solutions:

$$i_p(\mathcal{C}) = \sum_{C\in\mathcal{C}} \frac{|C|\,(|C|-1)}{2} = \frac{1}{2}\sum_{C\in\mathcal{C}} |C|^2 - \underbrace{\frac{1}{2}\,|V|}_{=\text{const.}} \tag{6.1}$$

Unfortunately, the unweighted MACP and hence minIP is $\mathcal{NP}$-complete even on bipartite graphs [151]. Furthermore, Li and Tang give a proof that a weighted version of this problem is $\mathcal{NP}$-complete on planar graphs. In his bachelor thesis, Fleck [80] shows that this proof is not entirely correct, as it is based on a reduction that does not necessarily lead to a planar graph. However, Fleck shows how the reduction for the weighted case can be corrected and further, how it can be modified to also work for the unweighted MACP. Hence, minIP is $\mathcal{NP}$-complete as well, if restricted to planar graphs.

Although the connection between surprise and this subproblem does not lead to polynomial time algorithms for surprise optimization, the formulation of minIP does not involve binomial coefficients and is thus in some aspects easier to handle. For example, in contrast to surprise, it can be easily cast into an integer linear program. We will use this in Section 6.4 to compute optimal solutions for small instances.

One might guess from the $\mathcal{NP}$-completeness of minIP that surprise minimization is also $\mathcal{NP}$-complete. However, there is no immediate reduction from minIP to the decision variant of surprise optimization, as the number of intracluster edges in an optimal clustering with respect to surprise is not fixed. In the following section, we will therefore give a proof for the hardness of finding a clustering with optimal surprise.

Related Work. The problem minIP and the associated problem MACP seem basic enough to have been previously occurred in the literature. Indeed, there is a couple of problems in the context of clustering and partitioning that are at least closely related. Grötschel and Wakabayashi [108, 109] consider the problem to find a clustering of a complete graph with both positive and negative edge weights, such that the total weight of the intracluster edges is maximized, which they term the *clique partitioning problem* (CPP). They propose a cutting plane algorithm for this problem that can be used to solve instances of moderate size optimally. CPP is of particular interest, as it subsumes other well known optimization problems in the context of graph clustering as a special case. For example, correlation clustering [19] and cluster editing [196] are equivalent to a special case of CPP with respect to optimal solutions. Furthermore, modularity can be formalized as a CPP problem [10]. The problem minIP can be seen as a modification of CPP where all edge weights are uniform and we have the additional constraint that at least k of a special type of edges (which correspond to the edges in the original graph) are within clusters. This connection becomes apparent in Section 6.4, where we consider integer linear programs. For minIP or MACP, we might think of the weight of a cluster as the squared total sum of the weights of the vertices it contains. Other partitioning problems arise if we omit the square and just weigh a cluster by the total sum

of vertex weights. Especially in the case of trees, these problems have been extensively studied [24, 142, 179]. Ito et al. [124] give a linear time algorithm for a problem in this context on graphs of bounded tree width. There is no apparent way to translate these efficient algorithms to our setting. Furthermore, there exist linear algorithms for graphs with bounded tree width that work for whole classes of partitioning problems. For example, Voice et al. [218] consider *Coalition Structure Generation* (CSG), which seeks a clustering of the graph maximizing the total weight of the clusters. Here, it is assumed that the weight of each possible cluster is explicitly given, i.e., the input is a set of objects together with a function that maps each subset to a real number. CSG is even more general than CPP and clearly contains this problem as a special case. They show that if the weight function is *independent of disconnected members*, i.e., the contribution of a vertex to the weight of its cluster does not depend on the question whether the cluster contains some of its non-neighbors, a linear time algorithm exists for graphs of bounded tree width. It is not hard to see that this property does not hold in our setting. Similarly, there is no obvious way of fitting minIP into the framework of vertex partitioning problems defined by Telle and Proskurowski [211].

6.2 NP-Completeness

To show the hardness of surprise optimization, we prove $\mathcal{NP}$-completeness for the corresponding decision problem:

Problem 3 (Surprise Decision (SD))**.** *Given a graph G and a parameter $k > 0$, decide whether there exists a clustering $\mathcal{C}$ of G with $S(\mathcal{C}) \leq k$.*

To show that SD is in $\mathcal{NP}$, we have to justify that $S(\mathcal{C})$ can be evaluated in polynomial time on a Turing machine. The size of the largest binomial coefficient involved is $\binom{n^2}{m}$, which is bounded above by 2^{n^2}. In our standard model of computation, a RAM with uniform cost model, binomial coefficients of this size can be easily computed in $O(n^2 m)$, for example by using a dynamic algorithm which computes the relevant entries in Pascal's triangle. Unfortunately, the resulting numbers can have $\Omega(n^2)$ bits, which is not logarithmic in the size of the input; thus, it is not immediately clear that the computation can be emulated on a Turing machine in polynomial time. As explained in Section 2.4, if we switch from the uniform to the logarithmic cost model, the complexity is equivalent to Turing machines in the sense that every polynomial time algorithm can be transferred to a polynomial time algorithm on a Turing machine. As this switch only incurs an extra factor of n^2 in the running time, the running time is still polynomial and we get that SD is in $\mathcal{NP}$. To show $\mathcal{NP}$-completeness, we reduce from Exact Cover by 3-Sets, which is known to be $\mathcal{NP}$-complete [88].

Problem 3 (Exact Cover by 3-Sets (X3C))**.** *Given a set $\mathcal{X}$ with $|\mathcal{X}| = 3q$ and collection $\mathcal{S}$ of 3-element subsets of $\mathcal{X}$. Does $\mathcal{S}$ contain an exact cover for $\mathcal{X}$, i.e., a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ such that every $x \in \mathcal{X}$ occurs in exactly one $S \in \mathcal{S}'$?*

Let $I = (\mathcal{X}, \mathcal{M})$ be an instance of X3C. The reduction is based on the idea of implanting large disjoint cliques in the transformed instance that correspond to the subsets in $\mathcal{M}$. The size of these cliques is polynomial in $|\mathcal{M}|$, but large enough to ensure that they can neither be split nor merged in a clustering with low surprise. Hence, each of these cliques induces a cluster. The transformed instance further contains a vertex for each element in $\mathcal{X}$ that is linked with the cliques corresponding to subsets it is contained in.

The idea is to show that in a clustering $\mathcal{C}$ with low surprise, each of these vertices is contained in a cluster induced by exactly one subset, and each cluster contains either three "element vertices" or none, which induces an exact cover of $\mathcal{X}$.

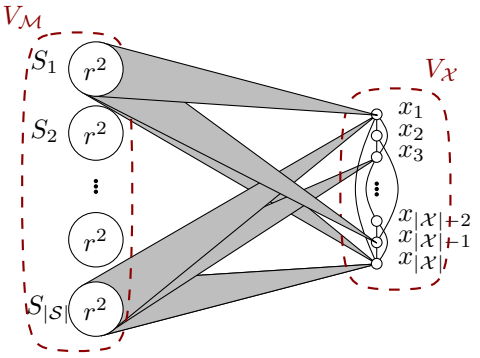


FIGURE 6.1: Illustration for reduction.

In the following, we will assume without loss of generality[3] that each element of $\mathcal{X}$ belongs to at least one set in $\mathcal{M}$. Hence, $|\mathcal{X}| \leq 3|\mathcal{M}|$. We construct an instance $I' = (G, k)$ of SD in the following way. Let $r := 3|\mathcal{M}|$. First, we map each set M in $\mathcal{M}$ to an r^2-clique $C(M)$ in G. Furthermore, we introduce an $|\mathcal{X}|$-clique to G, where each of the vertices $v(x)$ in it is associated with an element x in $\mathcal{X}$. We link $v(x)$ with each vertex in $C(M)$, if and only if x is contained in M. Let $V_{\mathcal{X}}$ be the set containing all vertices corresponding to elements in $\mathcal{X}$, and $V_{\mathcal{M}}$ the set of vertices corresponding to subsets. Figure 6.1 illustrates the reduction, clearly, it is polynomial. In the proof, we will frequently use the notion *for large r, statement $A(r)$ holds*. Formally, this is an abbreviation for the statement that there exists a constant $c > 0$ such that for all $r \geq c$, $A(r)$ is true. Consequently, the reduction only works for instances that are larger than the maximum of all these constants, which suffices to show that SD is $\mathcal{NP}$-complete[4].

Lemma 6.3. *Let $\mathcal{C}$ be an optimal clustering of G with respect to S. Then, $i_e(\mathcal{C}) \geq |\mathcal{M}| \cdot \binom{r^2}{2}$.*

Proof. Follows from Proposition 6.2(iii) and the fact that the clustering whose clusters are the cliques in $V_{\mathcal{M}}$ and the singletons in $V_{\mathcal{X}}$ is a clustering into cliques with $|\mathcal{M}| \cdot \binom{r^2}{2}$ intracluster edges. □

Next, we give an upper bound on the number of *intracluster non-edges*, i.e., vertex pairs within clusters that are not linked by an edge, in an optimal clustering of G. Its (rather technical) proof makes use of the asymptotic behavior of binomial coefficients and can be found in Section 6.7.

Lemma 6.4. *Let $\mathcal{C}$ be an optimal clustering of G with respect to surprise. Then, for large r, $i_p(\mathcal{C}) - i_e(\mathcal{C}) \leq \frac{r^4}{2}$.*

This can now be used to show that an optimal clustering of G is a clustering into cliques. We start by showing that the cliques in $V_{\mathcal{M}}$ cannot be split by an optimal clustering.

Lemma 6.5. *Let r be sufficiently large and $\mathcal{C}$ be an optimal clustering of G with respect to S. Then, the cliques $C(M)$ in $V_{\mathcal{M}}$ are not split by $\mathcal{C}$.*

Proof. Assume that there is at least one clique that is split by $\mathcal{C}$. $\mathcal{C}$ induces a partition of each clique that it splits. We call the subsets of this partition the *parts* of the clique.

Claim 1: Every clique $C(M)$ contains a part with at least $r^2 - 6$ vertices.

[3] Otherwise, the instance is trivially non-solvable.

[4] Smaller instances have constant size and can therefore be trivially solved by a brute-force algorithm.

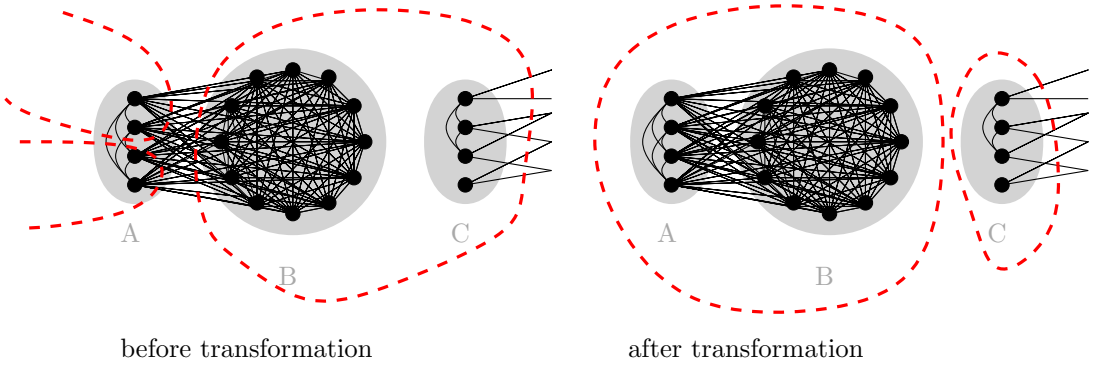


FIGURE 6.2: Illustration for proof of Lemma 6.5

Proof of Claim 1: Assume that there is a clique K where each part has at most $r^2 - 7$ vertices. We can now greedily group the parts in two roughly equal sized regions, such that the smaller region contains at least 7 vertices and the larger region at least $r^2/2$ vertices. Let us look at the clustering we get by removing the vertices in K from their clusters and cluster them together. The vertices in K have in total $3r^2$ edges to vertices outside K and we gain at least $7/2 \cdot r^2$ new intracluster edges between the regions. Hence, the number of intracluster edges increases and the number of intracluster non-edges can only decrease. By Lemma 6.1(iii) and Lemma 6.1(i), it can be seen that this operation leads to a clustering with better surprise, which contradicts the optimality of $\mathcal{C}$.

Let us now call the parts with size at least $r^2 - 6$ *large parts* and the other parts *small parts*.

Claim 2: No two large parts are clustered together.

Proof of Claim 2: Assume that there is a cluster that contains more than one large part. This cluster induces at least $(r^2 - 6)^2$ intracluster non-edges. For large r, this is larger than $r^4/2$ and Lemma 6.4 tells us that $\mathcal{C}$ was not optimal.

A simple counting argument now yields the following corollary.

Corollary: There must exist a large part B contained in a split clique whose cluster contains at most $|B| + 6$ vertices in $V_{\mathcal{M}}$.

Let B be as in the corollary and A be the set of the vertices that are in the same clique as B but not in B and C be the set of vertices that are in the same cluster as B but not in B. Figure 6.2 illustrates this case. We consider the clustering that we get by removing the vertices in A and B from their cluster and cluster them together. The number of vertices in A and C, respectively, is at most 6, and each of these vertices has at most 3 neighbors in $V_{\mathcal{X}}$. Hence, we lose at most 36 intracluster edges by this operation. On the other hand, we gain at least $r^2 - 6$ intracluster edges between A and B, thus, for large r, the number of intracluster edges increases. Again, the number of intracluster non-edges can only decrease and by Lemma 6.1(iii) and Lemma 6.1(i), we get that this operation leads to a clustering with better surprise, which contradicts the optimality of $\mathcal{C}$. □

Lemma 6.6. *Let r be large and $\mathcal{C}$ be an optimal clustering of G with respect to S. Then, no two of the cliques in $V_{\mathcal{M}}$ are contained in the same cluster.*

Proof. A cluster that contains two cliques in $V_{\mathcal{M}}$ induces at least r^4 intracluster non-edges. The statement now follows from Lemma 6.4. □

Lemma 6.7. *Let r be large and $\mathcal{C}$ an optimal clustering of G with respect to S. Then, each $v(x)$ in $V_{\mathcal{X}}$ shares a cluster with a clique $C(M)$ such that $x \in M$.*

Proof. From Lemma 6.5 and Lemma 6.6 we know that $\mathcal{C}$ clusters the vertices in $V_{\mathcal{M}}$ according to the cliques we constructed. Assume that there is a vertex $v(x)$ in $V_{\mathcal{X}}$ that is not contained in any of the clusters induced by the sets containing x. Since each element in $\mathcal{X}$ is contained in at least one set in $\mathcal{M}$, there exists a clique K in $V_{\mathcal{M}}$ that contains r^2 neighbors of $v(x)$. As $v(x)$ has at most $|\mathcal{X}| - 1$ neighbors in its own cluster, removing it from its cluster and moving it to the cluster of K increases the number of intracluster edges. On the other hand, x is linked with all vertices in its new cluster and thus, the number of intracluster non-edges cannot increase. Hence, this operation leads to a clustering with better surprise, which contradicts the optimality of $\mathcal{C}$. □

Theorem 6.8. *For large r, $I = (\mathcal{X}, \mathcal{M})$ has a solution if and only if there exists a clustering $\mathcal{C}$ of G with $S(\mathcal{C}) \leq k := \binom{p}{m}^{-1} \cdot \binom{\binom{|\mathcal{M}|\cdot r^2+|\mathcal{X}|}{2} - |\mathcal{M}|\cdot\binom{r^2}{2} - |\mathcal{X}|\cdot r^2 - |\mathcal{X}|}{(3|\mathcal{M}|-|\mathcal{X}|)\cdot r^2 + \binom{|\mathcal{X}|}{2} - |\mathcal{X}|}$.*

Proof. $\Rightarrow$: Let R be a solution of I. R induces a clustering of G in the following way: For each $M \in \mathcal{M} \setminus R$ we introduce a cluster $C_M = C(M)$ and for each $M' \in R$ a cluster $C_{M'} = C(M') \cup \{v(x) \mid x \in M'\}$. As R is an exact cover, this is a partition $\mathcal{C}$ of the vertex set. It is $p = \binom{|\mathcal{M}|\cdot r^2+|\mathcal{X}|}{2}$, $m = |\mathcal{M}| \cdot \binom{r^2}{2} + 3 \cdot |\mathcal{M}| \cdot r^2 + \binom{|\mathcal{X}|}{2}$ and $i_p(\mathcal{C}) = i_e(\mathcal{C}) = |\mathcal{M}| \cdot \binom{r^2}{2} + |\mathcal{X}| \cdot r^2 + |\mathcal{X}|$. It can be easily verified that $S(\mathcal{C}) = k$.

$\Leftarrow$: Let $\mathcal{C}$ be an optimal clustering of G with respect to surprise and assume that $S(\mathcal{C}) \leq k$. From Lemma 6.5, Lemma 6.6 and Lemma 6.7, we know that, for large r, we have one cluster for each set M in $\mathcal{M}$ that contains $C(M)$ and each vertex $v(x)$ in $V_{\mathcal{X}}$ shares a cluster with a clique $C(M)$ such that $x \in M$. In particular, all clusters in $\mathcal{C}$ are cliques and hence $\binom{i_p(\mathcal{C})}{i_e(\mathcal{C})} = 1$. It follows that $\binom{p}{m} \cdot k \geq \binom{p}{m} \cdot S(\mathcal{C}) = \binom{p-i_e(\mathcal{C})}{m-i_e(\mathcal{C})}$. This term is strictly decreasing with $i_e(\mathcal{C})$ and the above bound is tight for $i_e(\mathcal{C}) = |\mathcal{M}| \cdot \binom{r^2}{2} + |\mathcal{X}| \cdot r^2 + |\mathcal{X}| := t$. Hence, $\mathcal{C}$ contains at least t intracluster edges. The number of intracluster edges within $V_{\mathcal{M}}$ is exactly $|\mathcal{M}| \cdot \binom{r^2}{2}$ and the number of intracluster edges linking $V_{\mathcal{M}}$ with $V_{\mathcal{X}}$ is exactly $|\mathcal{X}| \cdot r^2$. The only quantity we do not know is the number of intracluster edges within $V_{\mathcal{X}}$, which we denote by $i_e(V_{\mathcal{X}})$. As $i_e(\mathcal{C}) \geq t$, it follows that $i_e(V_{\mathcal{X}}) \geq |\mathcal{X}|$. Thus, every vertex in $V_{\mathcal{X}}$ has in average two neighbors in $V_{\mathcal{X}}$ that are in the same cluster. On the other hand, vertices in $V_{\mathcal{X}}$ can only share a cluster if they are "assigned" to the same clique $C(M)$. As the sets in $\mathcal{M}$ only contain three elements, vertices in $V_{\mathcal{X}}$ can only have at most two neighbors in $V_{\mathcal{X}}$ in their cluster. It follows that $\mathcal{C}$ partitions $V_{\mathcal{X}}$ into triangles. Hence, the set of subsets R corresponding to cliques $C(M)$ whose clusters contain vertices in $V_{\mathcal{X}}$ form an exact cover of $\mathcal{X}$. □

We now have a reduction from X3C to SD that works for all instances that are larger than a constant $c > 0$. Hence, we get the following corollary.

Corollary 6.9. Surprise Decision *is $\mathcal{NP}$-complete.*

6.3 Trees and Bounded Treewidth

To show that an optimal clustering with respect to surprise can be found in polynomial time if G is a tree, we consider the above defined problem MACP [151]. For the special case that $w(v)$ equals the degree of v and G is a tree, Dinh and Thai give a dynamic program, based on *partial subtrees*, that solves MACP for all $0 \leq k \leq m$ simultaneously [66]. This yields an $O(n^5)$ algorithm for modularity maximization in (unweighted) trees.

The dynamic program of Dinh and Thai has a straightforward generalization to general vertex weights, which is polynomial in the case that each vertex has weight 1. A detailed description of the dynamic program in this special case, together with a runtime analysis, can be found in the bachelor thesis of Fleck [80] and in our technical report [81].

Theorem 6.10. *Let $T = (V, E)$ with $n := |V|$ be an unweighted tree. Then, a surprise optimal clustering of T can be calculated in $O(n^5)$ time.*

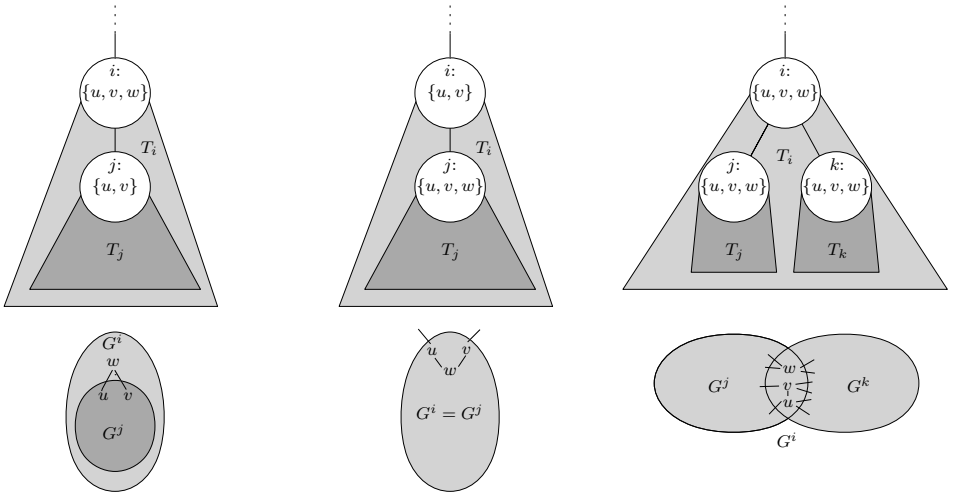
In the following, we will turn our attention again to the equivalent problem minIP. We give a polynomial time algorithm for graphs with bounded treewidth, i.e., graphs with treewidth at most w, where w is assumed to be constant. The dynamic program we use is similar in spirit to the program in the case of trees. To clarify the presentation, we will however abandon the idea to consider partial subtrees and will instead make use of the convenient concept of *nice tree decompositions* [137]. In the special case of trees, this dynamic program will add an additional factor of n to the running time; however, the main purpose of this section is to show that polynomial time algorithms exist.

To distinguish between vertices in the graph we want to cluster and in a tree decomposition thereof, we will always call the latter *nodes* instead of vertices. Let us assume in the following that tree decompositions contain an (arbitrarily chosen) node that we consider as the root of the tree, such that we can speak in terms of successors and ancestors of tree nodes.

Definition 6.11. A tree decomposition $\big(T = (I, F), \{X_i \mid i \in I\}\big)$ of a graph G is *nice*, if each node $i \in I$ is of one of the four following types:

1. *Leaf*: node i is a leaf of T, and $|X_i| = 1$
2. *Join*: node i has exactly two children, say j, and k and $X_i = X_j = X_k$
3. *Introduce*: node i has exactly one child, say j, and there is a vertex $w \in V$ with $X_j = X_i \setminus \{w\}$
4. *Forget*: node i has exactly one child, say j, and there is a vertex $w \in V$ with $X_j = X_i \cup \{w\}$

Given a (rooted) tree decomposition T of G, we will furthermore denote the subgraph of G that is induced by the vertices contained in the bags associated with the nodes in a subtree T_i of T, rooted at a node i, as G^i. It is not hard to see that if i is a join node, the vertices in X_i constitute a vertex separator of G^i [34]. Hence, the above node types join, introduce and forget, and the associated subtrees of G look like the examples in Figure 6.3. Note that we can restrict our considerations to nice tree decompositions without loss of generality, as these can be easily obtained from arbitrary tree decompositions of width w by the following lemma.



(a) Introduce (b) Forget (c) Join

FIGURE 6.3: Illustration of node types in a nice tree decomposition

Lemma 6.12 (Kloks [137], pages 149–150)**.** *For constant w, given a tree decomposition of a graph G of width w and $O(n)$ nodes, where n is the number of vertices of G, one can find a nice tree decomposition of G of width w and with at most $4n$ nodes in $O(n)$ time.*

Let i be a node in the tree decomposition and P be a partition of X_i. Furthermore, let s be a function that maps every $p \in P$ to an integer $0 \leq s(p) \leq n$. Let $\mathcal{C}$ be a clustering of G^i with a intracluster edges such that $\mathcal{C}$, restricted to the vertices in X_i, coincides with P and for each $p \in P$, the cluster containing the vertices in p has size $s(p)$. Then, we call $\mathcal{C}$ *eligible* with respect to a, P, and s. We now define $f^i(a, P, s)$ as the minimum number of intracluster pairs in any clustering of G^i that is eligible with respect to a, P, and s.

Let r be the root of a tree decomposition of G, $\mathcal{P}$ be the set of partitions over X^r and $\mathcal{S}$ the set of all possible functions s as defined above. It is now easy to see that the value of an optimal solution with respect to minIP with parameter k can be computed as

$$\min_{P \in \mathcal{P}, s \in \mathcal{S}} f^r(k, P, s)$$

It remains to show how all relevant table entries for the nodes in the tree decomposition can be computed from the tables of their children. If we can do that, the tables at the root node can be determined by a dynamic program that considers the nodes in T in the order given by a post-order traversal of T. We distinguish by the different types of nodes.

Leaf. It is immediate that if i with $X_i = \{v\}$ is a leaf,

$$f^i(a, P, s) = \begin{cases} 0 & a = 0, P = \{\{v\}\}, s(\{v\}) = 1 \\ \infty & \text{otherwise} \end{cases}$$

Introduce. Let i be an introduce node, j its successor in T such that $X_i \setminus X_j = \{w\}$, $P(w)$ be the part of P containing w and P' the partition of X_j that arises from deleting w from $P(w)$. We define s' such that $s'(p) = s(p)$ if $p \in P$ and $s'(p) = s(p) - 1$ if $p = P(w) \setminus \{v\}$. Let $\mathcal{C}$ be an eligible clustering of G^i with respect to a, P, and s, and let $\mathcal{C}'$ denote its restriction to G^j. If l denotes the number of neighbors of w inside $P(w)$, there are exactly $a - l$ intracluster edges in the subgraph G^j. Furthermore, s' maps each part of P' to the number of vertices in the associated clusters in $\mathcal{C}'$. Hence, $\mathcal{C}'$ is eligible with respect to $a - l$, P', and s'. On the other hand, it is easy to see that any eligible clustering of G^j with respect to $a - l$, P' and s' can be extended to an eligible clustering of G^i with respect to a, P and s such that the number of intracluster pairs increases by $s\big(P(w)\big) - 1$. Hence,

$$f^i(a, P, s) = f^j(a - l, P', s') + s\big(P(w)\big) - 1$$

Forget. Let i be a forget node and j its successor in T such that $X_j \setminus X_i = \{w\}$. Here, G^i is the same as G^j and thus, every clustering of G^i is a clustering of G^j and vice versa. Hence, the only difference between the entries associated with i and j is that we *forget* which of the components contains w. Let P be a partition of X_i. We have to distinguish two cases: either w is contained in one of the clusters that contain a part in P or the cluster of w does not contain any vertex in X_i. In both cases, we can extend P to a partition of X_j by defining for all $p \in P$ a partition P^p that arises from P by substituting p by $p \cup \{w\}$ (including the possibility that w is in an extra part, i.e., $p = \emptyset$). For the former case, we define s^p for a given $p \in P$ such that $s^p(p \cup \{w\}) = s(p)$ and $s^p(q) = s(q)$ for all $q \neq p \in P$. In the second case, we have to distinguish by the size of the cluster containing w. For $1 \leq r \leq n$, let s^r be such that $s^r(p) = s(p)$ for $p \in P$ and $s^r(\{w\}) = r$. In total, we get

$$f^i(a, P, s) = \min\big\{\min_{p \in P} f^j(a, P^p, s^p), \min_{1 \leq r \leq n} f^j(a, P^\emptyset, s^r)\big\}$$

Join. The most expensive part will be the computation of the tables associated with join nodes. Let i be a join node with children j and k and $\mathcal{C}$ an eligible clustering of G^i with respect to a, P, and s. $\mathcal{C}$ can be decomposed into two clusterings $\mathcal{C}^j$ and $\mathcal{C}^k$ of G^j and G^k. If p is a part of P, then, the sum of the number of vertices in the cluster of $\mathcal{C}^j$ containing p and the number of vertices in the cluster of $\mathcal{C}^k$ containing p equals $s(p) + |p|$, as we count the vertices in p twice. Hence, if s^j indicates the size of the clusters in $\mathcal{C}^j$ and s^k the size of the clusters in $\mathcal{C}^k$, $s^j(p) + s^k(p) = s(p) + |p|$ for all p. Similarly, if l is the number of intracluster edges between vertices in X_i, the number of intracluster edges a^j in G^j and the number of intracluster edges a^k in G^k sum up to $a + l$. On the other hand, every two clusterings $\mathcal{C}^j$ and $\mathcal{C}^k$ that are eligible with respect to s^j, s_k, a^j and a^k such that these parameters adhere to the above equations can be extended to an eligible clustering of $\mathcal{C}$ in a straightforward way. Furthermore, summing up the number of intracluster pairs in $\mathcal{C}^j$ and $\mathcal{C}^k$ counts the intracluster pairs in X_i twice and does not account for intracluster pairs (v_j, v_k) such that $v_j \in G^j$, $v_k \in G^k$ and $v_j, v_k \notin X_i$. Hence,

$$f^i(a, P, s) = \min_{s^j, s^k, a^j, a^k} f^j(a^j, P, s^j) + f^k(a^k, P, s^k) - \sum_{p \in P} \binom{|p|}{2} + \sum_{p \in P} \big(s^k(p) - |p|\big) \cdot \big(s^j(p) - |p|\big)$$

Theorem 6.13. *Given a nice tree decomposition of a graph G of width w containing at most $4n$ nodes, a clustering of G that is optimal with respect to surprise can be found in $O(w^{cw} \cdot n^{2w+4})$ time, for some constant c.*

Proof. It is not hard to see that the most expensive updates correspond to the tables associated with join nodes. Given a, P and s, there are at most n^{w+1} relevant choices of s^j and at most m choices for a_j; if s^j and a^j are given, this uniquely determines s^k and a^k. Hence, the time to compute a table entry of a join node can be bounded above by $O(n^{w+1} \cdot m)$. If $B(k)$ denotes the number of distinct partitions of a k-element set, the size of the table for each node in the tree decomposition is bounded above by $B(w+1) \cdot n^{w+1}$. As we have at most $4n$ nodes in the tree decomposition, we get a total running time in $O(n \cdot B(w+1) \cdot n^{2w+3} \cdot m)$ for computing all relevant table entries. As G is a partial k-tree, m is bounded above by wn (cf. page 13 of Kloks [137]) and hence, this is in $O(w^{cw} \cdot n^{2w+4})$ for some constant c. As explained above, the optimal solution of minIP with parameter k can then be found by parsing the entries of the root node; the value of an optimal clustering with respect to surprise is then given by

$$\min_{k \in [0,m]} \mathcal{S}\big(\min_{P,s} f^r(k, P, s), t\big)$$

With additional bookkeeping, the above program can be modified such that it not only computes the value of an optimal clustering, but the clustering itself. □

In summary, as it is possible to construct a tree decomposition of width w in linear time if G is a partial k-tree [33] and k is constant, there is a polynomial time algorithm that finds a clustering with optimum surprise in graphs of bounded tree-width. It is not too hard to see that the above dynamic program can be modified to solve the special case of MACP where each vertex is weighted with its own degree. As discussed by Dinh and Thai [66], this translates to a polynomial time algorithm for modularity optimization for unweighted graphs, which yields the following corollary

Corollary 6.14. *There is a polynomial time algorithm that solves modularity optimization for unweighted graphs of bounded tree width.*

It is needless to say that the constants involved in the above runtime analysis are quite high and hence, this result is more of theoretical interest instead of yielding a blue print for an efficient implementation. Hence, in order to obtain optimal solutions for small instances, the next section describes an approach based on integer linear programming.

6.4 Exact Solutions for General Graphs

In this section, we give an integer linear program for minIP and discuss some variants of how to use it to get optimal clusterings with respect to surprise.

Integer Linear Program for minIP. The following ILP is very similar to a number of linear programs used for other objectives in the context of graph clustering and partitioning, in particular, to one used for modularity maximization [66]. It uses a set of $\binom{n}{2}$ binary variables $\mathcal{X}_{uv}$ corresponding to vertex pairs, with the interpretation that

$\mathcal{X}_{uv} = 1$ iff u and v are in the same cluster. Let $\mathrm{Sep}(u, v)$ be a minimum u-v vertex separator in G if $\{u, v\} \notin E$ or in $G' = (V, E \setminus \{u, v\})$, otherwise. The objective is to

$$\text{minimize} \sum_{\{u,v\} \in \binom{V}{2}} \mathcal{X}_{uv} \tag{6.2}$$

such that

$$\mathcal{X}_{uv} \in \{0, 1\}, \quad \{u, v\} \in \binom{V}{2} \tag{6.3}$$

$$\mathcal{X}_{uw} + \mathcal{X}_{wv} - \mathcal{X}_{uv} \leq 1, \quad \{u, v\} \in \binom{V}{2}, w \in \mathrm{Sep}(u, v) \tag{6.4}$$

$$\sum_{\{u,v\} \in E} \mathcal{X}_{uv} = k \tag{6.5}$$

Dinh and Thai consider the symmetric and reflexive relation induced by $\mathcal{X}$ and show that Constraint (6.4) suffices to enforce transitivity in the context of modularity maximization [66]. Their proof solely relies on the following argument. For an assignment of the variables $\mathcal{X}_{uv}$ that does not violate any constraints, let us consider the graph G' induced by the vertex pairs $\{u, v\}$ with $\mathcal{X}_{uv} = 1$. Now assume that there exists a connected component in G' that can be partitioned into two subsets A and B such that there are no edges in the original graph G between them. Setting $\mathcal{X}_{ab} := 0$ for all $a \in A$, $b \in B$ never violates any constraints and strictly improves the objective function. It can be verified that this argument also works in our scenario. Hence, a solution of the above ILP induces an equivalence relation and therefore a partition of the vertex set. As $\mathrm{Sep}(u, v)$ is not larger than the minimum of the degrees of u and v, we have $O(nm)$ constraints over $O(n^2)$ variables.

Variants. We tested several variants of the approach described in Section 6.1 to decrease the number of ILPs we have to solve.

- *Exact(E)*: Solve m times the above ILP and choose among the resulting clusterings the one optimizing surprise.
- *Relaxed(R)*: We relax Constraint (6.5), more specifically we replace it by

  $$\sum_{\{u,v\} \in E} \mathcal{X}_{uv} \geq k \tag{6.6}$$

  Lemma 6.1(i) tells us that the surprise of the resulting clustering is at least as good as the surprise of any clustering with exactly k intracluster edges. Moreover, by Lemma 6.1(ii), if i_p is the value of a solution to the modified ILP, $S(i_p, k')$ is a valid lower bound for the surprise of any clustering with $k' \geq k$ intracluster edges. In order to profit from this, we consider all possible values for the number of intracluster edges in increasing order and only solve an ILP if the lower bound is better than the best solution found so far.
- *Gap(G)*: Similarly to the relaxed variant, we replace Constraint (6.5) by (6.6) and modify (6.2) to

  $$\text{minimize} \sum_{\{u,v\} \in \binom{V}{2}} \mathcal{X}_{uv} - \sum_{\{u,v\} \in E} \mathcal{X}_{uv} \tag{6.7}$$

TABLE 6.1: Number of linear programs solved and running times in seconds of successive ILP approach, different strategies.

	karate		grid6		dolphins		lesmis	
variant	ILP	t(s)	ILP	t(s)	ILP	t(s)	ILP	t(s)
Exact	79	51	61	470	160	494	255	1192
Relaxed	49	21	42	449	107	163	176	282
Gap	39	**15**	37	**401**	91	**147**	112	**205**

By Lemma 6.1(ii), if g is the objective value and i_e the number of intracluster edges in a solution to the modified ILP, $S(k'+g, k')$ is a valid lower bound for the surprise of any clustering with $k' \geq k$ intracluster edges. Moreover, by Lemma 6.1(iii), we know that $S(i_e+g, i_e)$ is not larger than the surprise of any clustering with exactly k intracluster edges. Again, we consider all k in increasing order and try to prune ILP computations with the lower bound.

Case Study.

Table 6.1 shows an overview of running times and the number of solved ILPs of the different strategies on some small instances. `karate`($n = 34, m = 78$), `dolphins`($n = 62, m = 159$) and `lesmis`($n = 77, m = 254$) are real world networks from the website of the 10th DIMACS implementation Challenge[5] that have been previously used to evaluate and compare clusterings, whereas `grid6`($n = 36, m = 60$) is a 2 dimensional grid graph. We used the C++-interface of `gurobi5.1` [114] and computed the surprise of the resulting clusterings with the help of the GNU Multiple Precision Arithmetic Library, in order to guarantee optimality. The tests were executed on one core of an AMD Opteron Processor 2218. The machine is clocked at 2.1 GHz and has 16 GB of RAM. Running times are averaged over 5 runs.

It can be seen that the gap variant, and, to a smaller extent, the relaxed variant, are able to prune a large percentage of ILP computations and thus lead to less overall running time. These running times can be slightly improved by using the heuristic modifications described and evaluated in the following section.

Heuristics for Linear Programs. We tried the following modifications to further decrease the running time to compute exact solutions:

- *Prune small k (PSK)*: We first determine the clustering into cliques that maximizes the number k_{start} of intracluster edges. This can be done by dropping (6.5), substituting (6.2) by

$$\text{maximize} \sum_{\{u,v\} \in E} \mathcal{X}_{uv} \tag{6.8}$$

and setting $\mathcal{X}_{uv} = 0$ for all vertex pairs $\{u, v\}$ not connected by an edge. Proposition 6.2(iii) then yields that we do not have to consider clusterings with less than k_{start} intracluster edges. This is in fact a special case of the gap variant, but as solving the modified ILP is usually very fast, its usage potentially decreases the overall running time for all variants.

[5] http://www.cc.gatech.edu/dimacs10/

TABLE 6.2: Running times in seconds of successive ILP approach, different strategies.

				karate		grid6		dolphins		lesmis	
var	TF	PSK	EMI	ILP	t(s)	ILP	t(s)	ILP	t(s)	ILP	t(s)
e	n	n	n	79	51	61	470	160	497	255	1194
e	n	y	n	54	50	43	470	104	489	119	1149
e	y	n	n	79	15	61	689	160	164	255	315
e	y	y	n	54	14	43	684	104	152	119	272
r	n	n	n	49	21	42	449	107	163	176	283
r	n	n	y	49	31	42	2091	107	383	176	373
r	n	y	n	25	20	25	448	52	158	41	254
r	n	y	y	25	30	25	2091	52	378	41	353
r	y	n	n	78	15	60	1154	159	218	254	302
r	y	n	y	78	15	60	1698	159	537	254	354
r	y	y	n	54	15	43	1129	104	212	119	264
r	y	y	y	54	14	43	1700	104	539	119	323
g	n	n	n	39	15	37	402	91	147	112	206
g	n	n	y	18	18	15	1904	45	131	23	165
g	n	y	n	20	15	20	**398**	50	143	29	186
g	n	y	y	18	18	15	1896	45	131	23	**162**
g	y	n	n	73	15	56	1774	144	168	195	259
g	y	n	y	52	**12**	38	2224	100	110	107	221
g	y	y	n	54	14	39	1724	103	160	112	245
g	y	y	y	52	**12**	38	2214	100	**108**	107	222

- *Testing for Feasibility (TF)*: From the value S of the best current solution, we can compute for each k the largest i_p such that $S(i_p, k) < S$. This can be modeled as an additional constraint; if this makes the model infeasible, we can safely proceed to the next k. The downside of this approach is that the lower bounds for the gap and relaxed variant are updated less often. However, it potentially decreases the time to solve individual ILPs in case the model is not feasible.

- *Enforce many intraedges (EMI)*: To enforce that the clustering we obtain by the linear program for the relaxed variant has the most intracluster edges among all valid clusterings that minimize the number of intracluster pairs, and therefore yields the best upper bound, we replace (6.2) by

$$\text{minimize} \quad m \cdot \left(\sum_{\{u,v\} \in \binom{V}{2}} \mathcal{X}_{uv} \right) - \sum_{\{u,v\} \in E} \mathcal{X}_{uv} \tag{6.9}$$

  Similarly, for the gap variant, we replace (6.7) by

$$\text{minimize} \quad m \cdot \left(\sum_{\{u,v\} \in \binom{V}{2}} \mathcal{X}_{uv} - \sum_{\{u,v\} \in E} \mathcal{X}_{uv} \right) - \sum_{\{u,v\} \in E} \mathcal{X}_{uv} \tag{6.10}$$

  Obviously, this does not make sense for the exact variant.

Table 6.2 shows an overview of running times and the number of solved ILPs of the different strategies on the test instances from Section 6.4.

In almost all cases, the PSK heuristic is able to decrease the running time slightly.

TABLE 6.3: Properties of optimal clusterings with respect to surprise. S' denotes the surprise as defined by Aldecoa and Marín [5], i.e., $S'(\mathcal{C}) = -\log_{10} S(\mathcal{C})$. S_o denotes the clustering with optimum surprise, S_h the heuristically found clusterings from [5], if this information was available, and M_o the modularity optimal clustering.

instance	i_e	i_p	$S(S_o)$	$S'(S_o)$	$S'(S_h)$	$\lvert S_o\rvert$	$\lvert S_h\rvert$	$\lvert M_o\rvert$
`karate`	29	30	$2{,}02 \cdot 10^{-26}$	25.69	25.69	19	19	4
`grid6`	36	54	$2{,}90 \cdot 10^{-29}$	28.54	-	9	-	4
`dolphins`	87	121	$9{,}93 \cdot 10^{-77}$	76.00	-	22	-	5
`lesmis`	165	179	$1{,}54 \cdot 10^{-184}$	183.81	-	33	-	6
`football`	399	458	$5{,}65 \cdot 10^{-407}$	406,25	-	15	15	10

Enforcing many intracluster edges always increased the running time of the relaxed variant; the average running time for each linear program increases and in none of our examples it helped to decrease their number. For the gap variant, this modification was beneficial in most cases. However, for `grid6`, the running time increased by almost a factor of five compared to the gap variant without modifications.

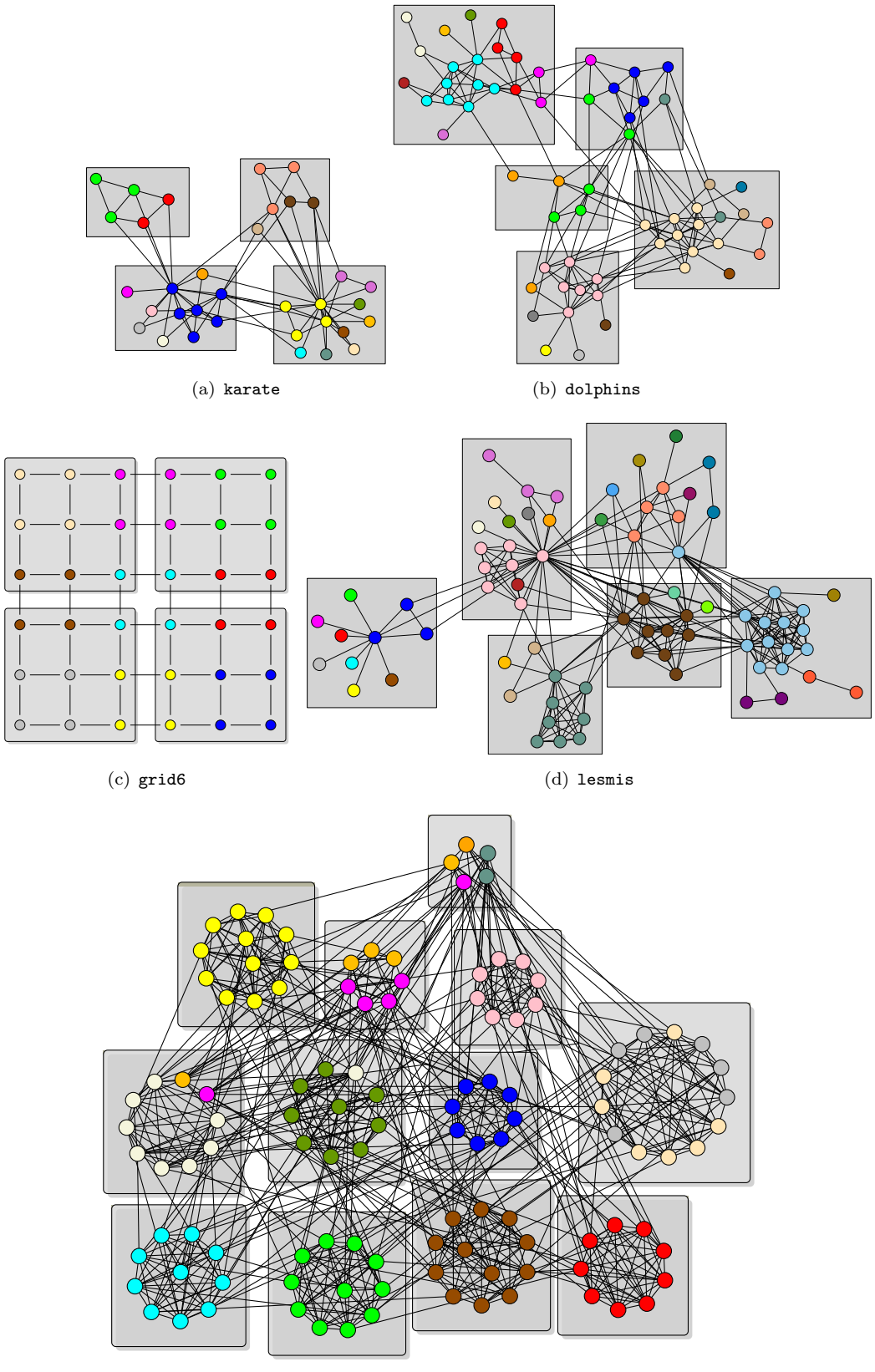
Similarly, testing for feasibility is beneficial in combination with the gap and relaxed variant in about half of the cases, but on `grid6`, it increases their running time significantly.

Overall, the unmodified version of the gap variant was always faster than any version of the relaxed or exact one. Among the versions of the gap variant, the one that uses only PSK and the one with all modifications exhibit good overall behavior, while the former seems to be more robust.

Properties of optimal clusterings. Figure 6.4 illustrates optimal clusterings with respect to surprise and modularity on the test instances, Table 6.3 summarizes some of their properties. We also included one slightly larger graph, `football`($n = 115, m = 613$), as it has a known, well-motivated *ground truth clustering* and has been evaluated in [5]. The surprise based clusterings contain significantly more and smaller clusters than the modularity based ones, being *refinements* of the latter in the case of `karate` and `lesmis`. Another striking observation is that the surprise based clusterings contain far more *singletons*, i.e., clusters containing only one vertex with usually low degree; this can be explained by the fact that surprise does not take vertex degrees into account and hence, merging low degree vertices into larger clusters causes larger penalties. It reconstructs the ground-truth clustering of the `football` graph quite well. This confirms the observations of Aldecoa and Marín based on heuristically found clusterings [5]; in fact, we can show that for `karate`, this clustering was already optimal.

6.5 Combining Surprise and Modularity

Clearly, modularity and surprise build upon the same basic idea, the comparison of the observed number of intracluster edges in a given graph G and the number of intracluster edges in a random graph G' on the same vertex set. However, they differ in two aspects:



(a) `karate` (b) `dolphins`

(c) `grid6` (d) `lesmis`

(e) `football`

FIGURE 6.4: Optimal clusterings with respect to surprise(colors) and, for (a) to (d), modularity(grouping). The grouping in (e) represents the *ground-truth clustering*, i.e., the mapping of teams to conferences.

the choice of the random graph G' and the tradeoff between the intracluster edges in G and G'.

In both cases, G' retains some of the properties of G. In the context of surprise, this is only the number of edges, whereas the random graph used by modularity additionally keeps expected vertex degrees. The former leads to a random graph following Gilbert's model [89], and the latter yields the configuration null model (cf. Section 2.3). A more subtle difference arises from the concrete random process that generates G'; in contrast to the configuration null model, the assumption in Gilbert's model is that vertices and edges are drawn without replacement. Hence, the configuration null model allows for loops and parallel edges, whereas each instantiation of Gilbert's model is simple. It is worth mentioning that variants of the configuration model have been proposed that avoid loops [44, 95] or parallel edges [44]. A drawback of these variants is that they lose the property that the expected degree of each vertex equals its observed degree in G.

The second distinction between modularity and surprise is that modularity evaluates the difference between the expected and the observed number of intracluster edges, whereas surprise seeks to minimize the probability that the number of intracluster edges in G' exceeds the number of intracluster edges in G. Although these two quantities are closely related, the corresponding optimum solutions will differ.

These two aspects are orthogonal in the sense that they can be arbitrarily combined. Combining the tradeoff of modularity with Gilbert's model as null model yields the above mentioned objective function[6] that has been considered by Reichardt and Bornholdt [181, 182]. To the best of our knowledge, combining the tradeoff of surprise with the configuration null model has not been considered before, although this idea suggests itself. In this section, we will derive the formula of the resulting objective function, which we call *S*Mod, and compare the resulting clusterings to the ones obtained by optimizing modularity and surprise, using our small example graphs.

Derivation of SMod. The configuration model can be explained by a simple random process that transforms G into a random graph G'. In a first step, we remove all edges from G, but remember the degree of each vertex. In a second step, the edges are redistributed at random. Each edge draws two endvertices independently, with replacement, such that vertices are chosen with a probability proportional to their degree. This means, the probability to draw vertex u is equal to $\frac{d_u}{2m}$. Hence, as we work with undirected edges, the probability that the edge links two vertices u and v is $\frac{d_u d_v}{2m^2}$, if $u \neq v$, and $\frac{{d_u}^2}{4m^2}$ otherwise. It follows that the total probability that the edge links vertices in the same cluster equals

$$p = \sum_{C \in \mathcal{C}} \left[\sum_{\{u,v\} \in \binom{C}{2}} \frac{d_u d_v}{2m^2} + \sum_{u \in C} \frac{{d_u}^2}{4m^2} \right] = \frac{1}{4m^2} \underbrace{\sum_{C \in \mathcal{C}} {v_C}^2}_{:= i_{v^2}(\mathcal{C})}$$

[6] Note that it does not matter here if edges are drawn with or without replacement, as the expected number of edges between each pair of vertices is $m/\binom{n}{2}$ in both cases.

In total, we draw m edges independently. Thus, the probability that we draw at least $i_e(\mathcal{C})$ edges within clusters follows a cumulative binomial distribution:

$$SMod(\mathcal{C}) = \sum_{i=i_e}^{m} \left[\binom{m}{i} p^i (1-p)^{m-i} \right] = \sum_{i=i_e}^{m} \left[\binom{m}{i} \cdot \left(\frac{i_{v^2}}{4m^2} \right)^i \cdot \left(1 - \frac{i_{v^2}}{4m^2} \right)^{m-i} \right]$$

Similarly to surprise, SMod only depends on two properties of $\mathcal{C}$, the number of intracluster edges i_e and the *total sum of squared volumes* i_{v^2}. Hence, we can denote the value of a clustering $\mathcal{C}$ with $i_{v^2}(\mathcal{C}) = i_{v^2}$ and $i_e(\mathcal{C}) = i_e$ by $SMod(i_{v^2}, i_e)$.

Optimal Solutions. With analogous arguments as we used for Lemma 6.1, the probabilistic view on SMod yields that an optimal clustering with respect to SMod is always Pareto optimal with respect to (maximizing) i_e and (minimizing) i_{v^2}. Note that the same holds trivially for clusterings that are optimal with respect to modularity; the difference to SMod is that the number of clusters may differ, i.e., we get clusterings of a different *resolution*. This is especially interesting in the light of the resolution limit of modularity (cf. Section 2.3). Although we do not claim that SMod, being a fundamentally global measure, circumvents the resolution limit in the strong sense, it yields another well motivated resolution scale in the context of multiresolution modularity [182] that potentially leads to smaller clusters in practice.

To compute optimal solutions with respect to SMod, we can exploit this bicriteria view on SMod and optimize i_{v^2} for all possible values of i_e, similar to the linear programs used for surprise optimization. Using the same set of variables $\mathcal{X}_{uv}$ as above, the objective i_{v^2} can be easily expressed as

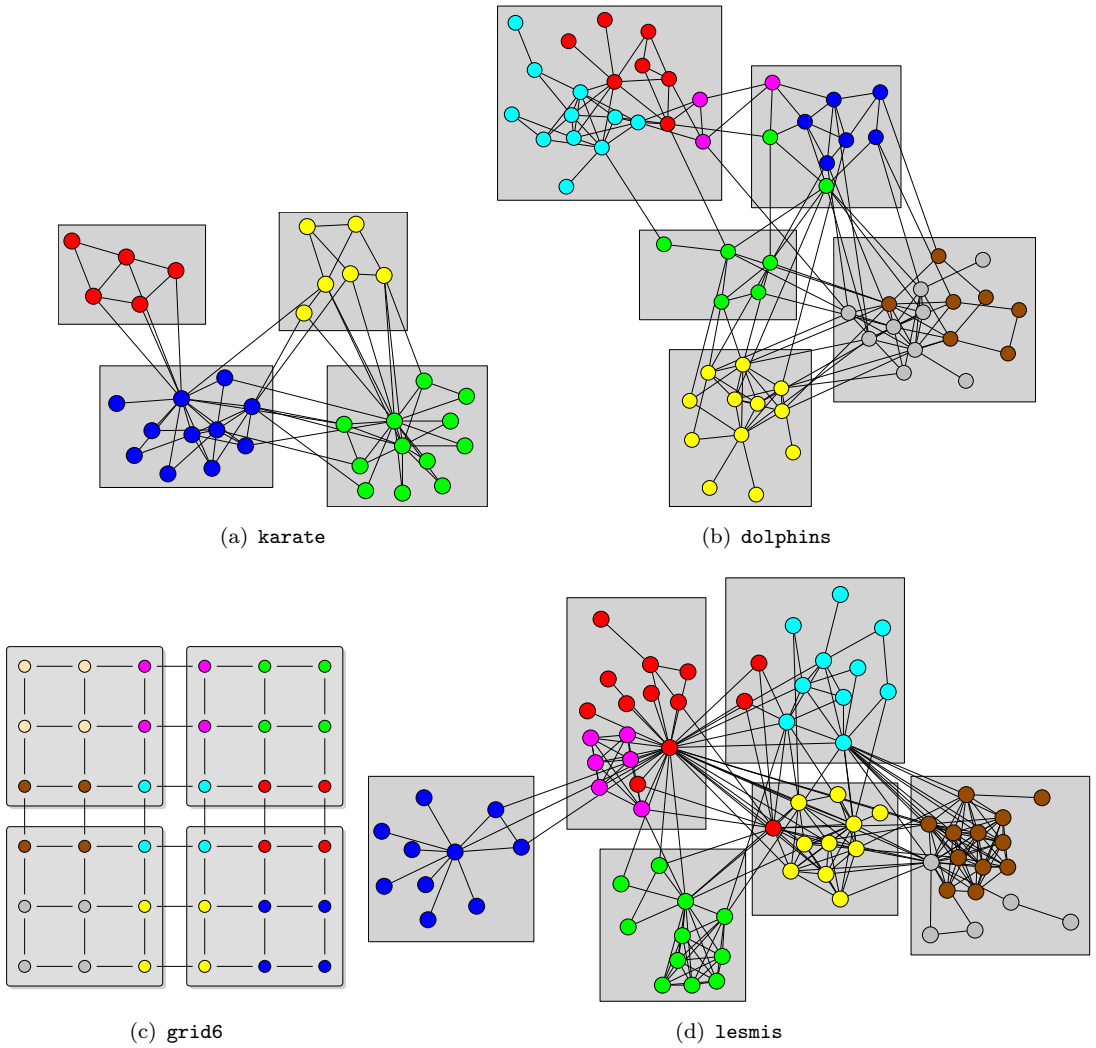
$$i_{v^2}(\mathcal{C}) = \sum_{C \in \mathcal{C}} {v_C}^2 = \sum_{C \in \mathcal{C}} \left[2 \sum_{\{u,v\} \in \binom{C}{2}} d_u d_v + \sum_{v \in C} {d_v}^2 \right] = \sum_{\{u,v\} \in \binom{V}{2}} 2 d_u d_v \mathcal{X}_{uv} + \sum_{v \in V} {d_v}^2$$

Note that using $i_{v^2}(\mathcal{C})$ as objective allows for the reduction of transitivity constraints to vertices in a minimum vertex separator.

We refrain from any engineering steps for SMod and compute optimal solutions similarly to the relaxed variant above, which stays correct in this modified scenario. Figure 6.5 illustrates clusterings that are optimal with respect to SMod on our small test instances. For `dolphins` and `lesmis`, the clusterings are similar to the clusterings based on modularity, with the difference that some clusters of the latter are split in the clusterings obtained by optimizing SMod. The clusterings obtained for `karate` are identical, whereas the clustering of `grid6` is identical to the clustering obtained by optimizing surprise. In all cases, we get at least as many clusters as in the clustering based on modularity; in contrast to surprise, the cluster sizes are more evenly distributed and all vertices belong to nontrivial clusters.

6.6 Concluding Remarks

We showed that the problem of finding a clustering of a graph that is optimal with respect to the measure surprise is $\mathcal{NP}$-hard. The observation that surprise is Pareto optimal with respect to (maximizing) the number of edges and (minimizing) the number



(a) `karate` (b) `dolphins`

(c) `grid6` (d) `lesmis`

FIGURE 6.5: Optimal clusterings with respect to SMod(colors) and modularity(grouping).

of vertex pairs within clusters yields a (polynomial time) dynamic program on graphs with bounded tree width. Furthermore, it helps to find exact solutions in small, general graphs via a sequence of ILP computations. The latter can be used to gain insights into the behavior of surprise, independent of any artifacts stemming from a particular heuristic. Moreover, optimal solutions are helpful to assess and validate the outcome of heuristics. Furthermore, we derived a new measure called SMod, which combines the basic idea behind modularity with the configuration null model used by modularity. On our test instances, optimizing SMod led to clusterings that are similar to modularity, but exhibit smaller clusters.

There are several interesting directions for future work. From a theoretic point of view, a self-evident follow-up question concerns the approximability of surprise maximization. One possibility is to derive approximation factors from approximate solutions of the subproblem minIP. Li and Tang [151] show that the equivalent problem MACP is $\mathcal{APX}$-hard, implying that we most probably cannot build upon a PTAS for minIP. On the other hand, they give a bicriteria approximation algorithm that finds a clustering whose

value is at most $(1 + \varepsilon)$ times the optimum and that cuts at most $O(\frac{1+\epsilon}{\epsilon} \log n) \cdot k$ edges. Due to the nature of the objective function of surprise, it is however unclear how this result or any other approximation guarantee for minIP translates to an easy to interpret approximation guarantee for surprise maximization. Another open problem is to fortify the empirical observation that surprise and *S*Mod lead to smaller clusters by investigating the asymptotic growth of the cluster sizes in optimal clusterings of graph families as the ring of cliques considered by Fortunato and Barthelémy [84], grids, or random graphs.

From a more practical point of view, it would be interesting to further engineer the linear programs we used in Section 6.4. Preliminary experiments based upon a direct quadratic formulation of surprise optimization using precomputed values did not achieve good results. Similarly, a combinatorial branch-and-bound approach branching on the question whether two vertices are contained in the same cluster and using straightforward bounds on the number of intracluster edges and pairs turned out to be slower than the algorithms described in this chapter. Hence, to benefit from such an approach, it is necessary to find more elaborate bounds or branching rules. Other promising candidates to replace or enhance the linear program we used to solve minIP are more advanced cutting plane and column generation algorithms. These turned out to be very successful with respect to related clustering and partitioning problems [108, 109, 122, 126]; in particular, the latter approach yields the currently state-of-the art algorithm to find clusterings with provably optimal modularity [10]. However, there is little hope to engineer this approach to also work for large instances typically occurring in many fields of application. Hence, there is a need for fast algorithms that efficiently find clusterings of good quality. To some extent, we close this gap in the following chapter.

6.7 Proof of Lemma 6.4

The proof of Lemma 6.4 is based on the following two observations on the asymptotic behavior of binomial coefficients.

Lemma 6.15. *Let $f\colon \mathbb{N} \to \mathbb{N}$ and $g\colon \mathbb{N} \to \mathbb{N}$ be two functions such that $g(n) \in o\,(f(n))$. Then,*

$$\binom{f(n)}{g(n)} \in \Omega\left(\frac{f(n)^{g(n)}}{\sqrt{g(n)} \cdot g(n)^{g(n)}}\right)$$

Proof. For $n > 0$, Stirling's formula yields

$$\sqrt{2\pi} \cdot n^{n+1/2} \cdot e^{-n} \leq n! \leq e \cdot \sqrt{2\pi} \cdot n^{n+1/2} \cdot e^{-n}$$

Hence, it is

$$\begin{aligned}\binom{f(n)}{g(n)} &= \frac{f(n)!}{g(n)! \cdot [f(n)-g(n)]!} \\ &\geq \frac{\sqrt{2\pi} \cdot f(n)^{f(n)} \cdot \frac{\sqrt{f(n)}}{e^{f(n)}}}{(e \cdot \sqrt{2\pi})^2 \cdot g(n)^{g(n)} \cdot \frac{\sqrt{g(n)}}{e^{g(n)}} \cdot [f(n)-g(n)]^{f(n)-g(n)} \cdot \frac{\sqrt{f(n)-g(n)}}{e^{f(n)-g(n)}}} \\ &= \frac{1}{e^2 \cdot \sqrt{2\pi}} \cdot \frac{f(n)^{f(n)} \cdot \sqrt{f(n)}}{g(n)^{g(n)} \cdot \sqrt{g(n)} \cdot [f(n)-g(n)]^{f(n)-g(n)} \cdot \sqrt{f(n)-g(n)}}\end{aligned}$$

As $f(n)$ grows faster than $g(n)$,

$$\sqrt{\frac{f(n)}{g(n) \cdot (f(n)-g(n))}} \in \Theta\left(\frac{1}{\sqrt{g(n)}}\right)$$

and thus,

$$\binom{f(n)}{g(n)} \in \Theta\left(\frac{1}{e^2 \cdot \sqrt{2\pi}} \cdot \frac{f(n)^{f(n)}}{g(n)^{g(n)} \cdot \sqrt{g(n)} \cdot [f(n)-g(n)]^{f(n)-g(n)}}\right)$$

It is $f(n) - g(n) \leq f(n)$ and hence,

$$\binom{f(n)}{g(n)} \in \Omega\left(\frac{f(n)^{g(n)}}{\sqrt{g(n)} \cdot g(n)^{g(n)}}\right)$$

□

Lemma 6.16. *Let $u_1, u_2, k_1, k_2 \in \mathbb{N}$ with $u_1 > k_1$, $u_2 > k_2$ and $k_1 > k_2$. Furthermore, let $f_1 \colon \mathbb{N} \to \mathbb{N}$, $f_2 \colon \mathbb{N} \to \mathbb{N}$, $g_1 \colon \mathbb{N} \to \mathbb{N}$ and $g_2 \colon \mathbb{N} \to \mathbb{N}$ be functions with $f_1(n) \in \Theta(n^{u_1})$, $f_2(n) \in \Theta(n^{u_2})$, $g_1(n) \in \Theta(n^{k_1})$ and $g_2(n) \in \Theta(n^{k_2})$. Then,*

$$\binom{f_2(n)}{g_2(n)} \in o\left(\binom{f_1(n)}{g_1(n)}\right)$$

Proof. From Lemma 6.15, it follows that

$$\binom{f_1(n)}{g_1(n)} \in \Omega\left(\frac{f_1(n)^{g_1(n)}}{\sqrt{g_1(n)} \cdot g_1(n)^{g_1(n)}}\right)$$

Furthermore, for large n there exist constants $a_1, b_1, b_2 > 0$ such that

- $b_1 \cdot n^{k_1} \leq g_1(n) \leq b_2 \cdot n^{k_1}$
- $a_1 \cdot n^{u_1} \leq f_1(n)$

and hence, as $f_1(n)/g_1(n) \geq 1$ for large n,

$$\frac{f_1(n)^{g_1(n)}}{\sqrt{g_1(n)} \cdot g_1(n)^{g_1(n)}} = \frac{1}{\sqrt{g_1(n)}} \cdot \left(\frac{f_1(n)}{g_1(n)}\right)^{g_1(n)} \geq \frac{1}{\sqrt{g_1(n)}} \cdot \left(\frac{f_1(n)}{g_1(n)}\right)^{b_1 \cdot n^{k_1}}$$
$$\geq \frac{1}{\sqrt{b_2} \cdot n^{1/2k_1}} \cdot \frac{(a_1 \cdot n^{u_1})^{b_1 \cdot n^{k_1}}}{(b_2 \cdot n^{k_1})^{b_1 \cdot n^{k_1}}}$$

From this, it follows that

$$\binom{f_1(n)}{g_1(n)} \in \Omega\left(\frac{{a_1}^{b_1 \cdot n^{k_1}} \cdot n^{b_1 \cdot u_1 \cdot n^{k_1}}}{n^{1/2k_1} \cdot {b_2}^{b_1 \cdot n^{k_1}} \cdot n^{b_1 \cdot k_1 \cdot n^{k_1}}} =: l_1(n)\right)$$

On the other hand there exist constants $a_2, b_3 > 0$ such that for large r

- $f_2(n) \leq a_2 \cdot n^{u_2}$
- $g_2(n) \leq b_3 \cdot n^{k_2}$

and hence,

$$\binom{f_2(n)}{g_2(n)} \leq f_2(n)^{g_2(n)} \leq (a_2 \cdot n^{u_2})^{b_3 \cdot n^{k_2}} = {a_2}^{b_3 \cdot n^{k_2}} \cdot n^{b_3 \cdot u_2 \cdot n^{k_2}} =: l_2(n)$$

It remains to show that $l_2(n) \in o(l_1(n))$. To see that this is the case, we look at the logarithm:

$$\begin{aligned}\log(l_1(n)) &= b_1 \cdot n^{k_1} \cdot \log(a_1) + b_1 \cdot u_1 \cdot n^{k_1} \cdot \log(n) - \frac{1}{2} \cdot k_1 \cdot \log(n) \\ &\quad - b_1 \cdot n^{k_1} \cdot \log(b_2) - b_1 \cdot k_1 \cdot n^{k_1} \cdot \log(n) \\ &= b_1 \cdot \underbrace{(u_1 - k_1)}_{>0} \cdot n^{k_1} \cdot \log(n) + b_1 \cdot (\log(a_1) - \log(b_2)) \cdot n^{k_1} \\ &\quad - \frac{1}{2} \cdot k_1 \cdot \log(n)\end{aligned}$$

Hence, $\log(l_1(n)) \in \Theta(n^{k_1} \cdot \log(n))$. On the other hand,

$$\log(l_2(n)) = b_3 \cdot n^{k_2} \cdot \log(a_2) + b_3 \cdot u_2 \cdot n^{k_2} \cdot \log(n) \in \Theta(n^{k_2} \cdot \log(n))$$

Thus, $l_2(n) \in o(l_1(n))$.

□

We are now ready to prove Lemma 6.4.

Proof of Lemma 6.4. Assume that $\mathcal{C}$ is an optimal clustering with respect to surprise and $i_p(\mathcal{C}) - i_e(\mathcal{C}) > \frac{r^4}{2}$. We will compare $S(\mathcal{C})$ to the value of the clustering $\mathcal{C}'$ used in the proof of Lemma 6.3. $\mathcal{C}'$ is a clustering into cliques with $|\mathcal{M}| \cdot \binom{r^2}{2}$ intracluster edges.

Hence, $i_p(\mathcal{C}') = i_e(\mathcal{C}')$ and thus

$$\binom{p}{m} \cdot S(\mathcal{C}') = \binom{i_p(\mathcal{C}')}{i_e(\mathcal{C}')} \cdot \binom{p - i_e(\mathcal{C}')}{m - i_e(\mathcal{C}')} = \begin{pmatrix} \binom{|\mathcal{M}| \cdot r^2 + |\mathcal{X}|}{2} - |\mathcal{M}| \cdot \binom{r^2}{2} := f_2(r) \\ |\mathcal{X}| \cdot r^2 + \binom{|\mathcal{X}|}{2} := g_1(r) \end{pmatrix}$$

with $f_2 \in \Theta(r^6)$ and $g_2 \in \Theta(r^3)$.

Case 1: $i_e(\mathcal{C}) \leq i_p(\mathcal{C})/2$: As $i_e(\mathcal{C}) \leq i_p(\mathcal{C})/2$, substituting a lower bound for $i_e(\mathcal{C})$ decreases $\binom{i_p(\mathcal{C})}{i_e(\mathcal{C})}$. From Lemma 6.3, we know that $i_e(\mathcal{C}) \geq |\mathcal{M}| \cdot \binom{r^2}{2}$, which can be estimated from below by r^4 for large r. Altogether, we get that

$$\binom{p}{m} \cdot S(\mathcal{C}) = \sum_{i=i_e(\mathcal{C})}^{m} \binom{i_p(\mathcal{C})}{i} \cdot \binom{p - i_p(\mathcal{C})}{m - i} \geq \binom{i_p(\mathcal{C})}{i_e(\mathcal{C})} \geq \begin{pmatrix} 2 \cdot |\mathcal{M}| \cdot \binom{r^2}{2} := f_1(r) \\ r^4 := g_1(r) \end{pmatrix}$$

with $f_1 \in \Theta(r^5)$ and $g_1 \in \Theta(r^4)$. Now we can use Lemma 6.16 to see that, for large r, $S(\mathcal{C})$ is larger than $S(\mathcal{C}')$, which contradicts the optimality of $\mathcal{C}$.

Case 2: $i_e(\mathcal{C}) > i_p(\mathcal{C})/2$: We have $\binom{i_p(\mathcal{C})}{i_e(\mathcal{C})} = \binom{i_p(\mathcal{C})}{i_p(\mathcal{C}) - i_e(\mathcal{C})}$. As $i_e(\mathcal{C}) > i_p(\mathcal{C})/2$, $i_p(\mathcal{C}) - i_e(\mathcal{C}) < i_p(\mathcal{C})/2$ and substituting a lower bound for $i_p(\mathcal{C}) - i_e(\mathcal{C})$ decreases $\binom{i_p(\mathcal{C})}{i_p(\mathcal{C}) - i_e(\mathcal{C})}$. Hence, by Lemma 6.4,

$$\binom{p}{m} \cdot S(\mathcal{C}) \geq \binom{i_p(\mathcal{C})}{i_e(\mathcal{C})} = \binom{i_p(\mathcal{C})}{i_p(\mathcal{C}) - i_e(\mathcal{C})} \geq \binom{i_p(\mathcal{C})}{r^4/2} \geq \begin{pmatrix} |\mathcal{M}| \cdot \binom{r^2}{2} := f_1(r) \\ r^4/2 := g_1(r) \end{pmatrix}$$

with $f_1 \in \Theta(r^5)$ and $g_1 \in \Theta(r^4)$. Analogously to Case 1, it follows that $\mathcal{C}$ was not optimal. □

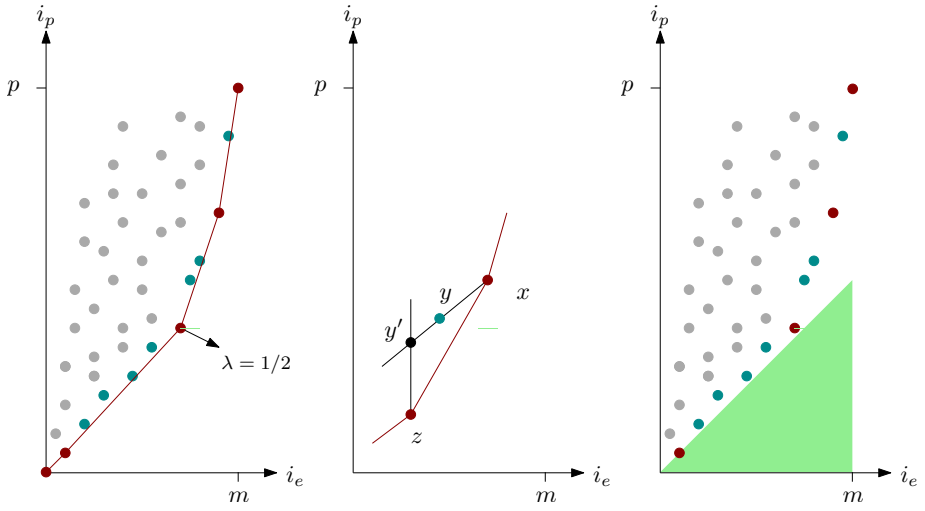
Chapter 7

Agglomerative Algorithms for Surprise Optimization

In Chapter 6, we discussed how optimal solutions with respect to surprise can be found by solving a sequence of integer linear programs. Although it is nice to have an exact method for small instances, typical applications require algorithms that are able to cluster graphs with millions of vertices and edges. In this context, it is often sufficient to compute clusterings that are not (provably) optimal, but still exhibit high quality, which is why we turn our attention to heuristics in this chapter. Heuristic algorithms for surprise maximization are still a largely untrodden field. The only algorithms we know of that explicitly target surprise are the algorithms UVCluster and SCluster [4, 15] and the metaheuristic SurpriseMe [9] that subsumes these algorithms. SurpriseMe runs a set of seven clustering algorithms, chosen by their performance with respect to surprise in the experiments by Aldecoa and Marín [8], and selects the clustering with the highest quality. Most of these algorithms are well known in the literature. In Chapter 6, we already saw that SurpriseMe yields an optimal clustering for the small instance `karate`, which indicates that it is indeed able to find clusterings of high surprise. A drawback of SurpriseMe is the large running time; clustering the small instance `football` already took approximately 4 seconds. Hence, there is still a need for improvement.

A straightforward method for explicit surprise maximization is to use a metaheuristic like Generic Greedy Vertex Moving that greedily optimizes surprise. A drawback of this approach is that the efficiency of Generic Greedy Vertex Moving fundamentally depends on the question whether we can determine the next cluster of each vertex v in $O(d_v)$. For surprise, it is unclear how to achieve this, even if the number of intracluster edges and pairs can be maintained efficiently throughout the algorithm. Another idea is to make use of the bicriteria view from Chapter 6 and optimize the number of intracluster pairs while constraining the number of intracluster edges, or vice versa. This is very similar to the algorithms discussed in Chapter 5, if we interpret $i_e = m - \mathsf{nxe}$ as a measure for intercluster sparsity and i_p as a measure for intracluster density. We deem however none of the two as a good priority function when building clusters on its own, as the former exclusively considers the number of covered edges and the latter only the balance of the clustering.

Contribution. We propose hence to use another standard approach to bicriteria optimization via optimizing a weighted sum of the two objectives. In Section 7.1, we give an



(a) Convex Hull (b) Impact of Concavity (c) Parameter Range

FIGURE 7.1: Illustration of weighted sum method. In (a), every red point can be found with the weighted sum method. Subfigure (b) illustrates that if surprise is strictly concave, optimum solutions always lie on the convex hull. Subfigure (c) indicates the relevant parameter range for λ: the green region does not contain any points.

explanation of why we deem this approach to be suitable in our context. We then describe and evaluate the individual algorithms contained in SurpriseMe on a benchmark set for graph clustering. It turns out that the high quality of the resulting clustering mostly stems from one particular algorithm, and that this algorithm is implicitly based on the weighted sum approach, although with a different motivation in mind. Motivated by this observation, we further engineer this approach, both with respect to quality and scalability.

7.1 Solving Surprise Maximization via the Weighted Sum Method

In Chapter 6, we exploited the fact that clusterings that are optimal with respect to surprise are Pareto optimal with respect to minimizing i_p and maximizing i_e, which means that all Pareto optimal solutions can be found by fixing one of the parameters to a certain value and optimizing the other under this constraint. Another simple standard approach to bicriteria problems is to consider the single objective problem that consists in optimizing a linear combination of the two objectives; in our case, this corresponds to maximizing the following objective for some parameter $\lambda \in \mathbb{Q}_0^+$.

$$\text{maximize } i_e(\mathcal{C}) - \lambda \cdot i_p(\mathcal{C}) \tag{7.1}$$

Let $\mathcal{Y}$ denote the *objective space* of our bicriteria problem, i.e., the set of $(x, y) \in \mathbb{Q}^2$ such that there is a clustering with x intercluster edges and y intracluster pairs. It is not too hard to see that all solutions of Equation 7.1 correspond to (weakly) Pareto optimal solutions. Furthermore, if $\mathcal{Y}$ is convex, all (weakly) Pareto optimal clusterings can be found by optimizing Equation 7.1 for some parameter $\lambda \in \mathbb{R}^+$ (cf. for example

Ehrgott [69], pages 68–71). A graphical illustration of the set $\mathcal{Y}$ and optimal solutions with respect to the weighted sum objective can be found in Figure 7.1 (a). The points in the figure depict all elements in $\mathcal{Y}$. Green and red points are nondominated, i.e., they correspond to Pareto optimal solutions; red points lie on the convex hull. The red point in the middle can be found by minimizing Equation 7.1 with $\lambda = 1/2$; this can be illustrated by the arrow indicating "in which direction" we seek for optimal solutions.

Unfortunately, for our problem, it is not hard to construct graphs such that $\mathcal{Y}$ is not convex. Hence, not all Pareto optimal clusterings can be found by optimizing the weighted sum. If, however, surprise, understood as a function in i_p and i_e, is strictly concave, it can be argued that optimal solutions with respect to surprise always correspond to points on the convex hull and the weighted sum approach works. There is no unique definition of what is meant by concave in the context of discrete functions [230]. For our purposes, it suffices if surprise can be extended to a strictly concave function on $\mathbb{R}^2$. This is illustrated in Figure 7.1 (b). Recall that for a strictly concave function f the value of any point on a line between x_1 and x_2 is strictly larger than the minimum of $f(x_1)$ and $f(x_2)$. Hence, if this property holds, the green point y is either worse than x or than y'. If x is worse than y', it is also worse than y by Lemma 6.1 (ii) and hence not optimal with respect to surprise.

Contour plots for the surprise landscape on an imaginary graph with 20 vertices and 10, 30, 50 or 100 edges can be found in Figure 7.2. The example demonstrates that surprise is not concave, which can be for example seen by considering the boundaries for levels 0.8 and 0.9. However, the plots suggest that it might be concave in the region below the red 0.5 line, which is actually the part we are interested in, given that the surprise of an optimal clustering of non degenerate graphs is usually far less than 0.5. Unfortunately, we do not know of any formal proof for this conjecture. Nonetheless, the discussion in this section gives an explanation why the weighted sum approach is expected to work well for surprise optimization. Indeed, it turns out that the best algorithm with respect to our test set among the algorithms contained in SurpriseMe uses exactly this approach, although it was designed with a completely different motivation in mind.

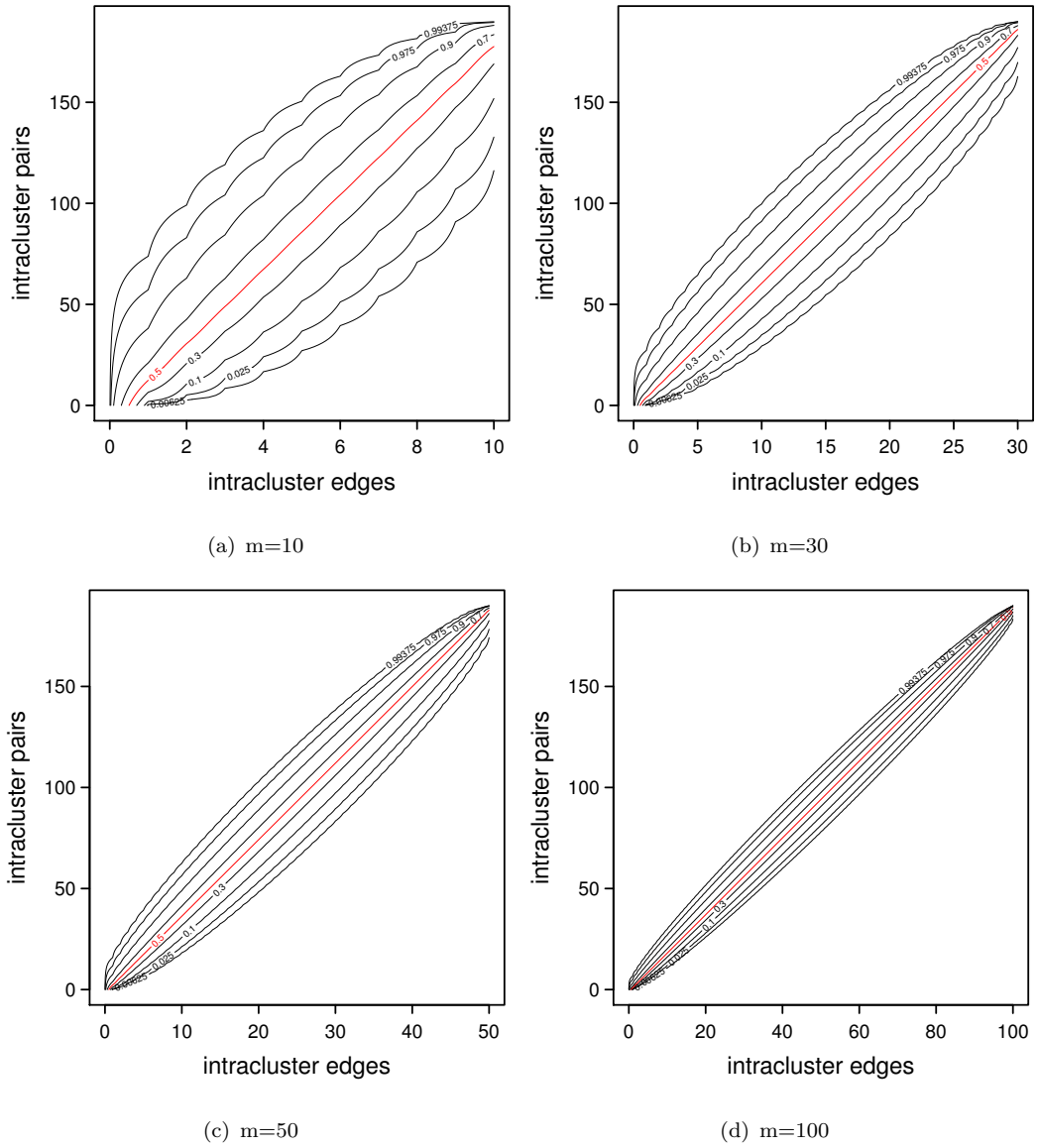
We close the discussion with an observation on the range of relevant parameter settings for λ. If $\lambda > 1$, as $i_e(\mathcal{C}) \leq i_p(\mathcal{C})$, the objective in Equation 7.1 is at most 0 and this value can only be obtained by the singleton clustering. This is graphically illustrated in Figure 7.1 (c); the green region does not contain any points. Hence, it is sufficient to consider values of λ in $[0, 1]$.

7.2 Algorithms contained in SurpriseMe

The metaheuristic SurpriseMe is based on the following seven graph clustering algorithms.

RNSC. The *Restricted Neighbourhood Search Clustering Algorithm* (RNSC) [136] is a local search algorithm that seeks to minimize the following objective function:

$$\frac{n-1}{3} \sum_{v \in V} \frac{|u \neq v \in V : \big(\{u,v\} \in E \text{ and } C(u) \neq C(v)\big) \text{ or } \big(\{u,v\} \notin E \text{ and } C(u) = C(v)\big)|}{1 + d_v + n_{C(v)}}$$



(a) m=10 (b) m=30

(c) m=50 (d) m=100

FIGURE 7.2: Contour plot of surprise as a function in i_e and i_p on an imaginary graph with 20 vertices and 10, 30, 50, or 100 edges. The set of levels used is $\{0.99375, 0.975, 0.9, 0.7, 0.5, 0.3, 0.1, 0.025, 0.00625\}$; the red line corresponds to Level 0.5.

To obtain this goal, the algorithm proceeds in two phases. In the first phase, a simplified objective function is used that only considers the numerator of the above equation, which is equivalent to the objective of cluster editing [196]. The resulting clustering then provides the initial clustering of the second phase, which optimizes the actual cost function. Both phases rely on a local search algorithm based on vertex moves, including some diversification steps to escape local minima. RNSC has been successfully applied to analyze protein-protein interaction networks [136].

UVCluster. UVCluster has first been proposed by Arnau et al. [15], again in the context of protein-protein interaction networks. SurpriseMe uses a more efficient implementation than the original one, which is part of the clustering suite Jerarca [4] provided

by Aldecoa and Marín. The motivation behind UVCluster is that conventional SAHN algorithms, especially in the first iterations, suffer from the *ties in proximity* problem if the distances are distributed only over a small range of values. This means that the decision of which clusters to merge in each step very much depends on the tie breaking strategy, and this affects the quality of the resulting clustering considerably. UVCluster solves this issue by computing secondary distances in a preprocessing step. To this end, a greedy algorithm for finding a clustering into disjoint cliques is applied, which is repeated several times. Now, the secondary distances are obtained by counting the number of times two vertices are not contained in the same cluster. After that, a dendrogram is created based on these secondary distances using a SAHN algorithm implementing average linkage [206], also called UPGMA. Finally, the level of the dendrogram is selected that corresponds to the clustering with the highest surprise.

SCluster. SCluster is also contained in Jerarca and almost identical to UVCluster. The difference lies in the computation of the set of clusterings that is produced to obtain the secondary distances. Instead of greedily computing disjoint cliques, an even faster approach is chosen which repeatedly selects a vertex and its neighbors, generates a cluster containing them and removes them from the graph.

Infomap. Infomap[1] is an algorithm that greedily optimizes the *map equation* [188]. The idea behind this objective function is that the information which cluster the vertices belong to helps in encoding a random walk on the graph. More precisely, if the random walk is interpreted as a discrete memoryless channel, the best expected encoding length we can hope for is given by the entropy of the source, using the probabilities given by the stationary distribution associated with the random walk. If it is however allowed to store the cluster of the last vertex in the walk, the walk can be encoded more efficiently by using customized codebooks to encode the vertices in each cluster. Informally, the map equation now associates with each clustering the expected encoding length per vertex of a random walk on a graph that uses a codebook for each of the clusters. The lower this value, the less likely are moves between clusters and hence, the better the clustering. Infomap optimizes this measure by using Generic Greedy Vertex Moving (cf. Algorithm 4 in Chapter 3), followed by some additional optimization steps.

RB. SurpriseMe also contains a variant of multiresolution modularity the authors attribute to Reichardt and Bornholdt [182] and therefore refer to it as the RB algorithm. In their article, Reichardt and Bornholdt show that modularity maximization can be seen as finding the ground state of an infinite range spin glass. The idea behind this point of view is that neighboring vertices attract each other in the sense that they aim for the same spin state and non-neighbors repulse each other and hence aim for different spin states. Given the concrete attractive and repulsive forces for each pair of vertices, the task is to find a spin configuration of minimum energy. If community ids are mapped to spin states, this defines a family of clustering measures. One particular instantiation of this problem is the modularity of a clustering as defined in Section 2.3. A more general model introduces a scaling factor γ into modularity that allows to specify a custom tradeoff between the first and the second term in the modularity equation:

$$\text{mod}_\gamma(\mathcal{C}) = \frac{1}{m} i_e(\mathcal{C}) - \gamma \sum_{C \in \mathcal{C}} \frac{{v_C}^2}{4m^2}$$

[1] http://www.mapequation.org/

If we replace v_C by n_C, minimizing this equation is equivalent to minimizing Equation 7.1, up to an additive and multiplicative constant. Again, the higher we choose the parameter γ the denser the clusters and hence, the more clusters we should expect. If $\gamma = 1$, we retrieve the definition of modularity. Aldecoa and Marín solve this problem heuristically for a range of parameter settings between 0.01 and 30. In the first iteration, the parameter is always increased by 1, in the second iteration by 1/2, in the third by 1/4 and so forth. If the last 100 solutions did not lead to a clustering that has higher surprise than any clustering found so far, the procedure terminates and returns the clustering with highest surprise among all solutions found. Finding a clustering that corresponds to a certain resolution scale is done by using the implementation of Traag et al.[2], which boils down to Generic Greedy Vertex Moving optimizing the above measure, without the optional refinement step.

RN. Ronhovde and Nussinov [187] propose another objective function that can be derived as a special case of the family of cluster measures defined by Reichardt and Bornholdt, called the *absolute Potts model*. For unweighted graphs, it can be written in our notation as

$$(1 + \gamma)i_e(\mathcal{C}) - \gamma i_p(\mathcal{C}),$$

up to a multiplicative and additive constant. Clearly, this is equivalent to the objective in Equation 7.1 by setting $\lambda = \gamma/(1 + \gamma)$. They motivate the use of this function by the observation that the tradeoff between the two terms does not depend on the size of the graph, which circumvents the resolution limit of modularity [84]. In their experiments, they use a move-based algorithm that is very similar, but not identical to Generic Greedy Vertex Moving to find good solutions for their model. Introducing a parameter that defines the resolution of the clustering entails the question how to choose one, or probably several, significant resolution scales. Ronhovde and Nussinov address this question in a follow-up article [186], where they propose to consider a sequence of parameter settings. For each parameter γ, they run their (non-deterministic) solver several times and compute the average similarity between the resulting clusterings using normalized mutual information [83]. If this is high, it is a sign that the local search landscape for this parameter setting contains only few significantly different local minima and hence, the cluster structure is more pronounced. Hence, they choose the scale with the highest similarity and return the best clustering found for this scale. SurpriseMe contains this metaheuristic under the name RN, using the implementation provided by the authors.

CPM. The last algorithm contained in SurpriseMe is based on the *constant Potts model* (CPM) proposed by Traag and Van Dooren [212]. Up to a multiplicative and additive constant, this objective function is the same as the one we defined in Equation 7.1 and therefore equivalent to the absolute Potts model up to a simple transformation of the parameter. Again, Traag and Van Dooren derive this function as a special case of the general model by Reichardt and Bornholdt and motivate its use by the resolution limit of modularity. In contrast to Ronhovde and Nussinov, they derive a formal definition of the notion *resolution-limit-free* and show that the CPM is resolution-limit-free according to this definition. In the experimental part, the model is evaluated by finding clusterings with Generic Greedy Vertex Moving, without the optional refinement. SurpriseMe tests a range of parameter values and chooses the one with the best surprise value. The choice of parameters used is similar to RB, with the difference that the parameter range is $[0.0001, 1]$ and the initial increment 0.5 instead of 1.

[2] https://launchpad.net/louvain

TABLE 7.1: Properties of graphs in test set

#	graph	n	m	#	graph	n	m
1	karate	34	78	19	as-22july06	22963	48436
2	chesapeake	39	170	20	cond-mat-2003_wi	30460	120029
3	dolphins	62	159	21	cond-mat-2005_wi	39577	175691
4	lesmis_uw	77	254	22	G_n_pin_pout_wi	99995	501198
5	polbooks	105	441	23	preferentialAttachment	100000	499985
6	adjnoun	112	425	24	smallworld	100000	499998
7	football	115	613	25	caidaRouterLevel	192244	609066
8	jazz	198	2742	26	coAuthorsCiteseer	227320	814134
9	celegansneural_uw	297	2148	27	citationCiteseer	268495	1156647
10	celegans_metabolic	453	2025	28	coAuthorsDBLP	299067	977676
11	email	1133	5451	29	cnr-2000	325557	2738969
12	polblogs_wi	1224	16715	30	coPapersCiteseer	434102	16036720
13	netscience_wi	1461	2742	31	coPapersDBLP	540486	15245729
14	power	4941	6594	32	eu-2005	862664	16138468
15	hep-th_wi	7610	15751	33	in-2004_wi	1382867	13591473
16	PGPgiantcompo	10680	24316	34	road_central	14081816	16933413
17	astro-ph_wi	16046	121251	35	uk-2002_wi	18483186	261787258
18	cond-mat_wi	16264	47594	36	road_usa	23947347	28854312

7.3 Performance of Algorithms in SurpriseMe

To get an impression how effectively and efficiently the algorithms contained in SurpriseMe optimize surprise on typical benchmark instances used for graph clustering, we ran the individual algorithms on a subset of the clustering category from the 10th DIMACS Implementation Challenge on Graph Partitioning and Graph Clustering[3]. As some of the algorithms took quite long to cluster the larger graphs, we only considered the graphs with up to 100000 vertices and set a time limit of 1 hour for each instance. The number of vertices and edges in these instances can be found in Table 7.1. These instances are a superset of the benchmark used in Chapter 5, but some of the graphs had to be modified slightly, which is why we list the properties of the benchmark graphs twice. The reason for that is that some of the implementations contained in SurpriseMe only support edge lists as input, which do not take into account isolated vertices. For this reason, we chose to remove them in a preprocessing step in order to guarantee that all algorithms work on the same graph. The graphs that are affected by this step can be identified by the appendix `wi` after the graph name. Furthermore, as surprise is only well defined on unweighted graphs, we consider all graphs as unweighted; this affects only the instances `lesmis` and `celegansneural`.

Parameter Setting and Measurement. Some of the algorithms contained in SurpriseMe depend on input parameters. To keep the running time for the experiments manageable, we used the parameter setting from SurpriseMe instead of tuning each algorithm separately. SurpriseMe applies default parameters whenever possible. UVCluster and SCluster only depend on one parameter, the number of iterations to obtain the secondary distances; for UVCluster, this is set to $\min\{10n, 1000\}$ and for SCluster to 10000. Infomap is a randomized algorithm that offers the possibility to supply a parameter r, which causes the base algorithm to be repeated r times; the final result then corresponds to the clustering with the best value according to the map equation. We

[3] http://www.cc.gatech.edu/dimacs10/

set r to 10, as in SurpriseMe. RB and CPM are parameter free. For RNSC and RN, default parameters are used, except for an upper bound of n on the number of scanned resolutions in combination with RN. To run the individual algorithms, we use the scripts in SurpriseMe. For the algorithms UVCluster, SCluster, RB and CPM, we measure the time for the whole script, as the former two are by the authors of SurpriseMe and for the latter two, the computation of surprise for each resolution scale is essential for the (meta)heuristic. For the other three, we only measure the time to cluster the graphs[4], without the computation of surprise and other pre or postprocessing steps. All experiments were executed on one core of an AMD Opteron Processor 6172. The machine is clocked at 2.1 GHz and has 256 GB of RAM in total.

Approximate computation of surprise. In Chapter 6, we used the GNU Multiple Precision library in order to compute the surprise of a given clustering with high precision. Applied to larger graphs, there are several issues with this approach. The first issue is that the numbers involved are very high due to the binomial coefficients; using arbitrary-precision arithmetic prevents overflows, but quickly results in unacceptably high running times and memory consumption. Aldecoa and Marín tackle this problem in the code provided for their metaheuristic SurpriseMe by computing the logarithm of surprise, $S'(\mathcal{C}) := -\log_{10} S(\mathcal{C})$, and use logarithms for intermediate results in the computation whenever possible. It turns out that this is sufficient to prevent overflows for the graph sizes we consider in this chapter. Nevertheless, computing the binomial coefficients is slow and actually dominates the running time for fast algorithms that rely on computing the objective more than once, as does our multilevel heuristic for surprise optimization. For this reason, Aldecoa and Marín use two approximations in their computation of surprise. The first one is to only take the first k summands of the surprise equation into account, where k is the first index such that the current summand is at least 4 orders of magnitude smaller than the sum of the preceding entries. As the contribution of the summands to the sum drops very quickly, this speeds up the running time considerably and we only observed a negligible impact on the result. Unfortunately, for large graphs, the running time is still not satisfactory. Hence, Aldecoa and Marín approximate factorials of numbers larger than 1000 by the Stirling approximation, i.e., they approximate $\log n!$ with $n \log n - n$. As this approach is fast enough for our purposes and yields good approximations, we adopt it in our algorithm and use the code provided in the source code for SurpriseMe.

Results. Table 7.2 shows the quality of the clusterings obtained by the particular algorithms in terms of S' and Table 7.3 the running time needed to compute them. Recall that $S'(\mathcal{C}) = -\log_{10} S(\mathcal{C})$ and we therefore aim at large values of S'. We set a time limit of 1 hour and a memory limit of 32 GB; if an algorithm exceeds these resources, this is indicated by the entry TO in the former and MO in the latter case in Table 7.3. In terms of quality, CPM and Infomap clearly dominate the field. There are only two instances where one of them does not give the best result: the small `karate` instance and a `preferentialAttachment` graph where both of them do not compete due to a timeout. Furthermore, for smaller graphs, CPM yields consistently better results than Infomap, which confirms the usefulness of the weighted sum method introduced in Section 7.1. CPM dominates RB, which might have been expected, given the fact that RB measures the size of a cluster in terms of accumulated vertex degrees and CPM in terms of the number of contained vertices; the latter is more likely to yield good results, in view of the bicriteria formulation of surprise. Furthermore, it is not surprising that RN yields worse

[4] In the case of RNSC, this includes the execution of the programs `rnscconvert`, `rnsc` and `rnscfilter`.

TABLE 7.2: Quality of algorithms in SurpriseMe with respect to S'

G	CPM	Infomap	RB	RN	RNSC	SCluster	UVCluster
1	23.85	17.17	21.19	**25.69**	25.04	23.66	25.69
2	**28.59**	0.00	16.67	25.30	28.02	23.54	20.07
3	**74.32**	51.66	56.75	44.57	74.28	72.20	66.26
4	**183.5**	136.5	119.5	30.40	182.1	181.7	161.1
5	**186.7**	151.4	157.9	154.1	166.7	166.1	162.2
6	**97.56**	37.63	72.22	22.83	74.31	82.84	80.66
7	**406.2**	403.8	405.6	403.8	405.7	379.8	365.1
8	**1026**	722.7	655.2	587.2	1003	971.4	965.9
9	**598.2**	442.5	431.5	75.64	542.4	500.9	459.2
10	**886.9**	652.1	674.5	67.16	730.8	752.2	721.8
11	**3366**	2960	2881	541.8	2505	3160	2855
12	**8995**	4195	4157	1630	5834	7691	3953
13	**6221**	6061	6068	4778	5936	6214	6186
14	12929	**13063**	12740	7482	9416	12869	12654
15	**31022**	30694	29795	25888	25362	-	30290
16	**49864**	47922	41738	42012	39114	-	-
17	**197631**	186190	162852	0.00	178364	-	-
18	**99056**	99002	93800	0.00	82884	-	-
19	**45998**	42459	43766	13896	16190	-	-
20	219938	**221672**	204116	0.00	179069	-	-
21	301109	**304198**	276365	0.00	240147	-	-
22	**353042**	-	-	-	257276	-	-
23	-	-	-	-	**231957**	-	-
24	-	**1201720**	-	-	976124	-	-

results than CPM, as both optimize Equation 7.1 for different values of λ and CPM chooses directly the clustering optimizing surprise, whereas RN is oblivious of surprise and chooses a clustering according to another criterion. Interestingly, RN deems 4 of the larger graphs as not well clusterable, i.e., it returns a trivial clustering. The other three algorithms yield fairly good results, but are, with few exceptions, consistently worse than CPM.

In terms of running time, CPM, RB and RN scale comparably well, managing to cluster graphs with up to approximately 35000 vertices. SCluster and UVCluster scale significantly worse with the graph size, while RNSC and Infomap are roughly comparable and faster than CPM.

In summary, with only few exceptions, RB, RN, SCluster and UVCluster are dominated by CPM and RNSC by Infomap with respect to both running time and quality. Hence, to save resources, we ran only these two algorithms on the whole benchmark set and compare our modifications to these two baseline algorithms only. As Infomap's base algorithm is repeated 10 times in SurpriseMe, we also included a version that only runs the base algorithm once, which yields an even faster heuristic we refer to as InfomapS. Table 7.8 and Table 7.9 show the quality and the running time of these algorithms on the whole benchmark set; the results for InfomapS are averaged over 5 runs (using random seeds between 1 and 5).

TABLE 7.3: Running time in seconds of algorithms in SurpriseMe

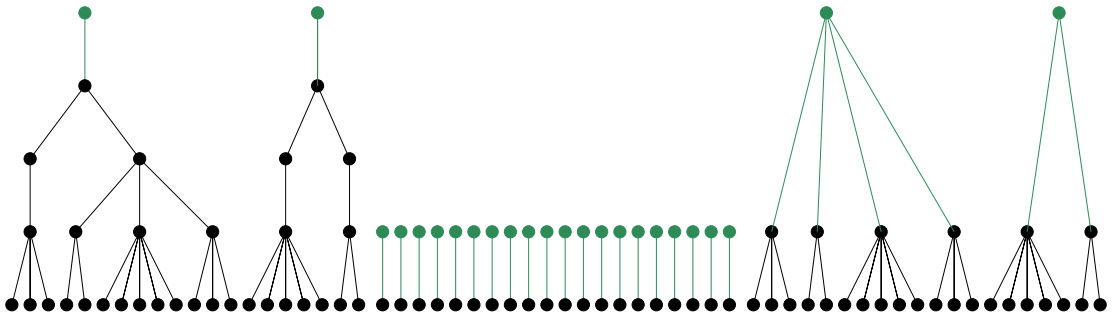
G	CPM	Infomap	RB	RN	RNSC	SCluster	UVCluster
1	5.41	**0.03**	5.51	0.08	0.04	0.08	0.04
2	5.75	**0.03**	3.52	0.13	0.07	0.11	0.04
3	7.68	**0.06**	34.81	0.29	**0.06**	0.27	0.09
4	4.7	**0.0**	5.0	0.46	0.1	0.2	0.1
5	8.4	**0.1**	7.8	0.8	**0.1**	0.3	0.2
6	14.50	0.11	5.49	0.82	**0.06**	0.30	0.19
7	5.8	0.2	7.4	0.8	**0.1**	0.3	**0.1**
8	14.7	0.2	18.8	4.2	**0.1**	0.8	0.3
9	17.8	0.4	30.8	5.57	**0.1**	1.4	0.7
10	17.9	0.5	31.8	8.12	**0.2**	3.0	0.9
11	43.0	2.8	66.4	30.0	**0.4**	15.8	7.6
12	60.6	**1.2**	263.0	53.6	2.8	17.4	6.2
13	16.3	0.5	29.6	11.1	**0.2**	39.4	23.8
14	76.2	7.0	148.0	79.9	**1.4**	1104	826.9
15	99.8	12.6	239.9	147.1	**4.5**	TO	2748
16	138.0	9.9	420.7	239.5	**9.3**	TO	TO
17	591.1	**49.9**	761.7	778.42	68.3	TO	TO
18	424.4	28.9	504.8	281.21	**28.6**	TO	TO
19	550.0	**31.8**	984.2	940.3	46.0	TO	TO
20	991.1	**100.2**	1187	1069	135.9	TO	TO
21	825.6	**170.7**	1797	1873	260.2	TO	TO
22	**2554**	TO	TO	TO	2837	MO	MO
23	TO	TO	TO	TO	**2455**	MO	MO
24	TO	**778.6**	TO	TO	2781	MO	MO

7.4 Engineering the Weighted Sum Method

Motivated by the good results of the weighted sum method included in SurpriseMe as CPM, we build upon this approach and further engineer it. Our main purpose is to decrease the running time, such that the algorithm can be used to cluster larger graphs within reasonable time bounds.

Base. Our first implementation is conceptually equivalent to CPM, with some small modifications. Recall that CPM finds clusterings optimizing Equation 7.1 with GENERIC GREEDY VERTEX MOVING as defined in Algorithm 4 in Chapter 3, without the optional refinement step. We keep this approach to compute clusterings for each parameter setting individually, which results in a fast way to estimate which parameter setting for λ leads to the best quality. In a postprocessing step, we run GENERIC GREEDY VERTEX MOVING once again for the estimated parameter setting, this time with refinement, to obtain the final result. Furthermore, instead of terminating as soon as the last 100 parameter settings for λ did not yield an improvement with respect to surprise, as is done in SurpriseMe, we use k parameter settings for λ, equidistant in $[0, 1]$. This avoids considering the same λ multiple times and leads to better predictable running times. We refer to the resulting metaheuristic as Base_k.

Dendrogram Heuristics (DH). Our student Philipp Glaser evaluated in his student thesis [92] different heuristics to speed up the running time to compute clusterings optimizing *multiresolution modularity*, which is equivalent to the objective of the RB heuristic. Given a set of parameters γ, he computes a clustering for each parameter setting by a modified GENERIC GREEDY VERTEX MOVING procedure which reuses parts



(a) After clustering (b) Initialization in Base_k (c) Initialization in $\text{DH}_k^{1/3,1}$

FIGURE 7.3: Illustration of how the algorithm is initialized with dendrogram heuristics.

of the dendrogram resulting from the previous parameter. Motivated by the promising results in this setting, we use this idea to compute clusterings optimizing Equation 7.1 for the parameters we are interested in.

Figure 7.3(a) depicts an example of a dendrogram that can be found by GENERIC GREEDY VERTEX MOVING for some parameter λ (the black vertices). The picture also contains an (artificial) additional level, which corresponds to the green vertices and emphasizes the clustering obtained on the highest level after local moving. Due to the way this algorithm proceeds, this clustering will always correspond to the singleton clustering, i.e., each green vertex is mapped to exactly one vertex on the highest level. With this convention, the initial state of the dendrogram in Base_k for each new parameter can be depicted as in Figure 7.3(b); the clustering process starts again from the lowest level and the clustering is reset to the singleton clustering.

Our DH heuristic reuses parts of the dendrogram instead, depending on two parameters d and $p \in [0, 1]$. Let us assume that the levels of the dendrogram are numbered from 0 to r, where Level 0 is the lowest level and Level r the highest level in the graph hierarchy, i.e., the black level directly below the green level. DH will now start the next computation from Level $\lfloor d \cdot r + 1/2 \rfloor$ and use the clustering defined by the vertices in Level $\lfloor p \cdot r + 1/2 \rfloor$. Figure 7.3(c) shows how the dendrogram is prepared in case $d = 1/3$ and $p = 1$. Here, we start the next computation from Level 1 and initialize the clustering with the two clusters corresponding to the highest level, prior to applying local moving to the graph at Level 1. Recall that the higher λ, the more clusters we expect. Hence, in order for this approach to make sense, we have to consider all λ in decreasing order. Similar to Base_k, we use a postprocessing step which runs GENERIC GREEDY VERTEX MOVING once again with the best parameter found, including the refinement phase. We call the resulting heuristic $\text{DH}_k^{d,p}$, where d and p are as defined above and k refers to the number of different parameter settings to be tested, analogous to Base_k.

Table 7.4 and Table 7.5 show the quality and running time of $\text{DH}_{100}^{d,p}$ for selected values of d and p with respect to our small benchmark set. Note that $\text{DH}_{100}^{0,0}$ is equivalent to Base_{100}. For multiresolution modularity, Glaser [92] finds that there is an inherent tradeoff between running time and quality; heuristics, which start from a high level in the dendrogram and use a coarse clustering as initialization take less time to complete but yield (slightly) worse results. With respect to Equation 7.1 and also with respect to the surprise of the clusterings found in the main phase of DH, we can confirm this observation. However, in terms of the quality of clustering after the postprocessing

TABLE 7.4: Quality of dendrogram heuristics

G	$\mathrm{DH}_{100}^{0,0}$	$\mathrm{DH}_{100}^{0,0.5}$	$\mathrm{DH}_{100}^{0,1}$	$\mathrm{DH}_{100}^{0.5,0.5}$	$\mathrm{DH}_{100}^{0.5,1}$	$\mathrm{DH}_{100}^{1,1}$
1	**25.55**	**25.55**	**25.55**	25.04	25.04	25.04
2	28.76	**28.94**	**28.94**	28.54	28.54	28.54
3	**75.72**	75.69	75.69	75.69	75.69	75.69
4	**183.8**	**183.8**	**183.8**	**183.8**	**183.8**	**183.8**
5	**189.7**	188.7	188.7	189.5	189.3	189.5
6	**102.3**	98.06	98.00	98.00	98.00	97.39
7	**406.2**	**406.2**	**406.2**	**406.2**	**406.2**	**406.2**
8	**1031**	1018	957.3	1021	1026	1016
9	**612.8**	592.7	585.3	590.7	592.1	592.1
10	**892.2**	885.6	887.1	**892.2**	892.0	892.0
11	3399	3373	3373	3399	3399	**3403**
12	**9043**	6729	6729	6792	6792	6729
13	**6233**	6231	**6233**	**6233**	**6233**	**6233**
14	**13029**	**13029**	**13029**	**13029**	**13029**	**13029**
15	**31041**	**31041**	**31041**	**31041**	**31041**	**31041**
16	**50197**	50066	**50197**	**50197**	**50197**	49867
17	**198076**	197981	197981	**198076**	**198076**	**198076**
18	**99769**	99717	99717	99393	99393	99717
19	**46482**	**46482**	**46482**	**46482**	**46482**	**46482**
20	**221584**	221036	221036	221036	221036	221036
21	**304292**	303281	303281	303281	300861	303281
22	304942	300063	**314481**	300063	304942	300063
23	271835	271835	271521	**271934**	271835	271835
24	**1190710**	**1190710**	**1190710**	**1190710**	1186650	**1190710**

step, the value does not drop significantly even for the fastest variant $\mathrm{DH}_{100}^{1,1}$, which still matches the quality of Base_{100} in 8 out of 24 cases and even surpasses the base algorithm on Instance 11. Hence, we ran this variant on the whole benchmark set and compare it to the other algorithms in the final discussion.

Parameter Selection by Fraction of Moved Vertices (F). Until now, we chose the set of parameters equidistant in the interval $[0, 1]$. Another possibility is to use the information from the dendrogram for the last parameter λ to determine the highest "interesting" parameter smaller than λ. The idea is that, to yield a substantially different clustering, optimizing Equation 7.1 with the new parameter should merge some of the clusters from the clustering obtained with parameter λ. Given a parameter $f \in [0, 100]$, we start with the singleton clustering and determine the largest λ such that at least f percent of the vertices would change their cluster membership in the local moving procedure, if we optimize Equation 7.1 with λ as parameter. This yields λ_1, which we use to obtain clustering $\mathcal{C}_1$. λ_2 is then defined by the largest λ such that at least f percent of the vertices on the highest level of the resulting dendrogram would change their cluster membership when optimizing Equation 7.1 with λ as parameter and so on. In principle, this way of parameter selection can be combined with any of the dendrogram heuristics, including the base algorithm. To keep the number of different configurations manageable, we only combine this approach with $DH^{1,1}$, which yields the fast heuristic F_f.

Intuitively, the smaller f, the more resolutions we scan and therefore, the better the expected quality of the resulting clustering. Table 7.6 however shows that this is not the case; the quality of F_0, which computes the largest value such that any of the vertices on the highest level is moved, is often worse than the quality obtained by using a constant

TABLE 7.5: Running time in seconds of dendrogram heuristics

G	$\mathrm{DH}_{100}^{0,0}$	$\mathrm{DH}_{100}^{0,0.5}$	$\mathrm{DH}_{100}^{0,1}$	$\mathrm{DH}_{100}^{0.5,0.5}$	$\mathrm{DH}_{100}^{0.5,1}$	$\mathrm{DH}_{100}^{1,1}$
1	0.11	**0.10**	**0.10**	**0.10**	**0.10**	**0.10**
2	**0.11**	0.13	**0.11**	**0.11**	**0.11**	0.13
3	3.23	0.12	**0.11**	**0.11**	**0.11**	0.13
4	0.1	0.1	0.1	0.1	**0.1**	0.1
5	0.2	0.1	0.1	**0.1**	**0.1**	**0.1**
6	0.3	0.60	0.15	**0.14**	0.16	0.16
7	0.3	0.2	0.1	0.1	**0.1**	7.5
8	1.7	0.3	0.1	**0.1**	**0.1**	0.3
9	0.3	0.3	0.3	**0.2**	**0.2**	**0.2**
10	0.3	0.3	0.3	**0.2**	**0.2**	0.3
11	0.9	0.5	0.5	0.3	0.3	**0.3**
12	1.8	1.1	1.0	0.7	0.5	**0.5**
13	0.3	0.3	0.3	0.2	**0.1**	**0.1**
14	1.3	0.8	0.7	0.5	0.5	**0.3**
15	2.8	1.5	1.4	0.9	0.8	**0.6**
16	4.4	2.9	2.7	1.5	1.3	**1.2**
17	20.6	12.0	11.2	4.5	4.3	**3.6**
18	9.2	5.1	4.9	2.3	2.1	**1.7**
19	11.9	8.9	7.3	5.7	5.5	**3.2**
20	27.1	16.9	16.6	7.1	6.6	**5.3**
21	49.6	27.6	26.8	12.3	11.2	**9.5**
22	243.6	147.8	150.4	98.9	95.8	**65.5**
23	177.2	123.6	118.9	86.7	78.4	**45.9**
24	155.7	70.4	122.0	44.2	41.1	**27.2**

fraction of the vertices. A possible explanation for that is the fact that this variant has the tendency to grow clusters one by one, similar to GENERIC GREEDY MERGE, which is often not beneficial for the resulting quality. Comparing the quality and running time (cf. Table 7.6) for different settings of f between 1.25 and 50 reveals no clear winner. In contrast to that, setting f to 100, hence requiring all clusters to merge in the next step, is too restrictive and the quality of the resulting clustering drops considerably in most cases. Overall, we think that F_{10} yields a good tradeoff between running time and quality.

7.5 Scanning the Convex Hull Directly

An alternative approach to find interesting values of the parameter λ is to reconstruct the convex hull of the set of nondominated points recursively. Let $p := \binom{n}{2}$. Starting with the two known border points $x^l = \binom{0}{0}$ and $x^r = \binom{m}{p}$ corresponding to the singleton and the all clustering, new points can be found as described in Algorithm 8. The first relevant value of λ is chosen such that we optimize in a direction that is orthogonal to the line between x^l and x^r. If the optimal clustering corresponds to a point x^λ to the lower right of this line, this point is guaranteed to be a nondominated point on the convex hull. In this case, we repeat this process with the line between x^λ and x^l and the line between x^λ and x^r. Otherwise, we know that we found all vertices of the convex hull that lie between x^l and x^r. Occasionally, it may happen that the optimal clustering for a particular λ is not unique. If we would like to find all points on the convex hull, we would have to enumerate all optimal solutions or at least to enumerate all points $\binom{x}{y}$

TABLE 7.6: Results for F_f heuristic, different values of f

(a) Quality with respect to S'

G	F_0	$F_{1.25}$	$F_{2.5}$	F_{10}	F_{50}	F_{100}
1	**25.04**	**25.04**	**25.04**	**25.04**	24.04	21.08
2	28.54	28.54	28.54	28.54	**28.71**	20.10
3	**75.69**	**75.69**	**75.69**	75.55	75.42	52.37
4	**183.8**	**183.8**	**183.8**	**183.8**	137.7	151.7
5	**189.3**	**189.3**	**189.3**	189.2	189.2	154.1
6	97.39	**98.09**	**98.09**	**98.09**	**98.09**	71.68
7	**406.2**	**406.2**	**406.2**	403.8	347.1	347.1
8	1016	1016	1016	**1029**	899.1	899.1
9	592.1	**610.9**	**610.9**	**610.9**	**610.9**	379.1
10	**890.6**	**890.6**	**890.6**	886.2	802.6	825.6
11	3337	3399	**3399**	3373	3249	2286
12	6767	6790	6782	**6795**	6776	5696
13	6230	6230	**6234**	**6234**	**6234**	5860
14	**13029**	13029	13026	13013	13026	12905
15	31055	31055	31032	**31096**	31096	29644
16	50181	**50193**	**50193**	**50193**	49021	43433
17	**198076**	198071	198071	197632	195422	161954
18	99723	**99732**	99723	99364	99365	88923
19	46069	46132	46132	**46258**	44897	45366
20	221259	**221324**	221096	220895	215468	171268
21	303167	303634	303095	**304035**	289160	218295
22	-	297451	299199	301420	293325	**317579**
23	-	**271929**	**271929**	271500	271908	269552
24	1189750	**1190530**	1190500	1173000	1132690	1185080

(b) Running Time in Seconds

G	F_0	$F_{1.25}$	$F_{2.5}$	F_{10}	F_{50}	F_{100}
1	0.09	**0.08**	**0.08**	0.09	0.09	0.09
2	**0.09**	**0.09**	**0.09**	0.11	0.10	**0.09**
3	**0.09**	0.10	**0.09**	0.11	**0.09**	**0.09**
4	0.1	0.1	0.1	0.1	**0.1**	**0.1**
5	0.1	0.1	0.1	0.1	**0.1**	**0.1**
6	0.12	0.12	0.12	0.11	0.11	**0.10**
7	0.1	0.1	0.1	2.2	0.1	**0.1**
8	0.1	**0.1**	**0.1**	2.0	**0.1**	**0.1**
9	0.1	0.1	**0.1**	0.1	**0.1**	**0.1**
10	0.2	0.1	0.1	0.1	0.1	**0.1**
11	0.4	0.2	0.1	0.1	**0.1**	**0.1**
12	0.7	**0.3**	**0.3**	**0.3**	0.3	0.3
13	0.1	0.1	0.1	0.1	**0.1**	**0.1**
14	0.6	0.3	0.3	0.3	0.2	**0.2**
15	1.5	0.5	0.4	0.3	**0.3**	0.3
16	3.5	1.1	0.9	0.5	**0.4**	**0.4**
17	16.7	1.7	1.5	1.5	1.3	**1.1**
18	6.3	1.1	0.9	0.7	0.7	**0.6**
19	37.4	19.9	16.7	13.7	2.0	**0.8**
20	30.6	2.5	2.3	1.9	1.5	**1.5**
21	63.7	4.1	3.4	2.7	2.5	**2.2**
22	MO	47.4	38.0	38.4	**20.8**	43.3
23	MO	35.1	30.0	24.6	18.9	**18.8**
24	373.1	14.2	12.3	16.3	**8.5**	9.2

such that a clustering with x intracluster edges and y intracluster pairs exists and such that $x - \lambda y$ corresponds to the value of the optimum. We did not choose this approach and restrict ourselves to finding all vertices of the convex hull. It is not hard to see that whenever we find a point on the convex hull that is no vertex, i.e., that lies on a line between two vertices l and r, these vertices will be found later on. This means, each point found by Algorithm 8 is a nondominated point on the convex hull, but not each nondominated point on the convex hull is necessarily found; this is only guaranteed for vertices. In contrast to the parameter scanning done in Base$_k$, this approach has the nice property that it scans at most $2c$ resolutions, where c is the number of points on the convex hull. A trivial upper bound on c is the number of edges in the graphs; in general, the algorithm will scan less resolutions.

OptConvex. Optimal solutions for Equation 7.1 can be easily found by an integer linear program similar to the ones we used in Chapter 6. We build upon the same set of variables $\mathcal{X}$ as in the case of minIP. Integrality (cf. Equation 6.3) and transitivity (cf. Equation 6.4) constraints guarantee that the solution of the linear program induces a clustering of the graph. With the same arguments we used in the case of minIP, it is sufficient to restrict the transitivity constraints for vertices u and v to vertices in a vertex separator of them. The objective in Equation 7.1 can now be easily translated as follows:

$$\text{maximize} \sum_{\{u,v\}\in E} \mathcal{X}_{uv} - \lambda \cdot \sum_{\{u,v\}\in\binom{V}{2}} \mathcal{X}_{uv} \tag{7.2}$$

As motivated in Section 7.1, we conjecture that the point corresponding to a clustering that is optimal with respect to surprise is a vertex on the convex hull. If this conjecture

Algorithm 8: OptConvex

Input : Graph G with $p = \binom{n}{2}$ vertex pairs and m edges
Output: Set S containing all nondominated points that are vertices of the convex hull

$S \leftarrow \{\binom{0}{0}, \binom{m}{p}\}$
$L \leftarrow ([\binom{0}{0}, \binom{m}{p}, m/p])$
while L *is not empty* **do**
  $[x^l, x^r, \lambda] \leftarrow L$.dequeue()
  $\mathcal{C}^\lambda \leftarrow \min_{\mathcal{C}}\{i_e(\mathcal{C}) - \lambda i_p(\mathcal{C})\}$
  $x^\lambda \leftarrow \binom{i_e(\mathcal{C}^\lambda)}{i_p(\mathcal{C}^\lambda)}$
  if x^λ *is not on line between* x^l *and* x^r **then**
    $S \leftarrow S \cup \{x^\lambda\}$
    if $x_1^\lambda - x_1^l > 1$ **then**
      L.enqueue($[x^l, x^\lambda, (x_1^\lambda - x_1^l)/(x_2^\lambda - x_2^l)]$)
    if $x_1^r - x_1^\lambda > 1$ **then**
      L.enqueue($[x^\lambda, x^r, (x_1^r - x_1^\lambda)/(x_2^r - x_2^\lambda)]$)
return S

is true, Algorithm 8, which we call OptConvex, yields an alternative way to obtain provably optimal clusterings with respect to surprise.

Bettinelli et al. consider the related problem to compute the piece-wise linear function P that assigns each λ the value of an optimal solution with respect to multiresolution modularity [29] and the weighted parsimonity criterion [30], with λ as parameter. Again, the latter is equivalent to CPM with respect to optimum solutions, up to a simple transformation of the parameter. It is not hard to see that computing the breakpoints of P and computing all vertices on the convex hull of the Pareto front are two sides of the same coin, as there is a one-to-one correspondence between the coordinates of these vertices and the equations associated with the pieces of P. Hence, OptConvex can be used to compute P, and the algorithm by Bettinelli et al. can be used to find all vertices on the convex hull of the Pareto front. Both approaches rely on computing optimum solutions for Equation 7.1 with the help of a linear program for a sequence of values for λ. In contrast to the recursive approach from Algorithm 8, Bettinelli et al. reconstruct P "from left to right", i.e., from the lowest possible value for λ to the largest one. Compared to OptConvex, in the best case, this might lead to slightly less calls to the ILP solver, but there is no good upper bound on the maximum number of calls needed. Hence, there is a priori no reason to choose this more complicated algorithm in our scenario instead of the conceptually simple recursive approach.

We ran OptConvex and the gap variant (Opt) from Chapter 6, including the modification that prunes small k, on the seven smallest instances from our test set, with a timeout of 5 hours. Although four of these instances have been already considered in Chapter 6, we had to rerun the experiments for the gap heuristic as we use another machine and timeout as in the experiments there. Table 7.7 shows the running time, number of solved linear programs and the quality for OptConvex and Opt for these instances. As expected, whenever both approaches found a solution within the time limit, their quality is identical. Except for Instance 4, where Opt needed to call the ILP solver less than half the times, OptConvex was always faster, for Instance 2 and 7, considerably.

TABLE 7.7: Experimental results for OptConvex

	OptConvex			Opt		
G	time	ILP	S'	time	ILP	S'
1	13s	32	25.69	16s	20	25.69
2	1h 44m 9s	40	29.24	TO	-	-
3	47s	53	76.00	2m 22s	50	76.00
4	4m 34s	67	183.81	3m 1s	29	183.81
5	TO	-	-	TO	-	-
6	TO	-	-	TO	-	-
7	11m 38s	41	406.25	4h 15m 47s	53	406.25

Convex$_k$. Obviously, the above approach can only be used for small instances. We hence considered a heuristic that works in principle as in Algorithm 8 but replaces the step to find optimal solutions for a particular parameter λ by finding a good clustering with GENERIC GREEDY VERTEX MOVING. Furthermore, we terminate as soon as we have scanned k resolutions and call GENERIC GREEDY VERTEX MOVING again on the best parameter found, this time using refinement. This is very similar to the base algorithm, the only difference is that we determine the set of resolutions to scan dynamically while executing the algorithm. Unfortunately, this hinders the combination with the heuristic DH, as this requires the set of λ values to be ordered decreasingly.

7.6 Experimental Results

Table 7.8 and Table 7.9 show the quality and running time of selected configurations on the whole set of graphs in Table 7.1.

Infomap vs. InfomapS. As to be expected, running Infomap's base algorithm once instead of 10 times reduces the running time by approximately a factor of 10. On 18 out of 25 instances where both variants found a solution within the time limit, the quality drops slightly without the repetition, but, with the exception of Instance 6, not considerably. Overall, InfomapS seems to be a good and fast heuristic for graphs with 100000 vertices and more, being able to cluster a webgraph with over 13 million edges within approximately 42 minutes.

Base vs. CPM and Infomap. It turns out that the quality loss incurred by only scanning a fixed number of 100 resolutions is more than compensated by the simple postprocessing which runs GENERIC GREEDY VERTEX MOVING once again on the best parameter found, including the refinement phase. In 21 out of 22 instances, Base$_{100}$ yields a better quality than CPM while the computation time drops by a factor of around ten to twenty, making the potential of the weighted sum approach more apparent. Compared to Infomap, Base$_{100}$ is significantly better on the small instances 1 to 12 with respect to quality, and still better than Infomap (InfomapS) on 9 out of 13 (14 out of 19) of the remaining instances. For larger instances, its running time is usually in between the time needed for Infomap and InfomapS, often comparable to InfomapS.

Convex vs. Base. Comparing the quality of Base$_{100}$ and Convex$_{100}$ reveals no clear advantage in using either of the both variants over the other, although there might be a slight tendency towards the former on small and towards the latter on larger instances.

TABLE 7.8: Quality of algorithms on large set with respect to S'

G	CPM	Infomap	InfomapS	Base_{100}	Convex_{100}	Base_{10}	Convex_{10}	$\text{DH}^{1,1}_{100}$	F_{10}
1	23.85	17.17	17.17	**25.55**	**25.55**	**25.55**	**25.55**	25.04	25.04
2	28.59	0.00	0.00	**28.76**	**28.76**	28.71	28.71	28.54	28.54
3	74.32	51.66	53.03	**75.72**	75.69	75.69	75.55	75.69	75.55
4	183.5	136.5	132.4	**183.8**	**183.8**	**183.8**	**183.8**	**183.8**	**183.8**
5	186.7	151.4	151.1	**189.7**	189.5	188.7	189.5	189.5	189.2
6	97.56	37.63	14.42	102.3	102.3	98.08	**103.2**	97.39	98.09
7	**406.2**	403.8	403.8	**406.2**	**406.2**	**406.2**	**406.2**	**406.2**	403.8
8	1026	722.7	699.1	**1031**	1025	1017	1025	1016	1029
9	598.2	442.5	440.9	**612.8**	**612.8**	593.5	**612.8**	592.1	610.9
10	886.9	652.1	646.3	892.2	**894.0**	880.4	884.2	892.0	886.2
11	3366	2960	2927	3399	3378	3323	3293	**3403**	3373
12	8995	4195	4182	**9043**	8981	8895	8981	6729	6795
13	6221	6061	6057	6233	6233	6219	6207	6233	**6234**
14	12929	**13063**	13036	13029	13021	12443	12970	13029	13013
15	31022	30694	30672	31041	31063	30601	30997	31041	**31096**
16	49864	47922	47916	**50197**	50134	49193	49051	49867	50193
17	197631	186190	186391	198076	197867	198187	**198389**	198076	197632
18	99056	99002	98980	99769	**99778**	98406	98763	99717	99364
19	45998	42459	42422	46482	**46705**	36756	44845	46482	46258
20	219938	221672	221772	221584	**222020**	218101	221299	221036	220895
21	301109	304198	303769	**304292**	303558	297904	300060	303281	304035
22	353042	-	**480393**	304942	304904	276966	297588	300063	301420
23	-	-	212059	**271835**	271524	264663	271524	**271835**	271500
24	-	**1.202e+6**	1.201e+6	1.191e+6	1.191e+6	1.171e+6	1.191e+6	1.191e+6	1.173e+6
25	-	1.354e+6	1.353e+6	1.358e+6	**1.361e+6**	1.175e+6	1.335e+6	1.358e+6	1.337e+6
26	-	2.378e+6	2.378e+6	**2.412e+6**	2.407e+6	2.347e+6	2.407e+6	**2.412e+6**	2.411e+6
27	-	-	1.995e+6	2.004e+6	**2.011e+6**	1.636e+6	2.011e+6	2.004e+6	2.010e+6
28	-	-	2.566e+6	2.572e+6	**2.573e+6**	2.449e+6	2.563e+6	2.571e+6	2.573e+6
29	-	4.601e+6	4.601e+6	4.314e+6	5.225e+6	3.519e+6	5.183e+6	4.314e+6	**5.225e+6**
30	-	-	4.604e+7	4.774e+7	**4.775e+7**	4.763e+7	4.774e+7	4.763e+7	4.759e+7
31	-	-	4.103e+7	**4.281e+7**	4.276e+7	4.278e+7	4.208e+7	4.269e+7	4.276e+7
32	-	-	2.470e+7	-	-	2.263e+7	**2.929e+7**	-	2.857e+7
33	-	-	3.522e+7	-	-	3.160e+7	**3.761e+7**	3.690e+7	-
34	-	-	-	-	-	7.588e+7	8.678e+7	8.692e+7	**8.719e+7**
35	-	-	-	-	-	-	-	-	-
36	-	-	-	-	-	1.343e+8	-	-	**1.549e+8**

Convex_{100} is often slightly slower, as it apparently tends to scan more resolutions that are more costly to optimize. If we however compare Base_{10} and Convex_{10}, the latter yields better quality on 26 and the same result on 4 out of 34 instances, sometimes considerably better quality. Our interpretation is that 100 parameter settings are more than enough for both variants, especially for the smaller instances, but the parameter estimation of Convex_k is preferable if k is small or if we are dealing with especially large instances. Compared to InfomapS, Convex_{10} is faster on large graphs, sometimes up to a factor of 8, while yielding better quality in 25 out of 33 cases.

DH and F. On our test set, $\text{DH}^{1,1}_{100}$ and F_{10} yield comparable quality; either of the two wins in about half of the cases and the results are often similar. As F_{10} is consistently faster than $\text{DH}^{1,1}_{100}$, we deem it the better choice among the two versions. Similarly, compared to Convex_{10}, it yields better results in 17 out of 30 cases and is usually faster. There is however a slight catch with both the DH and F heuristic; due to the way the dendrogram is built, it usually consists of considerably more levels in the end than by using the base algorithm for any of the λ considered. This is the reason why $\text{DH}^{1,1}_{100}$ needs

TABLE 7.9: Running time in seconds of algorithms on large set

G	CPM	Infomap	InfomapS	Base_{100}	Convex_{100}	Base_{10}	Convex_{10}	$\text{DH}^{1,1}_{100}$	F_{10}
1	5.41	0.03	**0.02**	0.11	0.09	0.09	0.09	0.10	0.09
2	5.75	0.03	**0.02**	0.11	0.12	0.09	0.09	0.13	0.11
3	7.68	0.06	**0.02**	3.23	2.58	0.09	0.11	0.13	0.11
4	4.7	**0.0**	0.1	0.1	0.1	0.1	0.1	0.1	0.1
5	8.4	**0.1**	0.2	0.2	0.2	0.1	0.1	0.1	0.1
6	14.50	0.11	**0.04**	0.3	0.3	0.75	2.4	0.16	0.11
7	5.8	0.2	**0.0**	0.3	0.3	0.1	0.7	7.5	2.2
8	14.7	0.2	**0.0**	1.7	0.3	0.1	1.6	0.3	2.0
9	17.8	0.4	**0.1**	0.3	0.3	0.1	0.1	0.2	0.1
10	17.9	0.5	**0.1**	0.3	0.3	0.1	0.1	0.3	0.1
11	43.0	2.8	0.3	0.9	0.6	0.3	0.3	0.3	**0.1**
12	60.6	1.2	**0.2**	1.8	1.1	0.3	1.4	0.5	0.3
13	16.3	0.5	1.4	0.3	0.3	**0.1**	0.3	0.1	**0.1**
14	76.2	7.0	2.3	1.3	1.6	**0.3**	0.5	0.3	**0.3**
15	99.8	12.6	2.9	2.8	2.5	0.5	0.5	0.6	**0.3**
16	138.0	9.9	1.8	4.4	3.6	0.7	0.7	1.2	**0.5**
17	591.1	49.9	5.3	20.6	17.9	2.8	2.9	3.6	**1.5**
18	424.4	28.9	3.2	9.2	7.6	1.3	1.2	1.7	**0.7**
19	550.0	31.8	3.3	11.9	22.8	**1.5**	3.4	3.2	13.7
20	991.1	100.2	11.3	27.1	27.2	3.8	3.8	5.3	**1.9**
21	825.6	170.7	18.5	49.6	45.4	6.3	6.0	9.5	**2.7**
22	2554	TO	548.8	243.6	596.2	**32.9**	104.6	65.5	38.4
23	TO	TO	703.2	177.2	1966	**24.5**	234.1	45.9	24.6
24	TO	778.6	79.8	155.7	213.1	56.2	24.4	27.2	**16.3**
25	TO	1317	140.6	340.2	715.9	45.5	40.1	56.7	**22.1**
26	TO	1482	135.5	213.4	200.9	27.8	31.5	48.8	**14.8**
27	TO	TO	895.2	689.8	964.2	115.0	113.3	172.2	**40.6**
28	TO	TO	424.7	345.8	478.8	46.6	54.1	84.3	**26.2**
29	TO	2883	256.9	1078	2369	**122.4**	302.2	159.7	143.4
30	TO	TO	477.6	1339	1280	204.3	192.6	120.2	**99.7**
31	TO	TO	829.0	1911	2330	284.9	229.5	211.1	**110.9**
32	TO	TO	1916	TO	TO	**865.5**	1913	MO	1115
33	TO	TO	2525	TO	TO	**653.5**	1226	735.8	MO
34	MO	TO	TO	TO	TO	1369	2973	1644	**876.3**
35	MO	TO	TO	MO	MO	MO	MO	MO	MO
36	MO	TO	TO	TO	TO	2210	TO	MO	**1199**

slightly more than 32 GB of main memory for Instance 32, which also happens for F_{10} on Instance 33. For the fast versions $\text{DH}^{1,1}_{100}$ and F_{10}, this could however easily be avoided by storing only the highest level of the dendrogram, as we do not reuse intermediate levels in the clustering process.

Summary. Of all configurations we tested, we deem Base_{100} and F_{10} to yield the best tradeoff between running time and quality. The former yields better results than any of the algorithms contained in SurpriseMe on a large majority of the test instances we considered, while its running time is still often comparable to a single application of Infomap. The latter is faster than InfomapS and yields better quality on all but 5 instances. For `karate` (Instance 1), `dolphins` (Instance 3), `lesmis` (Instance 4) and `football` (Instance 7), we know the surprise of an optimal solution (cf. Table 7.7). Furthermore, for `chesapeake` (Instance 2), by using OptConvex, we found a lower bound on the value of an optimal solution that we believe to be optimal. The clustering found by Base_{100} matches this quality in case of `lesmis` and `football`, while the results for the

other instances are slightly suboptimal. Hence, at least for small instances, the margin for significant improvements with respect to quality is small.

7.7 Concluding Remarks

In this chapter, we evaluated heuristic algorithms for surprise optimization. We first examined the clustering algorithms contained in the metaheuristic SurpriseMe and found that for typical test instances, one of these algorithms dominates the others with respect to the quality of the resulting clustering. We gave a theoretical justification of this approach by connecting it to the bicriteria view on surprise via the weighted sum method that optimizes a linear combination of the number of intracluster edges and intracluster pairs. The algorithm as contained in SurpriseMe is a metaheuristic on its own, which solves the above weighted sum objective for a range of weights or resolution scales. We engineered this approach by introducing a simple postprocessing step which improves the quality of the clustering and by reusing parts of the dendrogram between the clustering procedures for the particular parameters instead of always computing the clustering from scratch. Furthermore, we examined two orthogonal ways to determine interesting resolution scales automatically during the clustering process. The former is based on an estimation of how much the clustering will change when decreasing the weight, while the latter directly scans nondominated points on the convex hull of the objective space of our bicriteria problem, either exactly or heuristically. Based on the experimental results, we propose two simple configurations of the weighted sum approach; the former yields most of the time better quality than SurpriseMe while consuming far less resources, while the latter is faster than the fastest of the algorithms in SurpriseMe and still yields competitive quality. The main purpose of this and the preceding chapter is to demonstrate that even if the expression associated with surprise might seem daunting at first glance, it can be optimized reasonably well in practice.

The methods used in this chapter are not limited to the scope of surprise optimization, but can be used whenever the aim is to estimate properties of the clustering found by optimizing CPM [212] or, similarly, multiresolution modularity [182] depending on the resolution scale. For multiresolution modularity, some experiments can be found in the student thesis of Glaser [92]. Scanning the convex hull directly yields also a fast alternative to the algorithm by Bettinelli et al. [29, 30], and to recursive bisection on the resolution parameter as proposed for example by Traag et al. [213]. This is particularly useful if the task is to find clusterings which have a certain property that grows monotonously with the resolution parameter, as for example the number of clusters, as in this case a logarithmic number of computations is sufficient to find the correct resolution scale.

It would be interesting to know if our conjecture is true that clusterings that are optimal with respect to surprise always correspond to vertices on the convex hull in objective space. If yes, this yields both an even stronger connection to the constant Potts model CPM and a faster method for finding optimal solutions. To substantially widen the range of graph sizes where computing optimum clusterings is realistic, it is however necessary to use other improvements; potential candidates for this are mentioned in Chapter 6. Another direction of future work includes the development of better heuristics with respect to quality and running time. A simple modification of the methods we used is to perform the local moving procedure in parallel. As for modularity [209],

this can be done in a straightforward way by ignoring stale data, although it might lead to slightly worse quality. Another possibility would be to go back to the idea of directly optimizing surprise by Generic Greedy Vertex Moving. Evaluating the exact change in the objective function when moving vertices to other clusters seems infeasible on large instances, but using approximations similar to the ones we used to approximate surprise for large graphs might be possible. It is not obvious in advance if this leads to better quality or efficiency; even with approximations, evaluating the benefit of single move operations will be more costly than for CPM, and it is not clear if these approximations are good enough to be used on such a fine scale.

On a more conceptual level, it is interesting that Infomap seems to yield clusters of very high quality on large graphs, although it does not explicitly optimize surprise. Instead, it is based upon the map equation. One possible explanation for this might be that optimizing the map equation becomes more or less equivalent to optimizing surprise if the graph size goes to infinity. Investigating similarities between the two objectives is interesting, as such interconnections would strengthen the theoretical foundation of both of them.

Chapter 8

An Efficient Dynamic Generator for Clustered Random Graphs

Often, researchers developing new or enhanced algorithms are faced with the question which data sets to use to illustrate the advantages and validity of their approach. In the context of clustering dynamic graphs, the data needs to have some temporal aspects and, ideally, should come with a well-motivated ground truth clustering. Additionally, using data that has been previously used in the literature makes the comparison to other methods less cumbersome.

Synthetic data refers to graphs that are artificially generated by graph generators. Given a number of vertices, such generators decide which vertices are connected by an edge, based on some probabilistic model. The edge probabilities can be derived for example from a preferential attachment process [20], where vertices that already have a high degree are connected with higher probability than others, or from other rules that are characteristic for the particular generator. In the context of dynamic graphs, graph generators usually not only have to decide which vertices are linked in the beginning, but also which vertices or edges are added or deleted over time. Furthermore, if the generator incorporates a hidden ground truth clustering, this usually evolves randomly as well, which in turn influences the edge probabilities.

Depending on the aim of designing a certain clustering algorithm, there are good reasons to use synthetic data as well as good reasons to not entirely rely on synthetic data for its evaluation. One advantage of real world data, i.e., instances that stem from typical applications, is that they frequently exhibit very specific properties and symmetries that are difficult to analyze and rebuild in synthetic data. Hence, to predict the performance of an algorithm in a certain application, using only synthetic data is unrewarding, since experiments involving sample instances stemming from this application are often more accurate.

This raises the question of why to use synthetic data at all. There are some good arguments that justify the use of synthetic data, at least together with real world data:

- Tunable characteristics, as for example the density of the generated graphs, allow to evaluate algorithms in detail depending on these characteristics. A scenario where this can be useful is when an algorithm yields good results for some networks but bad results on others. A study on a large set of generated graphs might help to

identify characteristics of the graph that are difficult to handle for the algorithm, which in turn might raise some potential for improvements.

- Synthetic graphs can usually be generated in any possible size, even very large networks that might not (yet) exist in practice. This is especially useful in the context of scalability studies.
- Using a graph generator, an unlimited number of different networks with similar properties can be generated, preventing algorithms to focus only on very few benchmark instances. This permits to test algorithms on a representative sample of the graph class one is interested in, ensuring some degree of significance.
- In particular in the context of graph clustering, there is another reason why synthetic networks are quite popular. Since there is no general agreement on a single objective function evaluating the goodness of the clustering, a common approach to evaluate graph clusterings independent of any objective function is the comparison to a known ground truth clustering. The downside of this is that real world graphs with a well-motivated ground truth clustering are still rare. For this reason, synthetic networks incorporating a hidden ground truth clustering that has been used in the generation process are widely used.

Probably the most fundamental model for synthetic graphs are graphs where every edge exists with a fixed, constant probability. The *planted partition* model [37, 56, 87], also called *ad hoc* model, is a simple modification thereof that takes as input a given ground truth clustering and two parameters p_{in} and p_{out} that correspond to the linking probabilities between vertices in the same and different clusters. We already encountered such random graphs in the experiments described in Chapter 5.

Contribution. In this chapter, we augment the planted partition model by allowing *dynamic events*; edge and vertex events add or delete an edge or vertex, whereas cluster events split or merge clusters. More formally, we generate a time series of random graphs $G_0, \ldots, G_n$, where G_t emerges from G_{t-1} via exactly one *atomic update*, i.e., the insertion or deletion of an edge or vertex. Over the whole generation process, the generator keeps track of a (dynamic) ground truth clustering. The probability of atomic events is chosen in a way that adheres to this clustering, without losing randomness. Graph growth/shrinkage and cluster dynamics can be simulated, steered by input parameters. Together with the benefit of the reference clustering, this can be used to thoroughly evaluate dynamic graph clustering algorithms, i.e., algorithms that incrementally update the calculated clustering as new node/edge events occur. A preliminary version of this generator is documented in the technical report by Görke and Staudt [104], and the dissertation of Görke [94], and has been used in [100]. The new generator documented here differs fundamentally in the data structures used, which allows for faster practical and worst case running time, as well as linear space complexity. As the random model and parameters used are taken from the old generator, their description closely adheres to the technical report. Our generator is free for use and can be downloaded as Java software from our project page [1].

A large part of the credit goes to our former student Roland Kluge, who implemented the generator in his bachelor thesis [138]. I had the initial idea to improve the performance by using virtual Fisher-Yates shuffles for the individual clusters, supervised Roland's thesis

[1] http://i11www.iti.uni-karlsruhe.de/en/projects/spp1307/dyngen

together with Robert Görke, proved Theorem 8.1, and wrote the conference paper [98] this chapter is based upon.

8.1 Related Work

In the following, we give a short overview of models for synthetic graphs and benchmark instances occurring in the context of clustering evolving networks. We focus on models especially suited for complex networks that try to explain network evolution, and on benchmark instances that incorporate a hidden ground-truth clustering together with approaches to make these benchmarks dynamic.

Random graphs where edge probabilities are given by a constant and independent from each other have been first considered by Gilbert [89] in 1959. Until then, a lot of effort has been put into alternative models that better capture the properties of real world networks which typically exhibit characteristics like small diameter, high clustering coefficient and a powerlaw degree distribution [171]. Two classical models are small world networks [224] that explicitly address the first two issues and the Barabási-Albert model [20] that mostly addresses the degree distribution. The latter can be seen as a dynamic model for graph growth according to a preferential attachment process. Numerous variations thereof exist, most of which are targeted in capturing more accurately properties observed in real world social networks [140, 150, 217]. Based on the associated likelihood values, Leskovec et al. [149] automatically determine, among a set of parameterized models, the one fitting a set of four large online networks best. Similarly, Patro et al. [177] propose to use an evolutionary algorithm to choose among a set of parameterized models of network growth the one fitting a given set of input characteristics best, in order to automatically learn the best model for different graph classes. In their experiments, Brandes and Mader [39] use exponential-family random graphs [184] as basis and describe the evolution between two networks with the help of stochastic actor-oriented models [205]. Both steps rely on properties of real-world dynamic networks that are given as input.

Although these models incorporate network growth and already reflect common properties of observed complex networks as for example online social networks very well, they do not come with a well motivated inherent ground truth clustering that can be used to evaluate clustering algorithms. An exception to this is the model by Zheleva et at. [235] that is especially targeted at modeling the growth of social networks where vertices can additionally choose to enter special groups of interest. Here, the assumption is that both the network and the group structure evolve simultaneously, influencing each other. It might be possible to use the group structure chosen by the vertices as a ground truth clustering for overlapping clusters, although the group structure is correlated to the network only to a certain extent. In the model of Bagrow [18], starting from a graph generated according to preferential attachment, edges are randomly rewired to incorporate a given ground truth clustering. While this approach combines a ground truth clustering with a realistic degree distribution, the evolution stemming from the preferential attachment process is lost.

For static graph clustering, two synthetic benchmark sets have been used very frequently in the literature; the GN benchmark introduced by Girvan and Newman [91] and the LFR benchmark introduced by Lancichinetti et al. [145]. The GN benchmark is based on the planted partition model; typically, it is used to determine how well an algorithm is able to recover the ground truth clustering, depending on the gap between

p_{in} and p_{out}. The planted partition model has been generalized to weighted [75] and bipartite [113] graphs as well as hierarchical [236] and overlapping [191] ground truth clusterings. Closely related are relaxed caveman graphs [5, 223]. Among the dynamic graph clustering algorithms that have been evaluated with the aid of planted partition graphs are FacetNet [152], the approaches by Yang et al. [227] and Kim and Han [135], and the above mentioned algorithm framework by Görke et al. [99]. The former two evaluations use graphs from the GN benchmark and introduce dynamics based on vertex moves; in each time step, a constant fraction of vertices leave their cluster and move to a random one. Kim and Han additionally consider a dynamic network that also incorporates the forming and dissolving of clusters and vertex addition and deletion.

In the LFR benchmark, cluster sizes as well as vertex degrees are expected to follow a power law distribution. Similar to the planted partition model, vertices share a certain fraction of their links with other vertices in their cluster and the remaining links with random vertices in other parts of the graph. The LFR benchmark has been generalized to weighted and directed graphs, as well as to overlapping clusters [144]. Dinh et al. [65] have used a modification of this benchmark to a dynamic setting. Furthermore, Green et al. [106] use dynamic benchmarks based on LFR graphs that incorporate different cluster events, including membership switching, cluster growth, shrinkage, birth and death, and the merge and split of clusters. After the ground truth clustering has been adapted, a new random graph is drawn according to the mechanisms of the LFR benchmark, which results in large differences between adjacent timesteps.

Aldecoa and Marín [6] finally suggest to interpolate between two graphs with a significant clustering structure by rewiring edges at random. This is proposed as an alternative to benchmarks like the GN or LFR benchmark in the context of static clustering algorithms. Here, the assumption is that clusterings of the intermediate states of the graph during the rewiring process should have low distance to both the ground truth clustering of the initial and the final state. The rewiring process could be seen as a model for community evolution. In the context of tracking clusterings over time, Berger et al. [27] do not consider models for dynamic graphs but two scenarios for the evolution of clusters that are more sophisticated than random vertex moves or cluster splits and merges. It remains to mention that, in principle, all generative models used to infer clusterings of dynamic graphs via a Bayesian approach [152, 210, 227] might also be used as benchmark instances, as they naturally come with a dynamic ground truth clustering.

Other models for dynamic graphs based on random evolution according to a given Markov chain include *edge-Markovian Dynamic Graphs* [23, 54]. This model does not incorporate a reference clustering but uses two fixed parameters p and q that represent the *edge birth-rate* and *edge death rate* of each possible edge. In contrast to the model we consider, an arbitrary number of edge deletions and insertion can take place in each time step.

8.2 Static Model for Planted Partition Graphs

The first graph G_0 created by our generator is based on Gilbert's static model with uniform edge probability [89], which we modified to incorporate a planted partition [37, 87]. The idea behind this modification, which we will call $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$, is that vertices in the same cluster should be linked with high probability p_{in}, whereas intercluster edges should be present with a lower probability p_{out}, i.e., we always assume $p_{\text{in}} > p_{\text{out}}$.

The parameter n denotes the number of vertices. Edges are added randomly according to the following process. The generation is based on a fixed *ground truth* clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ of the vertices. We set p_{out} to a single value, whereas p_{in} is a list of length k: $p_{\text{in}} = \big(p_{\text{in}}(C_1), \ldots, p_{\text{in}}(C_k)\big)$. For two vertices u and v the probability for edge $e = \{u, v\}$ to exist in a graph created with $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ is called *edge probability* $p(e) = p(u, v)$, where

$$p(u,v) = \begin{cases} p_{\text{in}}(C_i) & u, v \in C_i \\ p_{\text{out}} & \text{otherwise} \end{cases}$$

The probability of a particular graph G according to this model is thus

$$p(G) = \prod_{e \in E} p(e) \cdot \prod_{e \in \overline{E}} (1 - p(e))$$

Our dynamic generator is strongly based on this concept, and the first graph in the generated sequence is a $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ graph. The number of clusters of the ground truth clustering, as well as a list of intracluster probabilities p_{in} and the uniform intercluster probability p_{out} are input parameters. Cluster sizes can either be set manually or determined automatically by the generator. In the latter case, we choose the cluster of a vertex uniformly at random, which entails a binomial distribution of the cluster sizes with mean $\frac{n}{k}$.

Furthermore we introduce a coefficient $\beta \in \mathbb{Q}$ which *skews* the binomial distribution as follows ($\beta = 1$ in the unskewed case): Each cluster C_i is assigned to a subinterval $\left[\frac{i-1}{k}, \frac{i}{k}\right)$ of $[0, 1)$. When searching for a cluster to add a new vertex to, we draw an integer $i \in [0, k-1]$. Now, we add the vertex to the cluster which is assigned to the surrounding interval of $\left(\frac{i}{k}\right)^{\beta}$. Examples for cluster size distributions with different values of β can be found in the technical report of Görke and Staudt [104].

8.3 Edge Dynamics

Neglecting cluster dynamics for the moment, we describe the random process we use for edge dynamics, along with some theoretical properties of the random sequence generated. We further give details on how this process is implemented in our generator, together with worst case guarantees on running times.

8.3.1 Associated Markov Chain and Distribution

Ideally, we would like to have a random process that triggers an edge operation, i.e., insertion or deletion, in each time step such that the relative frequency of a graph G in the resulting sequence follows its probability $p(G)$ in the $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ model. Unfortunately, such a random process does not have to exist in general: Consider for example the simple case that we have two vertices and the probability of the edge between these equals 0.1. Under these assumptions, the probability of the empty (complete) graph on two vertices equals 0.9 (0.1), respectively. On the other hand, there is only one possible edge operation in each state. Hence, each random sequence with an operation in each

step alternates between these states, which can never lead to the graph distribution resulting from the assumed edge probabilities.

There is however a simple random process that follows prescribed edge probabilities if we allow for repeated occurrences of a graph in the sequence. This process chooses in each step a pair of vertices u and v uniformly at random, deletes the edge $\{u, v\}$ if it exists and (re)inserts it with probability $p(u, v)$. There is a natural correspondence between this procedure and a Markov chain M' whose states represent all possible (labeled) graphs on n vertices. It is not hard to see that $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ is the unique stationary distribution of this chain. Thus, if we choose our initial state according to this distribution, the expected relative frequency of a graph generated by this Markov chain follows $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$.

However, in the context of evaluating dynamic algorithms, we are typically interested in sequences of graphs such that consecutive graphs follow from each other by atomic changes. We therefore slightly modify M' such that each time step that does not change the graph is discarded and call the resulting Markov chain M. Let $P_E := \sum_{e \in E}\big(1-p(e)\big)$ and $P_{\overline{E}} := \sum_{e \in \overline{E}} p(e)$. Let $p := \binom{n}{2}$ denote the number of vertex pairs in the graph. Then, the probability $p_{\text{del}}(u, v)$ that one step of M deletes an existing edge $\{u, v\}$ is given by the quotient of the probability that one step of M' deletes $\{u, v\}$ and the probability that this step causes an edge flip, i.e., the deletion of an existing or the insertion of a non-existing edge:

$$p_{\text{del}}(u, v) = \frac{\frac{1}{p} \cdot \big(1 - p(u, v)\big)}{\sum_{e \in E}\left[\frac{1}{p} \cdot \big(1 - p(e)\big)\right] + \sum_{e \in \overline{E}}\left[\frac{1}{p} \cdot p(e)\right]} = \frac{1 - p(u, v)}{P_E + P_{\overline{E}}}$$

Similarly, the probability $p_{\text{ins}}(u, v)$ of inserting a non-existing edge is

$$p_{\text{ins}}(u, v) = \frac{\frac{1}{p} \cdot p(u, v)}{\sum_{e \in E}\left[\frac{1}{p} \cdot \big(1 - p(e)\big)\right] + \sum_{e \in \overline{E}}\left[\frac{1}{p} \cdot p(e)\right]} = \frac{p(u, v)}{P_E + P_{\overline{E}}}.$$

Intuitively, we expect the relative frequency of unlikely states in the sequence generated by M to be slightly larger than in the sequence generated by M' and vice versa, as they are less often discarded. The following theorem makes this intuition precise.

Theorem 8.1. *If we choose the initial graph G_0 according to $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$, the expected relative frequency of a graph $G = (V, E)$ in a sequence R generated by M is*

$$p(G) \cdot \left[\frac{\sum_{e \in E}\big(1 - p(e)\big) + \sum_{e \in \overline{E}} p(e)}{2 \cdot \sum_{e \in \binom{V}{2}} p(e)\big(1 - p(e)\big)}\right]$$

Proof. We use that M is derived from M' and look at the random sequence $R' = G'_1, \ldots, G'_{T'}$ with $T' \geq T$ that M' generates. We call an occurrence of a graph at time step $t \geq 1$ *valid*, if $G'_t \neq G'_{t-1}$. The probability that G has a valid occurrence at a certain time step t is then

$$p(G'_t{=}G) \cdot p(G'_{t-1}{\neq}G \mid G'_t{=}G) = p(G'_t{=}G) \cdot [1 - p(G'_{t-1}{=}G \mid G'_t{=}G)] \qquad (8.1)$$

If we use that the probability of G at any time step equals $p(G)$, the probability that no edge in G is changed from $t-1$ to t equals

$$p(G'_{t-1}{=}G \mid G'_t = G) = \frac{p(G'_{t-1}{=}G)}{p(G'_t{=}G)} \cdot p(G'_t{=}G \mid G'_{t-1} = G) = \sum_{e \in E} \frac{p(e)}{\binom{n}{2}} + \sum_{e \in \overline{E}} \frac{1-p(e)}{\binom{n}{2}}$$

Hence, the right hand side of Equation 8.1 is equal to

$$p(G) \cdot \left[1 - \sum_{e \in E} \frac{p(e)}{\binom{n}{2}} - \sum_{e \in \overline{E}} \frac{1-p(e)}{\binom{n}{2}}\right] = p(G) \cdot \frac{1}{\binom{n}{2}} \cdot \left[\sum_{e \in E} \big(1-p(e)\big) + \sum_{e \in \overline{E}} p(e)\right] \tag{8.2}$$

The probability that a time step contains a valid occurrence of an arbitrary graph is the sum over all pairs of vertices that the corresponding edge is flipped in this time step, which equals $2 \cdot \frac{1}{\binom{n}{2}} \cdot \sum_{e \in \binom{V}{2}} p(e)(1-p(e))$. Dividing the probability of a valid occurrence of G at time step t (Equation 8.2) by this probability yields the claim. □

As an example illustrating the effect of the factor in Theorem 8.1, consider the simple case that the graph consists of one cluster such that the probability of each potential edge e equals $p(e) = \frac{1}{4}$. As $p(e) < \frac{1}{2}$, the complete graph is more unlikely than the empty graph. Indeed, the factor for the former equals 2 and for the latter $\frac{2}{3}$, which means that the complete graph occurs more often and the empty graph less often in the sequence generated by M, compared to their probability according to $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$. If the random graph matches the expectation in the sense that $m = \frac{1}{4}\binom{n}{2}$, we get an even smaller factor of $\frac{1}{2}$.

Until now, we have assumed that the initial graph is a $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ graph and that the following time steps are obtained by mere edge dynamics. As motivated in the introduction, our generator also incorporates vertex and cluster dynamics, which we will introduce later on. The latter two types of dynamics disturb the probability distribution in a way that is difficult to analyze. However, as it is always possible to reach any graph from any other graph in a finite number of steps with a positive probability, M is irreducible. The expected relative frequency in Theorem 8.1 is therefore the only stationary distribution of M and we can expect the relative frequency of graphs to get close to this distribution after a sufficiently large number of edge operations following vertex or cluster events [25].

8.3.2 Data Structures and Implementation

Before we introduce the dynamic data structures our generator builds upon and show how these data structures can be used to efficiently implement edge dynamics, we briefly describe how vertex pairs can be enumerated continuously, which corresponds to the enumeration of potential edges in a graph. We use a bijection between pairs of vertices and integers between 0 and $\binom{|V|}{2}$ proposed by Batagelj and Brandes [22]. Figure 8.1 intuitively illustrates this bijection by using the adjacency matrix of the graph. As we only consider undirected graphs, potential edges correspond to the entries below the diagonal. These entries can be enumerated by traversing this sub matrix row by row. For given vertices u and v, the index $e(u, v)$ can be obtained by

$$e(u,v) = \sum_{k=0}^{u-1} k + v = \frac{1}{2}(u-1)u + v$$

Vice versa, given the edge index $e(u,v)$, the corresponding vertices u and v can be found as follows:

$$u = 1 + \left\lfloor -\frac{1}{2} + \sqrt{\frac{1}{4} + 2 \cdot e(u,v)} \right\rfloor$$

$$v = e(u,v) - \frac{1}{2}u(u-1)$$

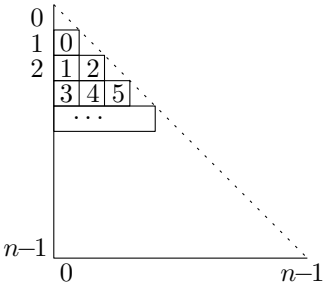


FIGURE 8.1: Indexing scheme

Random Binary Selection Tree. In order to choose the next edge to be added or deleted, we need a data structure that allows us to efficiently draw an element from a weighted set $O = \{o_1, \ldots, o_n\}$ such that the probability to choose a certain element is proportional to its weight. Given such a data structure, a naïve generator could simply store each potential edge $\{u,v\}$ with weight $p(u,v)$ and each non-edge with weight $1 - p(u,v)$ and iteratively draw edges to add or delete. Later on, we will show that one entry for each cluster rather than for each edge is sufficient.

A simple solution for such a data structure is an array A storing prefix sums of the weights, i.e., $A[k] = \sum_{i=1}^{k-1} w(o_i)$, $1 \leq k \leq n+1$. Let W be the sum of all weights. Then we can draw a uniformly distributed random number x in the interval $[0, W)$ and use binary search to find the index k such that $A[k] \leq x < A[k+1]$. It is easy to see that this process selects each element o_k with probability $w(o_k)/W$. For static sets this approach works well, however, we will need to update weights frequently, and to add and delete elements occasionally. Updating the prefix sums has linear worst-case complexity, which we would like to avoid.

We therefore use a complete binary tree to store the elements. We define each vertex of this tree to be a tuple $q_i = (o_i, w_i, l_i, r_i)$, where o_i is an element, w_i is its associated weight $w(o_i)$, l_i is the sum of the weights in the left subtree and r_i is the sum of weights in its right subtree. The weight l_m and r_m of a leaf q_m are simply 0. Contrary to the prefix sum array, inserting and deleting elements as well as weight updates can be done in logarithmic time. To keep the tree balanced, new elements are inserted as leaves with minimum distance to the root and deleted elements are replaced by a leaf element with maximum distance to the root. Afterwards, weight changes have to be propagated on the path(s) to the root.

The procedure for the selection of an element starts at the root s by drawing a random number x from the interval $[0, l_s + w_s + r_s)$. Now there are three possible ranges for x: if $l_s \leq x < l_s + w_s$, the element is returned; if $x < l_s$, the carryover x is sent to the left subtree; and if $l_s + w_s \leq x$, the carryover $x - w_s - l_s$ is sent to the right subtree. The procedure continues recursively from there until an element is returned after at most $O(\log n)$ steps. The correctness of the selection process can be seen by constructing an array with prefix sums of the weights such that elements in the array are ordered according to an inorder traversal of the tree. Selecting an element in the tree is equivalent to a binary search in this array. An example for a random binary selection tree and the corresponding array can be found in Figure 8.2.

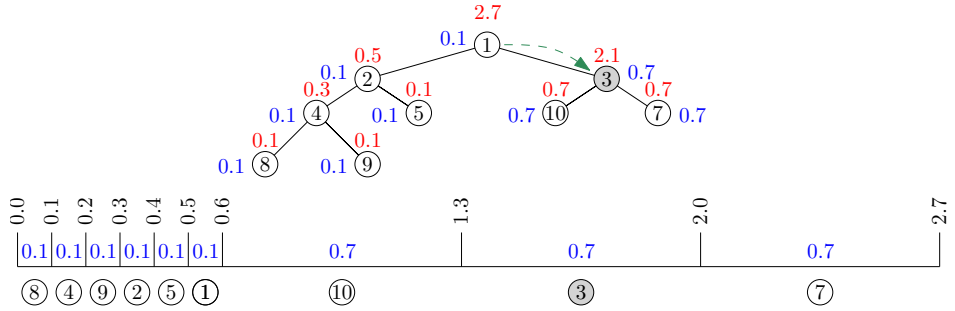


FIGURE 8.2: Example of a random binary selection tree and its associated array. Blue numbers are weights of individual elements, whereas red number correspond to the accumulated weight of subtrees. Random number 1.08 in $[0, 2.7)$ guides the selection process through the tree.

Virtual Fisher Yates Shuffle. Recall that our generator considers a single value p_{out} and a list p_{in} of values for the clusters, i.e., edges between pairs of vertices in the same cluster are equiprobable. Using a binary tree to store all intracluster vertex pairs is thus quite inefficient. Instead, we use for each cluster a modified *Fisher-Yates shuffle* [78] that maintains the information which intracluster edges exist in the graph.

A Fisher-Yates shuffle is a simple method to uniformly sample without replacement from a given set of N elements. The elements are stored in an array of size N with indices from 0 to $N-1$ and the *border index* i of the shuffle is initially set to 0. In each step, a random number r between i and $N-1$ is drawn, the corresponding element at index r is marked as selected and swapped with the element at index i. Then, the border index i is increased by 1. It is easy to see that each element can only be chosen once and the probability to choose each subset of size k is the same. For our purposes, the elements correspond to all potential edges within a cluster, and the selected elements, which are stored in the first i entries of the array, to edges currently existing in the graph.

A drawback of this approach is that we have to enumerate and store each element explicitly. For elements that can be easily enumerated, it is more efficient to store an implicit representation of this array one of which is the *virtual Fisher-Yates shuffle* introduced by Batagelj and Brandes [22].

Let i be the number of elements drawn so far, L be the set of indices smaller than i that have not yet been drawn and H the set of indices at least i that have been drawn. In our view, the indices in L and H are "exceptions from the rule that small indices are selected and large indices unselected". The crucial observation is that the cardinality of L and H is equal. Hence, we can define a bijection from H to L and store it in a map `replace`. Similarly to the original Fisher-Yates shuffle, we can now iteratively draw a random number r between i and $N-1$. If $\texttt{replace}(r) = \perp$, i.e., if there is no entry for r in the map, index r has not yet been selected and we select the corresponding element. If $\texttt{replace}(r) = s$, we choose index s instead. This process guarantees that we draw in each step an unselected element with uniform probability.

After we have selected the element, we have to update the map such that it is still guaranteed that each element in L is assigned to a corresponding element in H. Depending on the previous state of the shuffle, we have to consider the four cases shown in Table 8.1. Entries of the form $\texttt{replace}(x) = y$ are depicted as arrows from x to y and the thick line marks the border between elements with index less than i and larger i. It is

TABLE 8.1: Illustrative figures for *select*. Index $r \in \{i, \ldots, N-1\}$ is drawn uniformly at random.

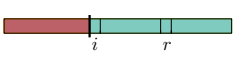
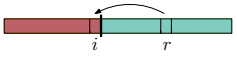
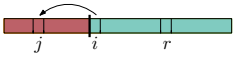
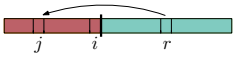
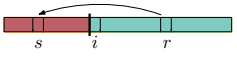
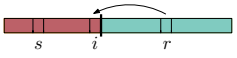
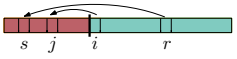
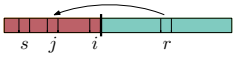
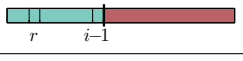
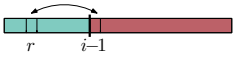
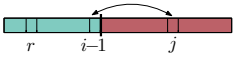
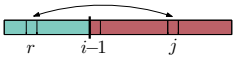
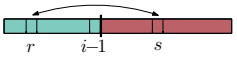
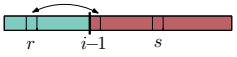
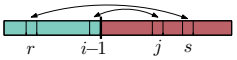
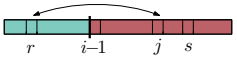
	Initial state	Final state
Case 1		
Case 2		
Case 3		
Case 4		

TABLE 8.2: Illustrative figures for *deselect*. Index $r \in \{0, \ldots, i-1\}$ is drawn uniformly at random.

	Initial state	Final state
Case 1		
Case 2		
Case 3		
Case 4		

easy to see that in each case a constant number of lookup, delete and insert operations in `replace` suffices. If we assume that the data structure we use for the map `replace` is a binary search tree, these operations take logarithmic time. Hence, the time complexity of choosing the next element is in $O(\log N)$.

Unlike the generation of static random graphs, we also need to delete edges over time. To this end, we have to modify the virtual Fisher-Yates shuffle slightly to deselect already selected numbers, i.e., putting them once again in the set of selectable elements. This can be done by making the pointers bidirectional, i.e., for each entry of the form $\texttt{replace}(j) = i$, we add a corresponding entry $\texttt{replace}(i) = j$. Deselection now works analogously to selection: First, we draw a random number r in the interval $[0, i-1]$. If there is no entry for r in `replace`, r is a currently selected index. In this case, we undo the selection by setting $\texttt{replace}(i-1) = r$ and vice versa and move the border one index to the left. If r is an unselected element, we undo the selection of $\texttt{replace}(r)$ instead. Special care has to be taken if the element at index $i-1$ is also unselected, i.e., if $\texttt{replace}(i-1) \neq \perp$. Figure 8.2 depicts all possible cases and the corresponding updates of the map. Deselection is therefore just as efficient as selection. For each selected element, we have to store at most two entries in the map, hence the space requirement is linear in the number i of currently selected elements.

Overview of Selection Procedure. In this section we explain how we use a combination between a random binary selection tree and virtual Fisher-Yates shuffles to

efficiently implement edge dynamics according to the random model described in Section 8.3.1. For the sake of simplicity, we first assume that we only want to draw intracluster edges, i.e., $p_{\text{out}} = 0$. In this case, the following procedure can be used.

Each cluster has an associated shuffle that stores all intracluster edges in the cluster. This shuffle is used to uniformly select edges in the cluster to delete or to insert. To be able to use the enumeration scheme described above, each vertex v receives two ids, a global id that unambiguously identifies the vertex during the whole generation process and a local id in the range $[0, |\mathcal{C}(v)| - 1]$. The local id is used to continuously enumerate intracluster edges in the individual shuffles; it may change if vertices are inserted or deleted from the cluster.

On top of that, we store two randomized binary selection trees Γ_{ins} and Γ_{del} that each contain one entry for every cluster. The weight of each cluster C in Γ_{ins} corresponds to the sum of the probability weight of edge insertions within the cluster, $\overline{e_C} \cdot p_{\text{in}}(C)$. Similarly, its weight in Γ_{del} is defined as $e_C \cdot \big(1 - p_{\text{in}}(C)\big)$. The overall process of selecting the next edge operation is now divided into three steps:

1. As introduced in Section 8.3.1, let $P_E = \sum_{\{u,v\} \in E} \big(1 - p(u,v)\big)$ and furthermore $P_{\overline{E}} = \sum_{\{u,v\} \notin E} p(u,v)$. With probability $P_E/(P_E + P_{\overline{E}})$ we decide to delete and with probability $P_{\overline{E}}/(P_E + P_{\overline{E}})$ to insert an edge.
2. For edge deletions, we choose a cluster C in Γ_{del} according to the stored weights. Similarly, we choose a cluster in Γ_{ins} if we have decided to insert an edge.
3. Depending on the choice in the first step, we insert or delete an edge in the virtual Fisher-Yates shuffle associated with C.

Finally, the weight of C in Γ_{ins} and Γ_{del} has to be updated. Later on, we will show that this process inserts an intracluster edge between two unconnected vertices u and v with probabilities according to the random process described in Section 8.3.1. It remains to describe how this procedure can be altered to be able to deal with intercluster edges.

Dealing with Intercluster Edges. In principle, it would be possible to handle intercluster edges analogously and just introduce a virtual Fisher-Yates shuffle containing all pairs of vertices in different clusters. As all of these vertices exist with the same probability, this would be perfectly feasible. The problem with this approach is that, unlike the local ids we use to identify vertices in the individual clusters, it is not easy to continuously enumerate intercluster vertex pairs in the presence of vertex and cluster dynamics. For this reason, we introduce a shuffle for a *pseudocluster* containing all vertices in the graph. This pseudocluster gets an entry in Γ_{ins} with weight $\overline{m} \cdot p_{\text{out}}$. As the pseudocluster contains all vertices, the associated shuffle also contains intracluster vertex pairs. Hence, it is possible to draw intracluster edges either in the shuffle of the corresponding cluster or in the shuffle of the pseudocluster, which overestimates the probability of choosing such edges. To correct this, we exploit our assumption that $p_{\text{in}}(C) > p_{\text{out}}$ for each cluster C and decrease the weight associated with C in Γ_{ins} to $\overline{e_C} \cdot (p_{\text{in}}(C) - p_{\text{out}}) \geq 0$.

For edge deletions, this trick cannot be used as $1 - p_{\text{out}}$ is larger than $1 - p_{\text{in}}(C)$. This is why we introduce an additional array storing the global id of all intercluster edges and draw a random edge in this array in case we decide to delete an intercluster edge. The respective edge is then both deleted in the array and in the shuffle of the

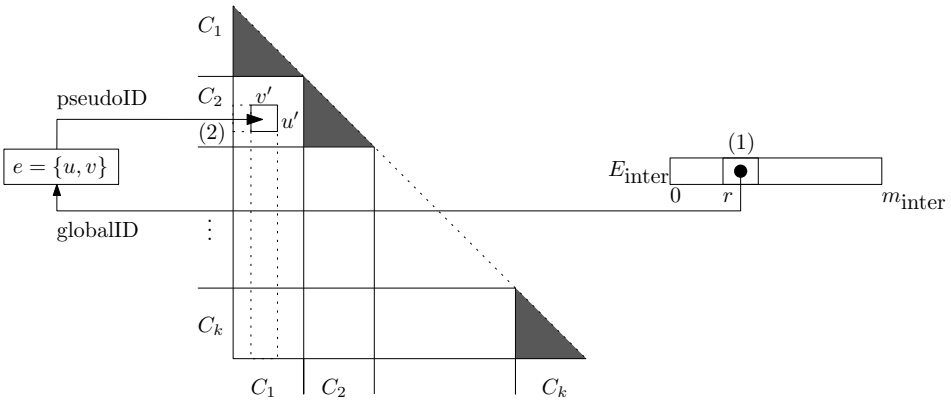


FIGURE 8.3: Example for a *deselect* operation in the pseudocluster: r is the integer drawn uniformly at random in the range of $0, \ldots, x_e - 1$ and u', v', resp. u, v are the local, resp. global vertex indices of the endpoints of the selected edge. The *white* area consists of the intercluster edges and *dark gray* areas contain all (forbidden) intracluster edges.

pseudocluster. This procedure guarantees that we do not erroneously delete intracluster edges by choosing from the pseudocluster, which is why the weight of the true clusters in Γ_{del} remains unchanged. The weight of the pseudocluster in Γ_{del} corresponds to $x_e(1 - p_{\text{out}})$. Whenever an intracluster edge is inserted or deleted, either in the pseudocluster or in its own cluster, the corresponding entry in the other shuffle has to be updated. Asymptotically, this does not increase running times and space requirements. Figure 8.3 illustrates the data structure used for the pseudocluster and the process of deleting an intercluster edge.

Correctness and Time Complexity

In the following, we show that the selection process for edge dynamics described up to now follows the random model introduced in Section 8.3.1.

Proposition 8.2. *The described process inserts an edge between two unconnected vertices u and v with probability*

$$p_{ins}(u, v) = p(u, v)/(P_E + P_{\overline{E}}).$$

Proof. The probability of an edge insertion is $P_{\overline{E}}/(P_E + P_{\overline{E}})$. We first consider the case that u and v are both contained in a cluster C. The probability of choosing C in Γ_{ins} is $\overline{m}(C)/P_{\overline{E}} \cdot (p_{\text{in}}(C) - p_{\text{out}})$ and the probability of choosing the pseudocluster in Γ_{ins} is $\overline{m}/P_{\overline{E}} \cdot p_{\text{out}}$. Similarly, the probability of choosing $\{u, v\}$ in the associated shuffles is $1/\overline{m}(C)$ and $1/\overline{m}$, respectively. Hence, these probabilities sum up to

$$\frac{P_{\overline{E}}}{P_E + P_{\overline{E}}} \cdot \Big[\frac{\overline{m}(C)}{P_{\overline{E}}} \cdot (p_{\text{in}}(C) - p_{\text{out}}) \cdot \frac{1}{\overline{m}(C)} + \frac{\overline{m}}{P_{\overline{E}}} \cdot p_{\text{out}} \cdot \frac{1}{\overline{m}}\Big] = \frac{p(u, v)}{P_E + P_{\overline{E}}}$$

If $\{u, v\}$ is an intercluster pair, it can only be chosen via the pseudocluster. Hence, the corresponding probability is

$$\frac{P_{\overline{E}}}{P_E + P_{\overline{E}}} \cdot \frac{\overline{m}}{P_{\overline{E}}} \cdot p_{\text{out}} \cdot \frac{1}{\overline{m}} = \frac{p(u, v)}{P_E + P_{\overline{E}}}$$

□

A very similar proof yields the following proposition stating that deletions also follow the specified random model.

Proposition 8.3. *The described process deletes an existing edge* $\{u, v\}$ *with probability*

$$p_{del}(u, v) = \big(1 - p(u, v)\big)/(P_E + P_{\overline{E}})$$

Deciding whether to insert or delete an edge is done in constant time. For both operations, choosing the cluster in the respective cluster tree takes time $O(\log k)$, where k denotes the current number of clusters. Similarly, selection or deselection in the corresponding virtual Fisher-Yates shuffle takes time logarithmic in the size of the shuffle. Hence, the expected time complexity for both operations is in $O(\log n)$. As the total size of the shuffles is asymptotically bounded above by the number of edges of the current graph, it is easy to verify that the total space requirement is linear in the size of the graph.

8.4 Vertex and Cluster Dynamics

Assessing dynamic clustering algorithms usually involves the question whether the algorithm is able to follow changes in the ground truth clustering. This is why we also included the possibility to incorporate vertex and cluster dynamics.

Vertex Dynamics. Vertex dynamics are steered by a parameter p_χ that specifies the probability that instead of an edge operation, a vertex operation occurs, i.e., we delete or insert a vertex. If a vertex operation is to be performed, another parameter p_ν determines the probability that this operation is a vertex insertion. Choosing p_ν to be smaller or larger than 0.5 gives the opportunity to simulate graph growth or shrinkage. For deletion, a vertex is chosen uniformly at random and all incident edges are deleted. To adhere to the initial cluster size distribution, new vertices are assigned to clusters according to expected cluster sizes, similar to the generation of the initial clustering. To dampen the impact of vertex dynamics on the edge distribution and to stay closer to the $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ model, new vertices are immediately connected to other vertices according to the prescribed edge probabilities. Naïvely, this takes $O(n)$ time, however, it is possible to use the *geometric method* introduced by Fan et al. [74] and used by Batagelj and Brandes [22] to reduce the running time to $O(\deg v)$, where $\deg v$ is the resulting degree of the new vertex v.

It remains to explain what has to be done to update the data structures. Updating the affected entry in the random binary selection trees associated with the clusters takes $O(\log k)$ time. If a new vertex is inserted or deleted, the index space of the shuffle of its cluster and of the pseudocluster has to be adapted. For insertion, it suffices to assign the highest local vertex id in the cluster to the new vertex and increase the index space of the shuffle accordingly. If a vertex is deleted, we have to guarantee that the index space is still continuous. We do this by relabeling the vertex v_f with

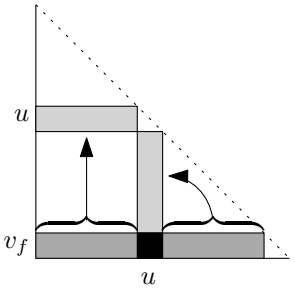


FIGURE 8.4: Update if vertex with local id u is deleted.

the previously largest local id in the shuffle of the deleted vertex u by the id of u. The same procedure has to be performed for the vertex w_f with the largest local id in the pseudocluster. Figure 8.4 illustrates this case.

For the relabeling step, we first delete all edges incident to v_f or w_f in the shuffle and then reinsert them using the new edge ids. Hence, the overall expected time complexity of the deletion of vertex v is $O\big((\deg v + \deg v_f + \deg w_f) \cdot \log n\big)$, whereas for vertex insertion we get $O\big(\deg v \cdot \log n\big)$.

Cluster Dynamics. Cluster dynamics is independent of vertex and edge dynamics in the sense that in each time step, additionally to the vertex or edge operation performed, a cluster operation can take place. The probability of a cluster operation is determined by the input parameter p_ω and the probability that this cluster operation merges two clusters is determined by the parameter p_μ. With probability $1-p_\mu$, one of the clusters is split by assigning each of its vertices to one of the new clusters with uniform probability. The expected cluster size(s) of the resulting cluster(s) are sampled from a Gaussian distribution, being estimated from the current ground truth clustering.

To calculate new values for p_{in}, an obvious possibility is to just use the old value(s). For a cluster split, this means that the new clusters inherit p_{in} from the old cluster, whereas for a cluster merge, the new cluster is assigned the average of the intracluster edge probabilities of the participating clusters. This process leads to increasingly uniform p_{in} values over the course of time. A more detailed description can be found in the technical report of Görke and Staudt [104].

One of the main motivations for using $\mathcal{G}(n, p_{\text{in}}, p_{\text{out}})$ graphs for the evaluation of clustering algorithms is the knowledge of a ground truth clustering the result of the algorithm can be compared to. However, for cluster dynamics, it can easily be imagined that immediately after a split or merge operation, the clustering algorithm has no chance to detect the current ground truth clustering, as the change is not yet reflected in the edge structure. For this reason, the generator keeps track of an additional *reference clustering* that follows the current ground-truth clustering with some delay, roughly speaking, as soon as the involved subgraph becomes similar enough to the ground truth's expectation. A detailed description of the behavior of the reference clustering can be found in [104]. To prevent the interleaving of concurrent cluster events, as long as the change in the ground-truth is not propagated to the reference clustering, the participating clusters are not available for further cluster operations. If, for this reason, in some time step a cluster event is triggered but no available clusters are found, the cluster event does not take place.

To keep the data structures up to date, we delete the involved old shuffle(s) and create one or two new shuffles from scratch, depending on the kind of cluster operation. Furthermore, the additional array associated with the pseudocluster that stores all intercluster edges has to be updated. The reason for this is that for a split operation, new intercluster edges between the two parts arise, whereas cluster merges turn some intercluster edges into intracluster edges. If pointers are used to link the occurrences of an edge in different data structures, all these operations take $O\big((n(C) + e_C) \cdot \log n\big)$ time, where C is either the cluster that is about to be split or the new cluster after a merge. Removing the old cluster(s) and inserting the new cluster(s) in the cluster trees takes logarithmic time. Hence, this does not increase the (asymptotic) running time.

8.5 Experiments

We give a brief impression of the absolute running times of different sample configurations. All values are averaged over 15 runs and do not include the time to write the graph to hard disc. The first experiments in Figure 8.5(a) evaluate the running times for one million steps of the generator, while only the intracluster density varies. The planted partition contains 15 clusters and the cluster size distribution is unskewed ($\beta = 1$). The number of vertices as well as the intercluster edge probability $p_{\text{out}} = 0.1$ are constant. As the number of edges is in $\Theta(n^2)$, the time to generate the initial graph sometimes dominates the time needed for edge dynamics and is therefore not included in the plot. Similarly, as the expected degree of each vertex is in $\Theta(n)$, we didn't include vertex and cluster dynamics and set the corresponding probabilities to 0. As expected, the running time is almost independent of p_{in}. The low running times for the experiments with $p_{\text{in}} = 1$ can be explained by the fact that intracluster edges are never deleted or inserted and all dynamics involve intercluster vertex pairs. To obtain logarithmic worst case running time for edge operations, the map `replace` used for the virtual Fisher-Yates shuffles can be stored in a binary search tree. For comparison, we repeated the experiments with hash maps instead of these trees. It can be seen that for graphs of high density, hash maps yield better practical running times[2].

Figure 8.5(b) illustrates the running time for less artificial parameter settings. Here, the number of clusters equals $\sqrt{n}$ and the size distribution is skewed ($\beta = 0.5$). The probability of a vertex event instead of an edge event is set to 0.1 and in half of the cases a vertex is added (deleted). The probability of a cluster event is 0.01 and in half of the cases a cluster is split (two clusters are merged). The expected degree of a vertex is constant, which yields very sparse graphs. To give a more realistic impression of the total running time, we included the time to generate the initial graph, followed by 100 000 dynamic updates. As above, the running times obtained by using hash maps are better than for the tree based variant.

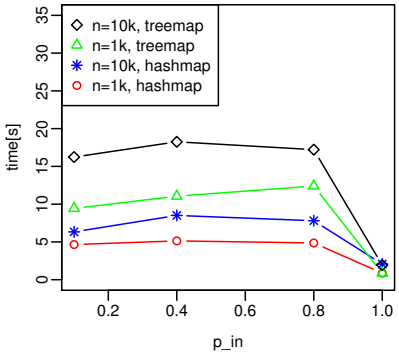
In summary, the experiments show that hash maps yield better practical performance and that dynamics can be added to the planted partition model without causing much overhead.

Implementation Notes. We conducted all experiments on a Dual-Core AMD Opteron Processor clocked at 2.6 GHz, using Java version 1.6.0_22. The machine has 32GB of RAM and 2×1 MB of L2 cache. The implementation uses no external libraries. As hash map, we used `java.util.HashMap`, whereas the tree-based implementation uses `java.util.TreeMap`, which is based on a red-black tree.

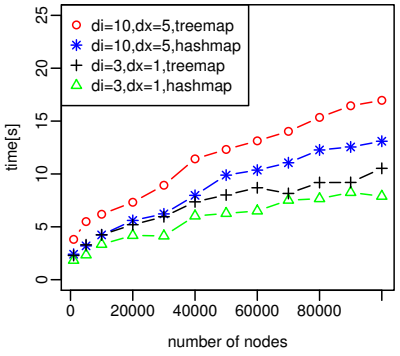
8.6 Concluding Remarks

We proposed a dynamic generalization of the planted partition model that can be used to evaluate dynamic graph clustering algorithms, with the additional benefit of a known reference clustering. Furthermore, we described how large dynamic random graphs according to this model can be efficiently generated and showed the practicability of this approach on selected example configurations. In order to make this model more

[2] Note that entries in the hash map are not distributed evenly over all possible indices, which is why we don't have expected constant time for all parameter settings



(a) Edge dynamics, constant number of vertices, variable p_{in}



(b) Full dynamics, constant expected intracluster (intercluster) degree `di` (`dx`), variable n

FIGURE 8.5: Running times for some sample configurations.

realistic, modifications similar to the static model are conceivable. Possible changes in the random model include vertex movements, less uniform degree distribution, higher clustering coefficient as well as the generalization to hierarchical reference clusterings.

Chapter 9

Conclusion

Building directly upon the intracluster density vs. intercluster sparsity paradigm introduced in Chapter 1, we investigated the generic problem of finding graph clusterings with guaranteed intracluster density and best possible intercluster sparsity, where the latter is based on the notion of sparse cuts in the graph. Besides systematically verifying whether our intuition that intracluster density strives towards coarse and intercluster sparsity towards fine clusterings, we investigated two metaheuristics for the problem, where the first is primarily based on merging clusters and the second on moving vertices between clusters. We discussed similarities between the former and SAHN algorithms for distance based clustering and examined under which circumstances efficient algorithms for SAHN clustering can be used in our scenario. An experimental evaluation of the move based heuristic revealed that it often yields better quality than the merge based heuristic, and that it efficiently addresses its own objective. We exploited this in a study on real world and synthetic networks to identify similarities and differences between different notions of intracluster density and intercluster sparsity.

The second part of this thesis was dedicated to a recently introduced clustering measure called surprise. A bicriteria view on surprise laid the foundation for the main contributions. We showed $\mathcal{NP}$-completeness of the associated decision problem, derived polynomial time algorithms for graphs of bounded tree width, and computed the first provably optimal solutions for small example graphs with a sequence of integer linear programs. Furthermore, we argued how the bicriteria view can be exploited to derive efficient heuristics and gave empirical evidence that the quality of the clusterings produced by an existing metaheuristic is implicitly due to this approach. Engineering this base algorithm led to algorithms that are able to cluster graphs with several millions of vertices, while still yielding competitive quality. We further discussed a new measure called SMod, which combines the configuration null model with the basic idea behind surprise.

The last chapter of this thesis dealt with the generation of synthetic test instances incorporating a hidden reference clustering that can be used to evaluate clustering algorithms for dynamic graphs. We described a more efficient implementation of the generator used by Görke et al. [99]; the main technical contribution is the reduction of the memory footprint to linear size using customized data structures, which renders the generation of significantly larger test data possible.

Table 9.1: Classification of measures with respect to bicriteria view

Pareto optimal with respect to	
maximizing i_e and minimizing i_p	maximizing i_e and minimizing i_{v^2}
2-expansion / 2-density	2-conductance
RB* [181, 182]	modularity [173]
surprise [15]	*S*Mod
constant/absolute Pott's model [212, 187]	multiresolution modularity [182]
performance/cluster editing/ correlation clustering [215, 196, 19]	

Bicriteria View. In the two main parts of this thesis, we considered two orthogonal ways to assess the quality of a clustering, the first one being based on the sparsity of the induced cuts and the second on the measure surprise, and compared the resulting clusterings to the ones obtained by optimizing modularity. A self-evident question one might ask is concerned with the similarities and differences between these two approaches. In both parts, we followed a bicriteria view on clustering; in the context of cut based measures, this was explicitly given by optimizing intercluster sparsity with constraints on the intracluster density, whereas for surprise, we argued that optimal clusterings are always Pareto optimal with respect to maximizing the number of intracluster edges i_e and minimizing the number of intracluster pairs i_p.

Although the latter is not true for the concrete intercluster measures we constructed in Chapter 4, the related problems to find a 2-partition optimizing expansion, density and conductance can be associated with the second kind of bicriteria view, which builds a bridge between the two approaches. Recall that the density of a cut was defined as

$$\mathsf{density}\big(\mathcal{C} = \{C, V \setminus C\}\big) = \frac{e_{C,V\setminus C}}{n_C n_{V\setminus C}} = \frac{m - i_e(\mathcal{C})}{p + i_p(\mathcal{C})}$$

Hence, optimal solutions with respect to 2-density, i.e., the problem of finding a 2-partition minimizing density, are Pareto optimal with respect to maximizing i_e and minimizing i_p. With similar arguments, it can be seen that the same holds for 2-expansion, whereas optimal solutions with respect to 2-conductance are always Pareto optimal with respect to maximizing i_e and minimizing the total sum of squared cluster volumes i_{v^2}, which relates conductance to modularity and *S*Mod.

Table 9.1 summarizes the classification of the objective functions considered in this thesis with respect to the two bicriteria problems associated with surprise and modularity. It also includes the objective that has been earlier considered by Reichardt and Bornholdt [181, 182], a variant of modularity using an Erdős-Rényi type random graph as null model, which we term RB*, the variant of surprise using the configuration model as null model, which we termed *S*Mod in Chapter 6, as well as the multiresolution variants of modularity and surprise. As we saw in Chapter 7, the multiresolution variant of surprise is known as constant [212] or absolute [187] Pott's model.

It turns out that other clustering measures from the literature can be integrated into this context as well. For example, the *performance* [215] of a graph clustering is defined as the fraction of vertex pairs $\{u, v\}$ that are correctly identified in the sense that either

u and v are in the same cluster and connected with an edge, or u and v are in different clusters and not adjacent; in terms of the number of intracluster edges and pairs, this can be written as follows, which makes it evident that performance belongs to the first column of Table 9.1:

$$\mathsf{performance}(\mathcal{C}) = \underbrace{\frac{1}{\binom{n}{2}}}_{=\text{const.}} \left[\underbrace{\binom{n}{2} - m}_{\text{const.}} + 2i_e(\mathcal{C}) - i_p(\mathcal{C}) \right]$$

It is not hard to see that performance is equivalent to the objective of *cluster editing* [196] with respect to optimum solutions, which in turn is equivalent to the version of *correlation clustering* minimizing disagreements [19]. Hence, these two approaches can be included into the above classification as well.

Note that, although both surprise and performance lead to small clusters in practice, the former allows for less dense clusters in general. From the above equation, it is evident that the density of a clustering with optimum performance, in the sense of the number of intracluster edges divided by the number of intracluster pairs, is always at least $1/2$. This does not hold for surprise. As an example, consider a graph that consists of six disjoint cycles of six vertices each; an optimum clustering with respect to surprise will cluster the graph according to its connected components, whereas a clustering that is optimal with respect to performance corresponds to a perfect matching of the edges.

Besides the different resolution scale, one of the main differences we observed between the objectives lies in how they handle vertices that are scarcely connected to the remainder of the graph. Assigning them to the clusters of their anchor vertices increases the number of intracluster pairs considerably, but has only a minor impact on the total sum of component squares i_{v^2}, as the respective vertices have low degree; hence, in contrast to the objectives in the second column of Table 9.1, objectives in the first column are prone to leave them unclustered. Depending on the application context, one might argue for any of the two solutions. On a more conceptual level of reasoning, considering the total sum of component squares is motivated by the use of the more realistic configuration null model, which takes the observed degree sequence into account, whereas considering the total number of intracluster pairs is more tightly connected to the intuitive intercluster sparsity vs. intracluster density paradigm. Hence, recommendations for any concrete method should take into account if outliers are desirable or not, and, if available, informations about the expected density or size of the clusters.

Outlook. Besides the more technical follow-up questions discussed in the individual chapters, I would like to point out some more general directions for future work. In his recent survey [83], Fortunato points out that "...it appears that the field has grown in a rather chaotic way, without a precise direction or guidelines." He argues that the most important task is "...defining a set of reliable benchmark graphs, against which algorithms should be tested".

I partly agree with this assessment, although in my opinion it is unlikely that there will be one clustering measure or method that turns out to be appropriate for all types of application. As discussed above, the question what is the best clustering of a graph may be answered ambiguously, and this goes beyond the self-evident issue of graphs exhibiting a hierarchical cluster structure. To get a better classification and overview of existing methods, it would be worthwhile to further theoretically and empirically analyze them to

identify similarities and differences that are helpful to know when choosing a particular method. As a concrete example, it might be interesting to know how and if clusterings produced by optimizing information theoretic measures for graph clustering [47, 188] differ from the two general classes of bicriteria measures from Table 9.1. Besides thinning out the jungle of fundamentally different approaches, interconnections between existing methods that have been derived with a different motivation in mind might strengthen their theoretical foundation. From a practical point of view, it would be nice to have some general guidelines to the question under which circumstances which method is the most promising, depending on some easily understandable user preferences. In the other direction, incorporating more feedback from concrete applications would be invaluable for the improvement of existing methods; similar to the field of graph drawing, conducting user studies might be conceivable.

Another more specific issue concerns the identification of significant resolution scales with respect to the multiresolution variants of modularity and surprise. Although we discussed some simple possibilities to automatically determine interesting parameter values in the context of surprise optimization (cf. Chapter 7), and there has been some work in this direction [186, 213], the question of what is the best way to do this and, especially, how this can be done efficiently, is still far from satisfyingly resolved.

Appendix A

Integer Linear Programs for Density-Constrained Graph Clustering

In this section we sketch how different instantiations of Density-Constrained Clustering can be cast into integer linear programs. As an in-depth treatment of this topic is beyond the scope of this thesis and some of the constructions entail prohibitively large sets of variables and constraints, we limit ourselves to the description of some common building blocks that can be used to model our objective functions and constraints and exemplify their usage by showing how to model aixd

A.1 Building Blocks for Density-Constrained Graph Clustering

Similar to the linear programs used in the context of surprise optimization in Section 6.4, the foundation of all our formulations is a set $\mathcal{X}$ of $\binom{n}{2}$ binary variables X_{uv}, which indicate whether vertices u and v share a cluster ($X_{uv} = 1$) or not ($X_{uv} = 0$). Since for a clustering, $\mathcal{X}$ constitutes an equivalence relation on V, $O(n^3)$ constraints suffice to ensure transitivity within $\mathcal{X}$. Based on $\mathcal{X}$, we can model many convenient values, e.g., whether an edge e is internal to $\mathcal{C}(v)$, the number of intercluster edges of $\mathcal{C}(v)$, $|\mathcal{C}(v)|$ or the number of clusters, $|\mathcal{C}|$. To increase readability, we defer some of the more technical details to Section A.3, but always indicate when we do so.

***Node-Node-Equivalence* Variables X_{uv}:** "is $C(u) = \mathcal{C}(v)$?" To render our base variables X_{uv} consistent, we use constraints ensuring their transitivity (Eq. A.12); reflexivity and symmetry are immediate, since X_{uv} and X_{vu} are the same variable and since we can simply set $X_{vv} = 1$ for all $v \in V$.

$$\mathcal{X} = \{X_{uv} \mid \{u, v\} \in \binom{V}{2}\} \text{ with } X_{uv} = \begin{cases} 1, & \text{if } C(u) = \mathcal{C}(v) \\ 0, & \text{otherwise} \end{cases} \tag{A.1}$$

***Internal-Edge-Indicator* Variables E_{ev}:** "is e inside $\mathcal{C}(v)$?" Using constraints for a binary AND (Eq. A.13), we align E_{ev} with $\mathcal{X}$ to behave as follows:

$$\mathcal{E} = \{E_{ev} \mid e \in E, v \in V\} \text{ with } E_{\{i,j\}v} = X_{iv} \text{ AND } X_{jv} \tag{A.2}$$

***External-Edge-Indicator* Variables L_{ev}:** "is e an intercluster edge of $\mathcal{C}(v)$?" Determining e's end-vertices $\{i, j\}$, we align L_{ev} with $\mathcal{X}$ by constraints for binary XOR using X_{iv} and X_{jv} (Eq. A.14), which ensures the following behavior:

$$\mathcal{L} = \{L_{ev} \mid e = \{i,j\} \in E, v \in V\} \text{ with } L_{\{i,j\}v} = X_{iv} \text{ XOR } X_{jv} \tag{A.3}$$

***Cluster-Size* Variables $Z_{v\ell}$:** "is $|\mathcal{C}(v)| = \ell$?" For each $v \in V$ we can write $|\mathcal{C}(v)| := \sum_{u \in V} X_{uv}$, and then use constraints for value matching (see Eq. A.15) to align $Z_{v\ell}$ with $\mathcal{X}$ such that the following holds:

$$\mathcal{Z}(V) = \{Z_{v\ell} \mid v \in V, \ell \in \{1, \ldots, n\}\} \text{ with } Z_{v\ell} = \begin{cases} 1, & |\mathcal{C}(v)| = \ell \\ 0. & \text{otherwise} \end{cases} \tag{A.4}$$

***Über-Node* Variables $\ddot{U}_v$:** "is v über-vertex of $\mathcal{C}(v)$?" Let us assume that the vertices are ordered, i.e., we identify each vertex with a unique index between 1 and n. We call vertex $v = \min_{u \in C(w)}\{u\}$ the *über-vertex* of $C(w)$, i.e., the smallest index in a cluster.

$$\ddot{\mathcal{U}}(V) = \{\ddot{U}_v \mid v \in V\} \text{ with } \ddot{U}_v = \begin{cases} 1, & v = \min_{u \in \mathcal{C}(v)}\{u\} \\ 0, & \text{otherwise} \end{cases} \tag{A.5}$$

$$\forall v \in V : \ddot{U}_v \leq 1 - \frac{1}{n}(\sum_{u<v} X_{uv}) \text{ and } \ddot{U}_v \geq 1 - \sum_{u<v} X_{uv} \tag{A.6}$$

Note that from $\mathcal{E}$, $\mathcal{L}$ and $\ddot{\mathcal{U}}$, we can obtain the values $e_{\mathcal{C}(v)}$, $x_{\mathcal{C}(v)}$, $v_{\mathcal{C}(v)}$ and $|\mathcal{C}|$, respectively, and even $e_{\mathcal{C}(v),\mathcal{C}(u)}$. Building upon the variables above, we can express all measures discussed in this work, however, with no claim about minimal complexity. The following example models the constraint $\mathsf{mid} \geq \alpha$:

$$\forall v \in V : \sum_{e \in E} E_{ev} - \alpha \cdot (\sum_{\ell \in \{1,\ldots,n\}} \binom{\ell}{2} \cdot Z_{v\ell}) \geq 0 \tag{A.7}$$

***$\mathcal{C}(v)$-Indicator* Variables:** "$J_{v,P}$ = does $\mathcal{C}(v)$ meet properties P?" We use these variables as brute-force building blocks for substituting divisions, which we require but cannot formulate linearly. In the following we show how to define appropriate variables $J_{v,P}$ for aixd. For aixd we require P to contain the variables necessary to precompute $\mathsf{ixd}(\mathcal{C}(v))$, which are $x_{C(v)}$ and $n_{C(v)}$. Thus we build a variable for each candidate combination of vertex v, $x_{\mathcal{C}(v)}$ and $n_{\mathcal{C}(v)}$:

$$\mathcal{J}^{\mathsf{aixd}}(V) = \{J_{v,\tilde{x},\tilde{n}} \mid v \in V, \tilde{x} \in \{0, \ldots, m\}, \tilde{n} \in \{1, \ldots, n\}\} \tag{A.8}$$

$$\text{with } J_{v,\tilde{x},\tilde{n}} = \begin{cases} 1, & \text{if } x_{\mathcal{C}(v)} = \tilde{x} \text{ and } n_{\mathcal{C}(v)} = \tilde{n} \quad (\text{i.e., } \mathcal{C}(v) \text{ meets } P) \\ 0, & \text{otherwise} \end{cases} \tag{A.9}$$

But these two properties can easily be enforced as follows, using binary constraints for value matching, as shown in Eq. A.15 for general settings, and variables L_{ev} and X_{uv} as defined above:

$$\begin{aligned}
&0 \text{ for wrong } \tilde{x}\text{: } J_{v,\tilde{x},\tilde{n}} \leq 1 - \frac{\tilde{x} - \sum_{e \in E} L_{ev}}{m+1}, \quad J_{v,\tilde{x},\tilde{n}} \leq 1 - \frac{\sum_{e \in E} L_{ev} - \tilde{x}}{m+1}, \\
&0 \text{ for wrong } \tilde{n}\text{: } J_{v,\tilde{x},\tilde{n}} \leq 1 - \frac{\tilde{n} - \sum_{u \in V} X_{uv}}{n}, \quad J_{v,\tilde{x},\tilde{n}} \leq 1 - \frac{\sum_{u \in V} X_{uv} - \tilde{n}}{n}, \\
&\text{force the one fitting variable to 1: } \forall v \in V : \sum_{\tilde{x}=0}^{m} \sum_{\tilde{n}=1}^{n} J_{v,\tilde{x},\tilde{n}} = 1
\end{aligned} \tag{A.10}$$

With the set of all variables $J_{v,\tilde{x},\tilde{n}}$ at our disposal, we can now compute the value $\mathsf{ixd}(\mathcal{C}(v)) = \frac{x_{\mathcal{C}(v)}}{n_{\mathcal{C}(v)} n_{V \setminus \mathcal{C}(v)}}$ as follows:

$$\mathsf{ixd}(\mathcal{C}(v)) = \sum_{\tilde{x}=0}^{m} \sum_{\tilde{n}=1}^{n} \left(J_{v,\tilde{x},\tilde{n}} \cdot \frac{\tilde{x}}{\tilde{n}(n-\tilde{n})} \right) \tag{A.11}$$

A.2 Example Usage and Discussion

A general obstacle for modeling our measures linearly are the division operations necessary for the computation of almost all of them. We can substitute such operations in a brute-force manner: We multiply binary *indicator variables* $J_{v,P}$, which equal 1 if and only if $\mathcal{C}(v)$ meets properties P, with precomputed intermediate results for P, and take the sum over all possible configurations of P.

We exemplify the usage of $J_{v,P}$ with the measure aixd. To calculate $\mathsf{ixd}(\mathcal{C}(v))$, we need P to contain $n_{\mathcal{C}(v)}$ and $x_{\mathcal{C}(v)}$. Thus, we add constraints forcing $J_{v,\tilde{x},\tilde{n}}$ to equal 1 if and only if $\tilde{n} = n_{\mathcal{C}(v)}$ and $\tilde{x} = x_{\mathcal{C}(v)}$. We can then use these indicators as follows: Let

$$A_v := \sum_{\tilde{x}=0}^{m} \sum_{\tilde{n}=1}^{n} J_{v,\tilde{x},\tilde{n}} \cdot \frac{\tilde{x}}{\tilde{n}(n-\tilde{n}) \cdot \tilde{n}}.$$

Then, $\mathsf{ixd}(\mathcal{C}(v))$ is given by $\sum_{w \in \mathcal{C}(v)} A_w$ and thus $\mathsf{aixd}'(\mathcal{C}) := \sum_{v \in V} A_v = |\mathcal{C}| \cdot \mathsf{aixd}(\mathcal{C})$. We can circumvent a division by $|\mathcal{C}|$ by more subtle means, avoiding an asymptotic increase in complexity: For all possible values $\tilde{k}$ which $|\mathcal{C}|$ can assume, we add constraints $\mathsf{aixd}(\mathcal{C}) \geq \frac{1}{\tilde{k}} \cdot \mathsf{aixd}'(\mathcal{C}) - M_{\tilde{k}}$, with $M_{\tilde{k}}$ being large if $\tilde{k} \neq |\mathcal{C}|$. When minimizing $\mathsf{aixd}(\mathcal{C})$, only the one inequality using the correct $\tilde{k}$ actually constrains, thus this yields the correct result.

In summary, we end up with $O(n^2 m)$ variables, where the number of indicator variables $J_{v,\tilde{x},\tilde{n}}$ dominates the term. Beyond the $O(n^3)$ constraints used for keeping the base variables $\mathcal{X}$ consistent, and the lesser number of constraints for helper variables, every single indicator variable requires a constant number (roughly 4 for aixd) of constraints, which yields $O(n^2 m)$ constraints.

Generally, we can use this method to construct ILPs for all our measures, however, some measures require prohibitively large sets P of properties. For instance, conductance requires additional properties representing the case distinction in the denominator and the volume which roughly incurs a factor $2m$ in the number of variables (and constraints);

all pairwise measures require indicators for *pairs* of vertices, adding another factor of n^2 or even nm. Used as constraints (not as objective functions), the intracluster measures gid and mid are simpler to model, while aid again seems to require a construction as above.

A.3 Modelling Equivalence Relations and Boolean Functions

This section reviews some standard techniques that can be used to model equivalence relations and boolean functions with the help of linear constraints. These techniques can be used to fill some of the details left out in Section A.1, as indicated there. All variables occurring in this section are binary, the value 1 indicating that the statement associated with the variable is true.

***Transitivity* Constraints:** "$A_{ab} \wedge A_{bc} \Rightarrow A_{ac}$" Given a set of objects Set and binary variables A_{ab} corresponding to each pair of objects $a, b \in \text{Set}$, indicating a relation on Set $\times$ Set, the following constraints ensure that this relation is transitive:

$$\forall \{a,b,c\} \in \binom{\text{Set}}{3} : \begin{cases} A_{ab} + A_{bc} - A_{ac} \leq 1 \\ A_{ab} + A_{ac} - A_{bc} \leq 1 \\ A_{ac} + A_{bc} - A_{ab} \leq 1 \end{cases} \tag{A.12}$$

Binary* And *Constraints: "$T = \bigwedge_{i=1}^{t} A_i$"

$$T \leq \frac{1}{t} \sum_{i=1}^{t} A_i \quad \text{and} \quad T \geq \sum_{i=1}^{t} A_i - (t-1) \tag{A.13}$$

Binary* Xor *Constraints: "$T = A_a$ Xor A_b"

$$T \leq 2 - A_a - A_b \qquad T \leq A_a + A_b \qquad T \geq A_a - A_b \qquad T \geq -A_a + A_b \tag{A.14}$$

***Binary Value-Match* Constraints:** "$T_{t\ell} = \begin{cases} 1, & t = \ell \\ 0, & \text{otherwise} \end{cases}$"
We set t to be the target value, then ℓ searches t's range and tries to match it. An example showing how these constraints can be used for our purposes is given around Eq. A.10.

$$T_{t\ell} \leq 1 - \frac{1}{|\text{range}|}(\ell - t), \quad T_{t\ell} \leq 1 - \frac{1}{|\text{range}|}(t - \ell), \quad \sum_{\ell \in \text{range}} T_{t\ell} = 1 \tag{A.15}$$

Appendix B

Additional Experiments from Chapter 5

B.1 Gini Coefficients GVM and GM: Additional Plots

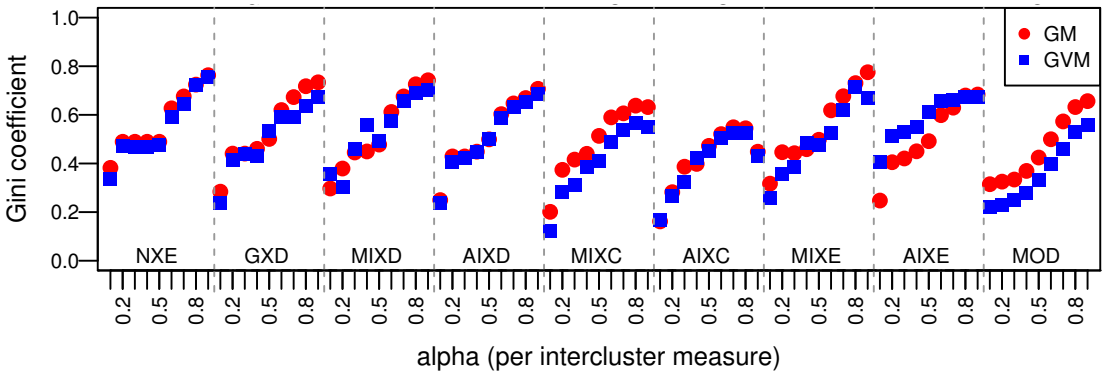


FIGURE B.1: Gini coefficient, clusterings produced by using GVM and GM with constraints on aid.

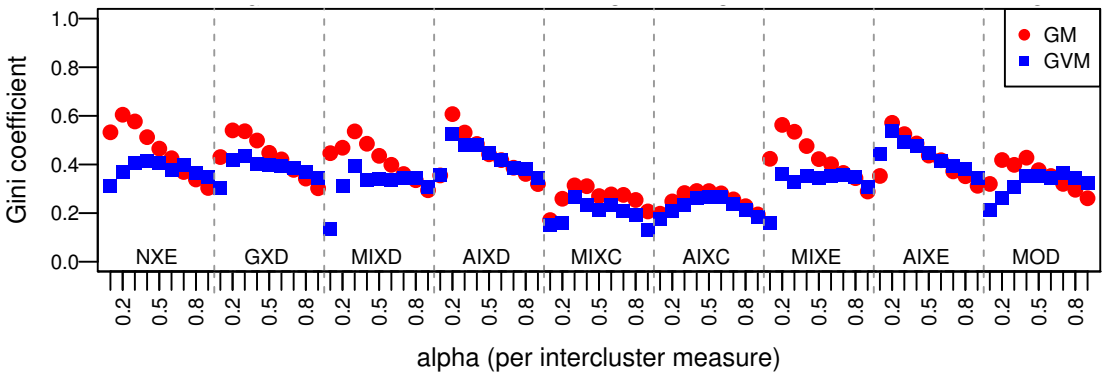


FIGURE B.2: Gini coefficient, clusterings produced by using GVM and GM with constraints on mid.

B.2 Effectiveness of Different Objective Functions: Additional Plots

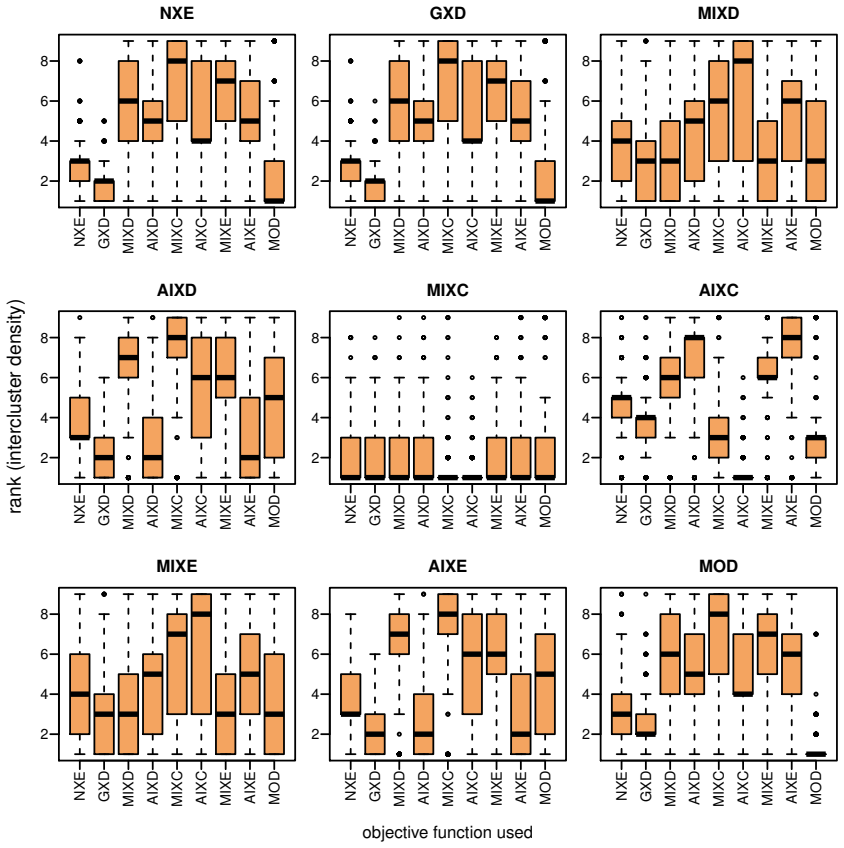


FIGURE B.3: Effectiveness of different objective functions, one plot per intercluster measure, i.e., evaluated measure (see supertitles). In each plot, ranks for different intercluster density measures as objectives in the GVM-algorithm (indicated by the label of the x axis) using mid as constraint are shown.

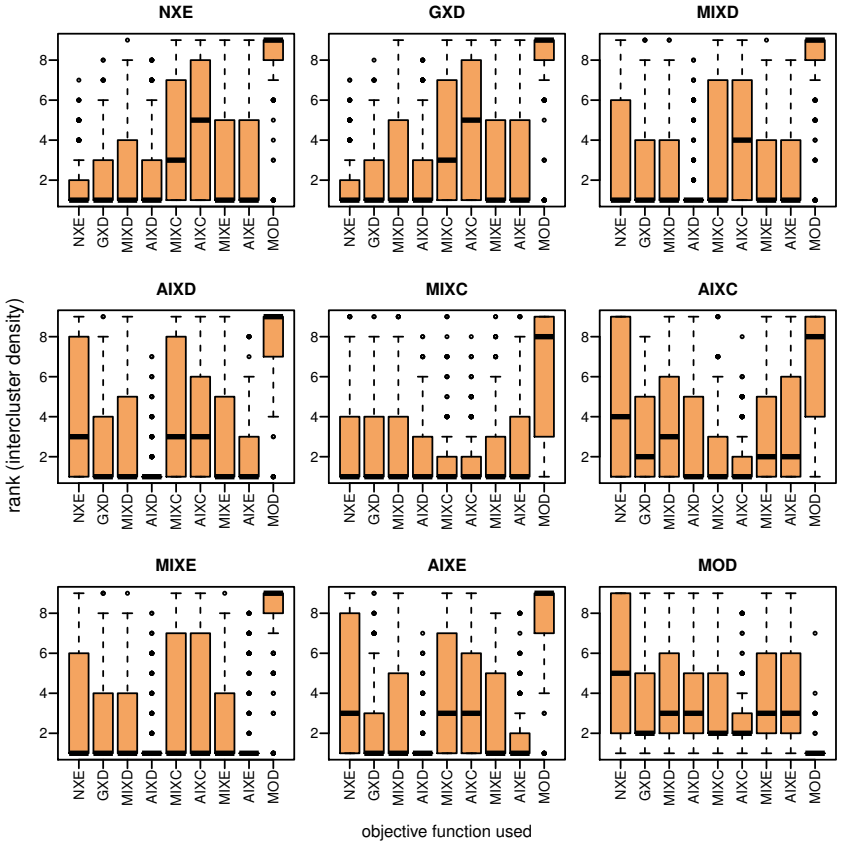
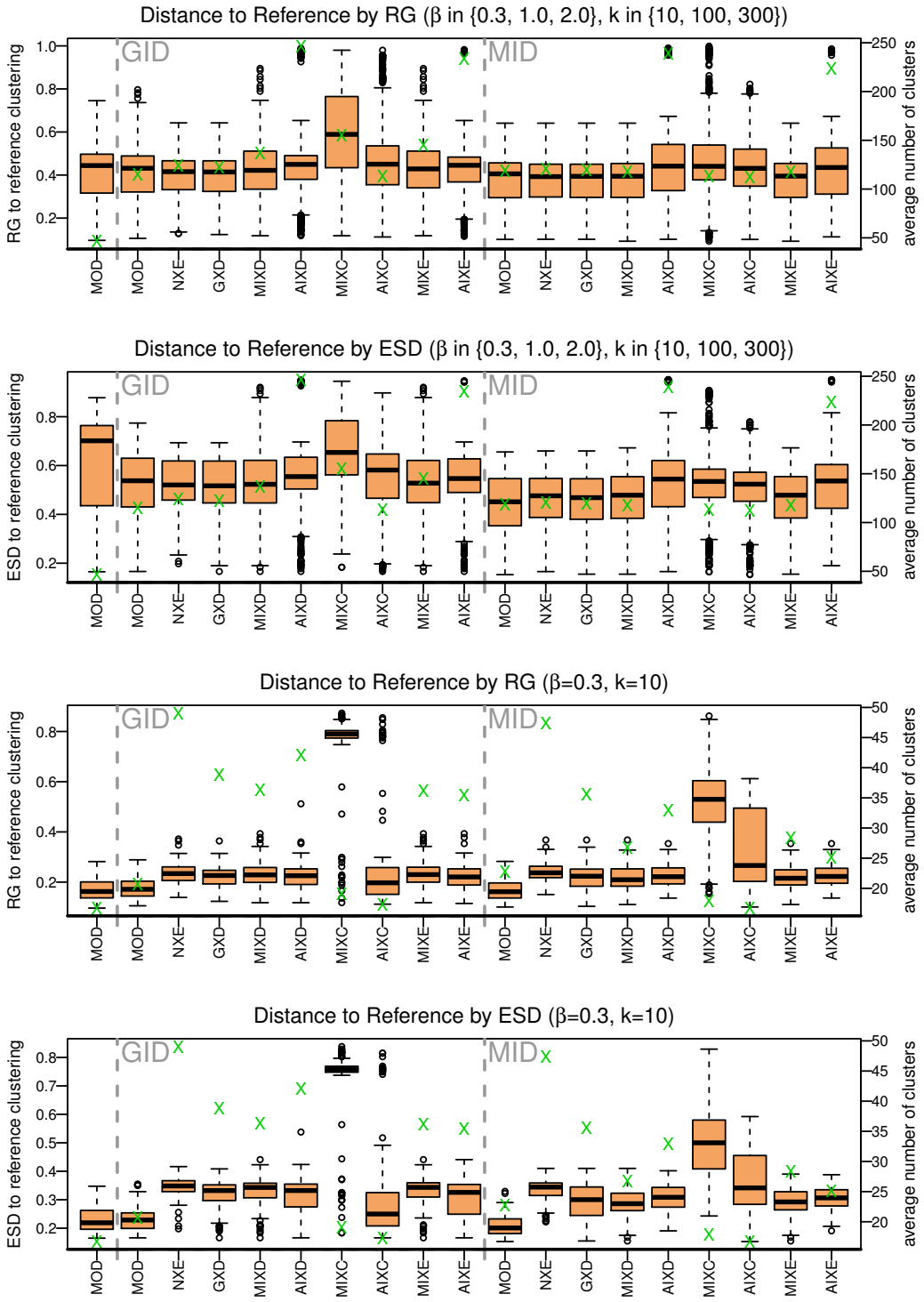
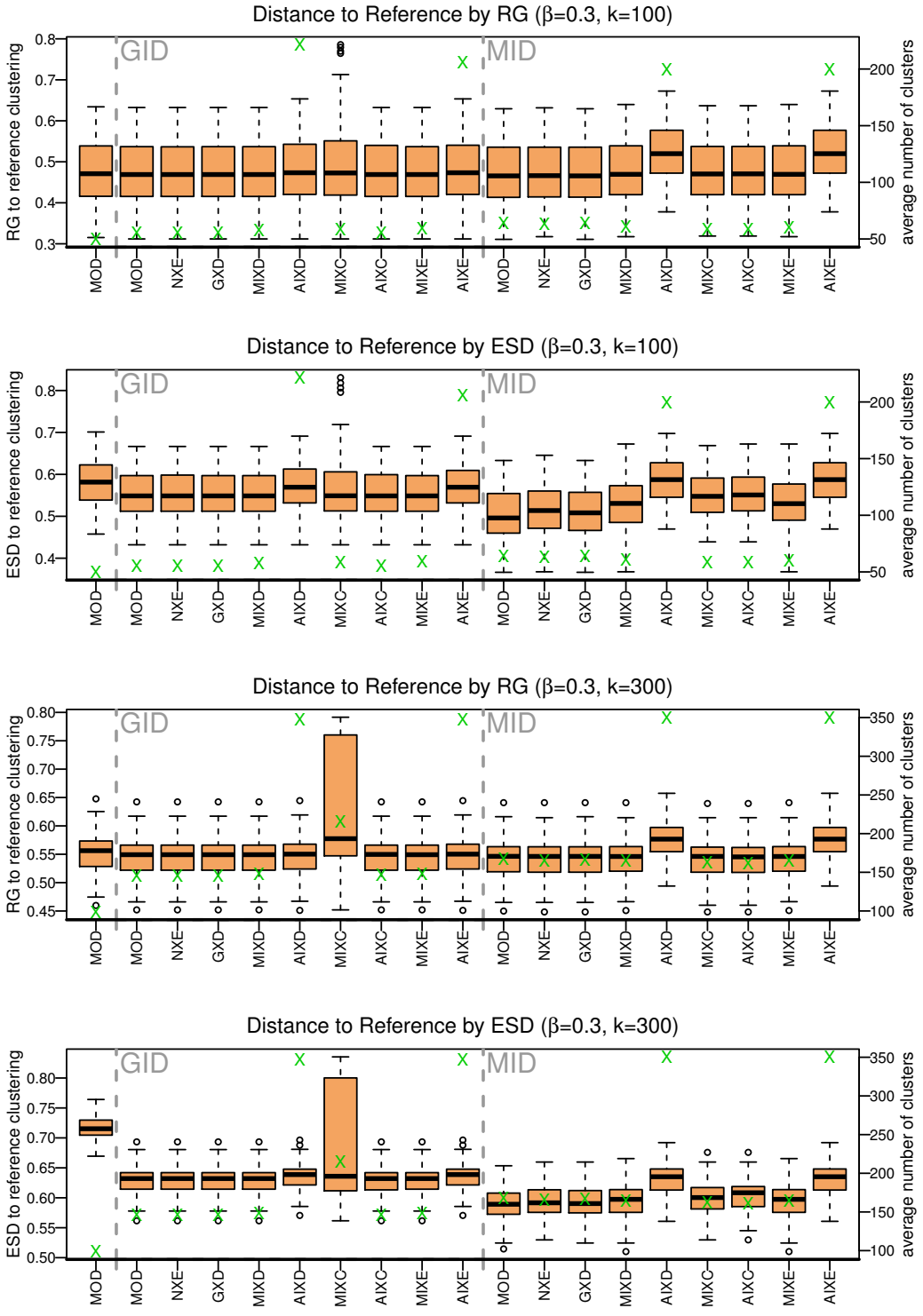
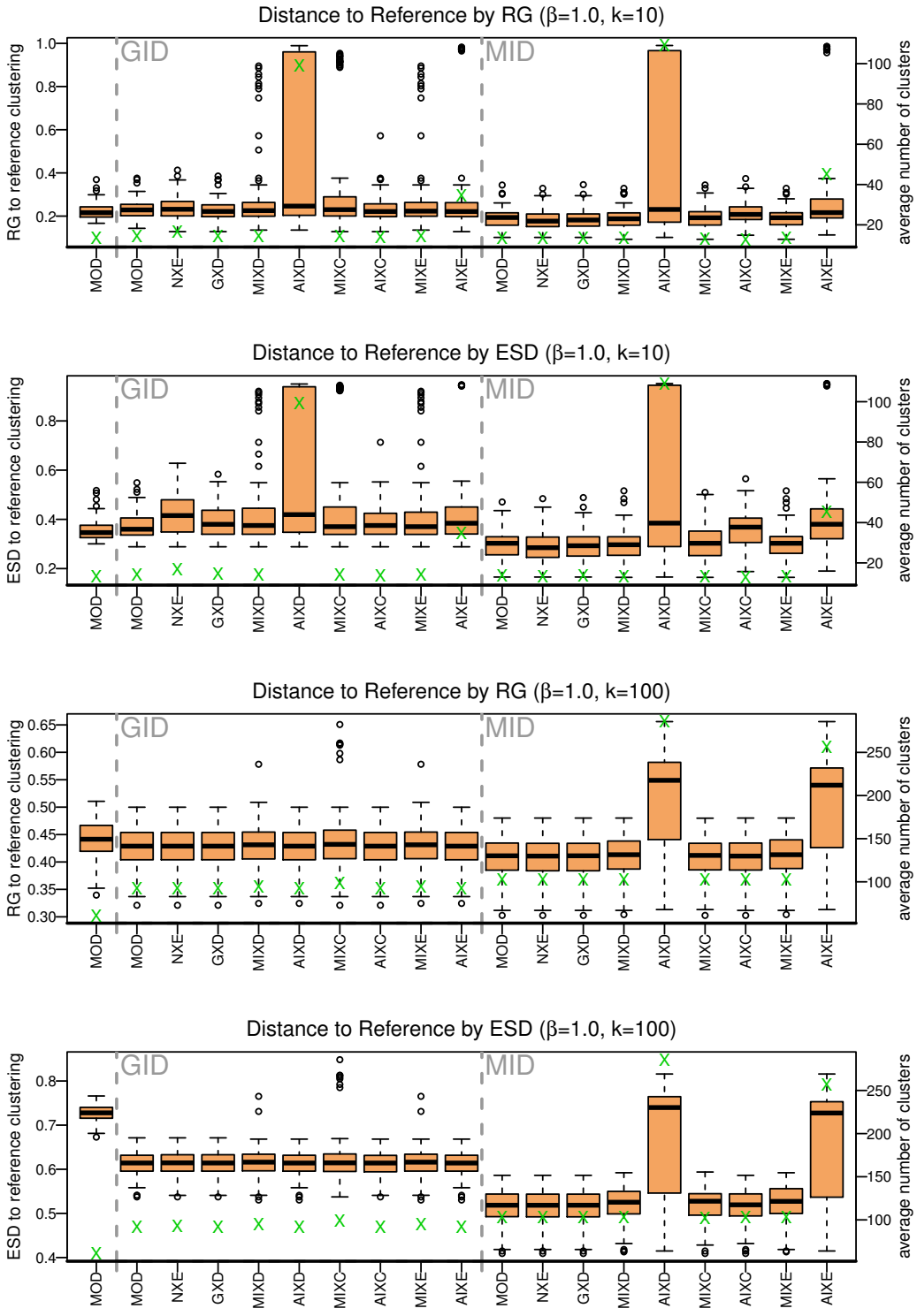


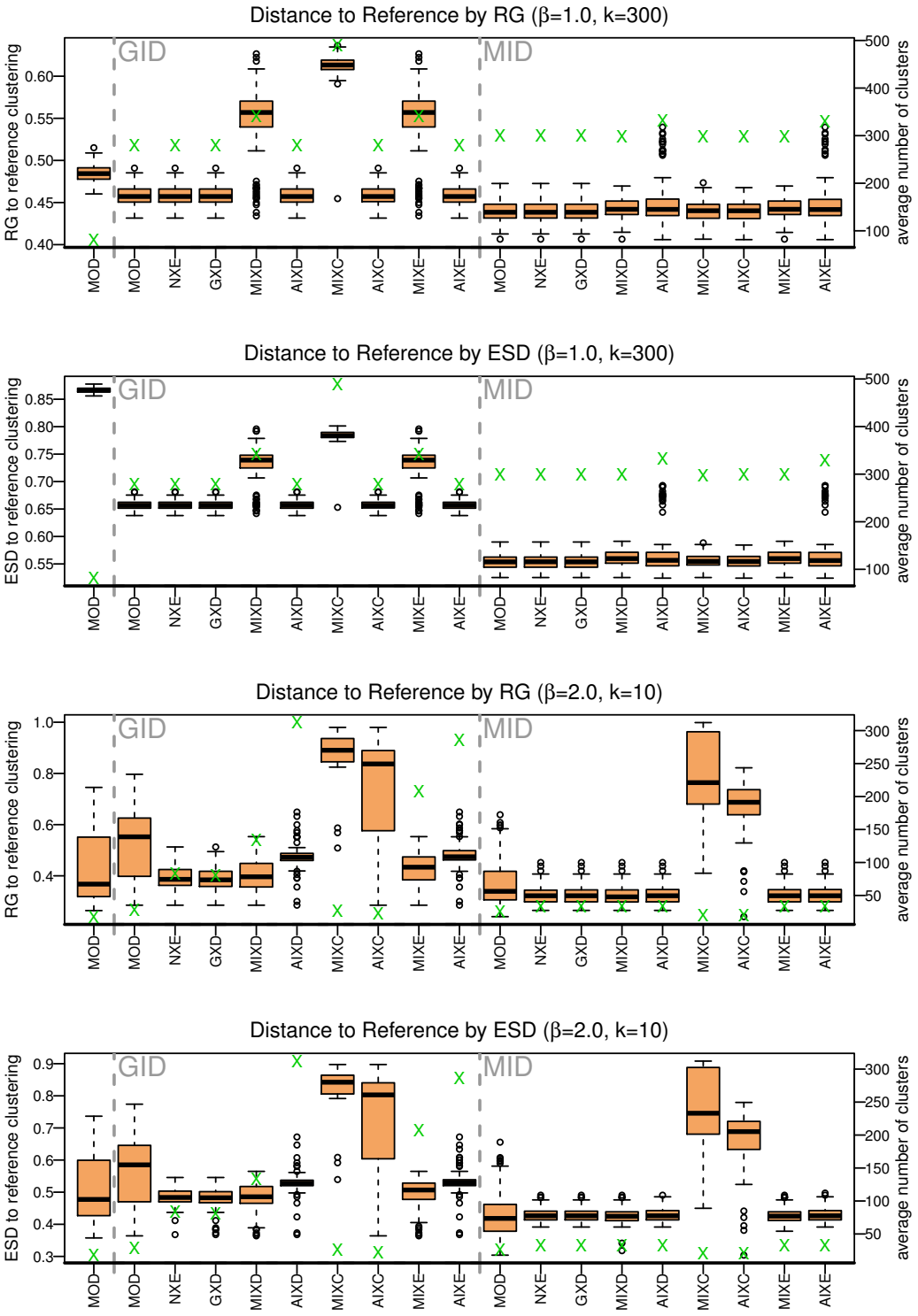
FIGURE B.4: Effectiveness of different objective functions, one plot per intercluster measure, i.e., evaluated measure (see supertitles). In each plot, ranks for different intercluster density measures as objectives in the GVM-algorithm (indicated by the label of the x axis) using aid as constraint are shown.

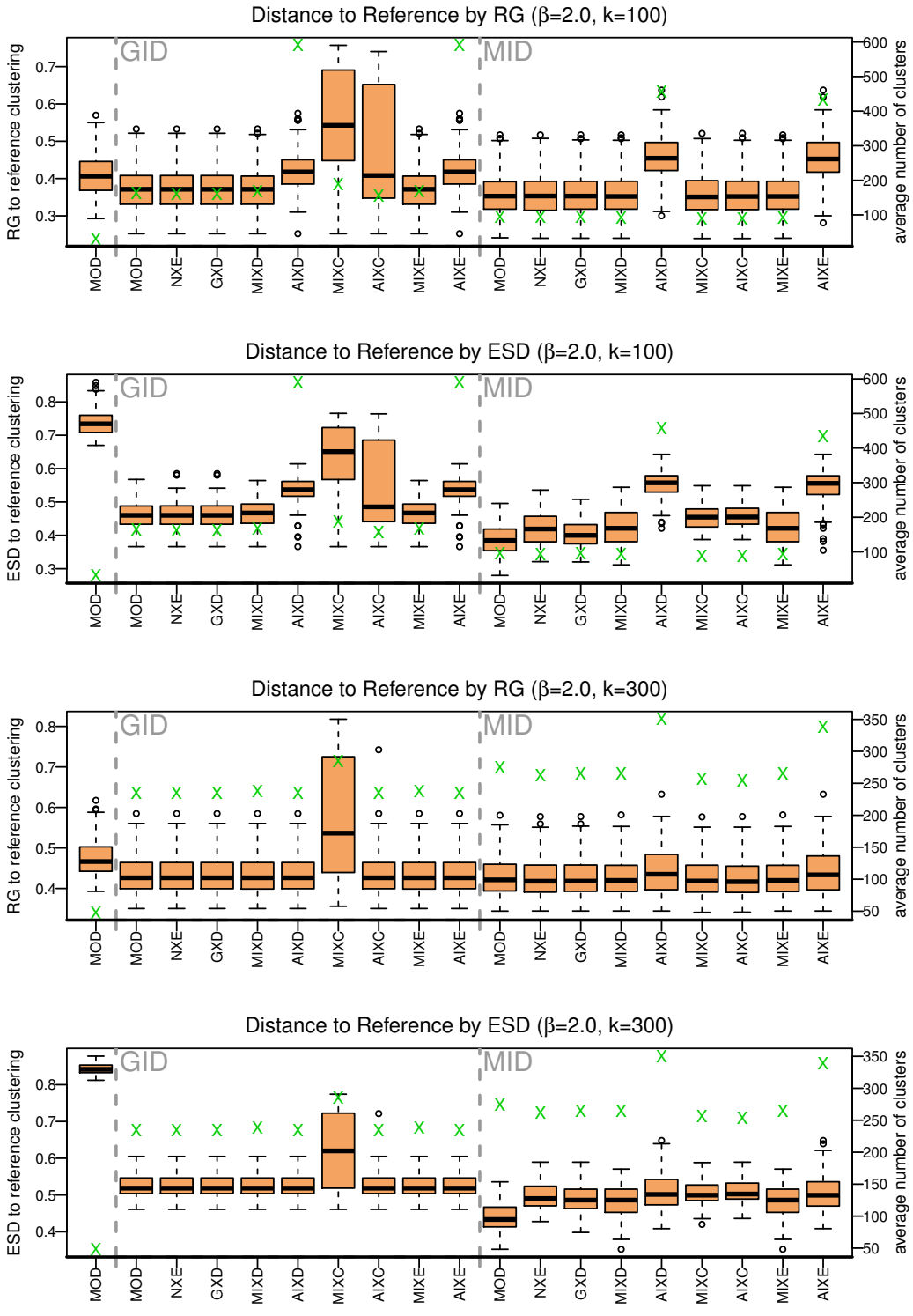
B.3 Complete Experiments with Planted Partition Graphs

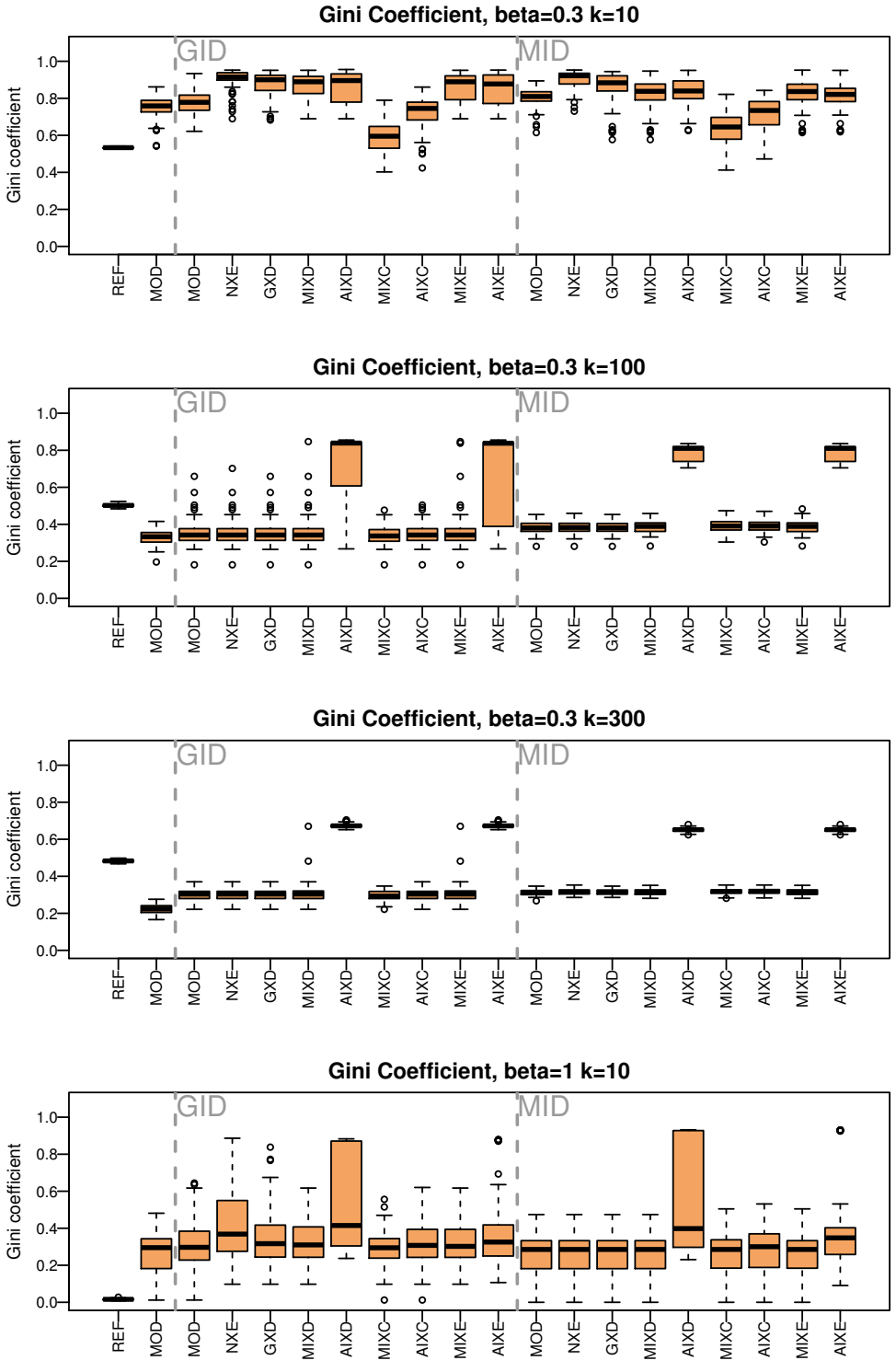


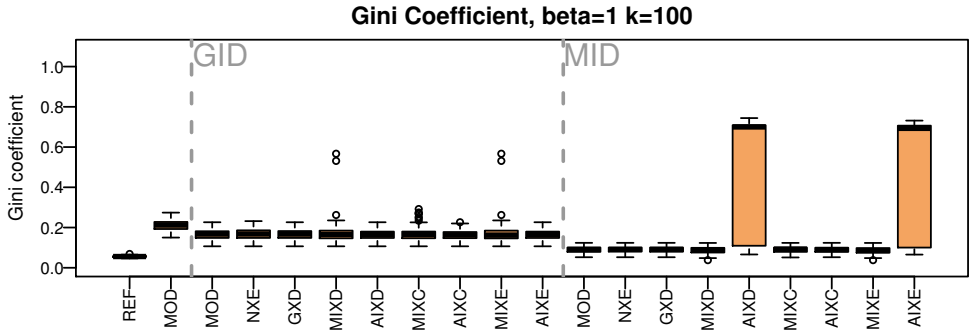

Distance to Reference by RG (β=0.3, k=100)
RG to reference clustering
average number of clusters
GID
MID
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE
Distance to Reference by ESD (β=0.3, k=100)
ESD to reference clustering
Distance to Reference by RG (β=0.3, k=300)
Distance to Reference by ESD (β=0.3, k=300)

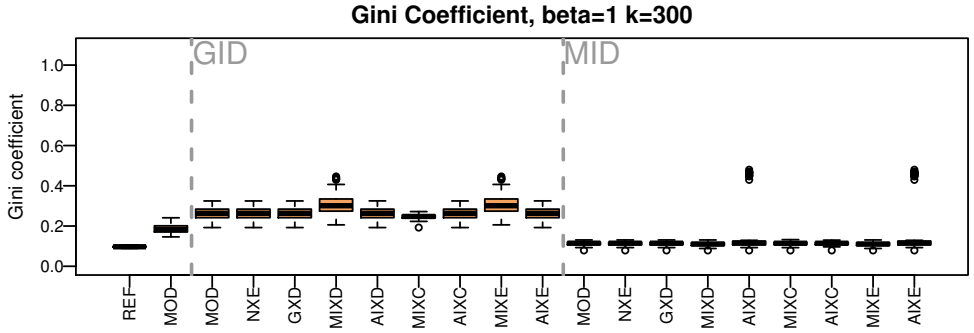

Distance to Reference by RG (β=1.0, k=10)
RG to reference clustering
average number of clusters
GID
MID
Distance to Reference by ESD (β=1.0, k=10)
ESD to reference clustering
average number of clusters
GID
MID
Distance to Reference by RG (β=1.0, k=100)
RG to reference clustering
average number of clusters
GID
MID
Distance to Reference by ESD (β=1.0, k=100)
ESD to reference clustering
average number of clusters
GID
MID
MOD
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE

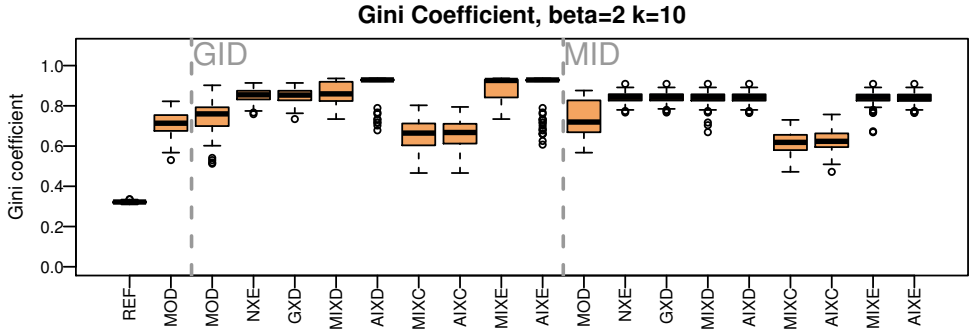

Distance to Reference by RG (β=1.0, k=300)
RG to reference clustering
average number of clusters
GID
MID
Distance to Reference by ESD (β=1.0, k=300)
ESD to reference clustering
average number of clusters
GID
MID
Distance to Reference by RG (β=2.0, k=10)
RG to reference clustering
average number of clusters
GID
MID
Distance to Reference by ESD (β=2.0, k=10)
ESD to reference clustering
average number of clusters
GID
MID
MOD
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE

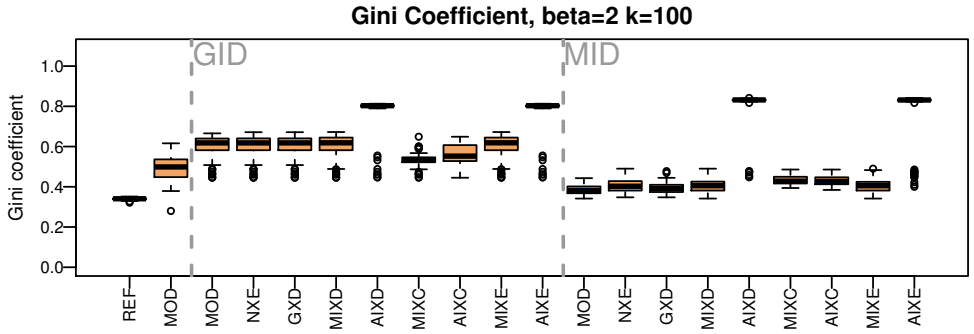

Distance to Reference by RG (β=2.0, k=100)
RG to reference clustering
average number of clusters
GID
MID
Distance to Reference by ESD (β=2.0, k=100)
ESD to reference clustering
average number of clusters
GID
MID
Distance to Reference by RG (β=2.0, k=300)
RG to reference clustering
average number of clusters
GID
MID
Distance to Reference by ESD (β=2.0, k=300)
ESD to reference clustering
average number of clusters
GID
MID
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE

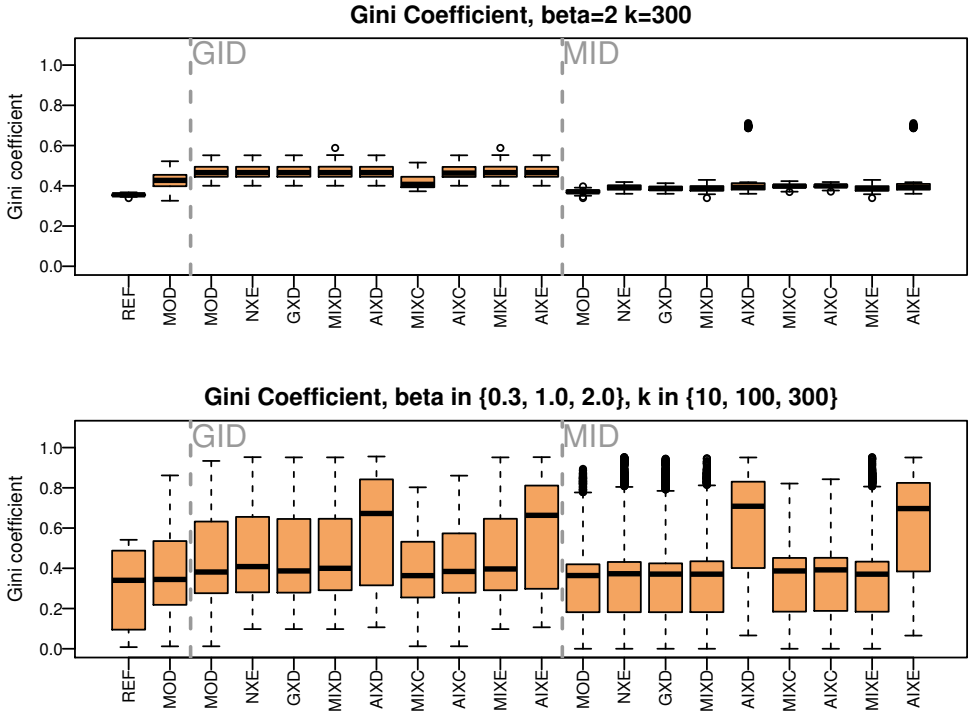

Gini Coefficient, beta=0.3 k=10
Gini Coefficient, beta=0.3 k=100
Gini Coefficient, beta=0.3 k=300
Gini Coefficient, beta=1 k=10
GID
MID
Gini coefficient
0.0
0.2
0.4
0.6
0.8
1.0
REF
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE


Gini Coefficient, beta=1 k=100
GID
MID
Gini coefficient
1.0
0.8
0.6
0.4
0.2
0.0
REF
MOD
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE


Gini Coefficient, beta=1 k=300
GID
MID
Gini coefficient
1.0
0.8
0.6
0.4
0.2
0.0
REF
MOD
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE


Gini Coefficient, beta=2 k=10
GID
MID
Gini coefficient
1.0
0.8
0.6
0.4
0.2
0.0
REF
MOD
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE


Gini Coefficient, beta=2 k=100
GID
MID
Gini coefficient
1.0
0.8
0.6
0.4
0.2
0.0
REF
MOD
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE


Gini Coefficient, beta=2 k=300
GID
MID
Gini coefficient
1.0
0.8
0.6
0.4
0.2
0.0
REF
MOD
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE
Gini Coefficient, beta in {0.3, 1.0, 2.0}, k in {10, 100, 300}
GID
MID
Gini coefficient
1.0
0.8
0.6
0.4
0.2
0.0
REF
MOD
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE
MOD
NXE
GXD
MIXD
AIXD
MIXC
AIXC
MIXE
AIXE

Bibliography

[1] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/. Last accessed 31 January 2015.

[2] James Abello, Mauricio Resende, and Sandra Sudarsky. Massive quasi-clique detection. In *LATIN 2002: Theoretical Informatics*, volume 2286 of *Lecture Notes in Computer Science*, pages 598–612. Springer, 2002.

[3] Richard D. Alba. A graph theoretic definition of a sociometric clique. *Journal of Mathematical Sociology*, 3:113–126, 1973.

[4] Rodrigo Aldecoa and Ignacio Marín. Jerarca: Efficient analysis of complex networks using hierarchical clustering. *PLoS ONE*, 5:e11585, July 2010.

[5] Rodrigo Aldecoa and Ignacio Marín. Deciphering network community structure by surprise. *PLoS ONE*, 6:e24195, September 2011.

[6] Rodrigo Aldecoa and Ignacio Marín. Closed benchmarks for network community structure characterization. *Physical Review E*, 85:026109, February 2012.

[7] Rodrigo Aldecoa and Ignacio Marín. Exploring the limits of community detection strategies in complex networks. *Nature Scientific Reports*, 3:2216, July 2013.

[8] Rodrigo Aldecoa and Ignacio Marín. Surprise maximization reveals the community structure of complex networks. *Nature Scientific Reports*, 3:1060, January 2013.

[9] Rodrigo Aldecoa and Ignacio Marín. SurpriseMe: an integrated tool for network community structure characterization using surprise maximization. *Bioinformatics*, 30(7):1041–1042, 2014.

[10] Daniel Aloise, Sonia Cafieri, Gilles Caporossi, Pierre Hansen, Sylvain Perron, and Leo Liberti. Column generation algorithms for exact modularity maximization in networks. *Physical Review E*, 82(4):046112, 2010.

[11] Michael R. Anderberg. *Cluster Analysis for Applications*, volume 19 of *Probability and Math.* Academic Press, 1973.

[12] Alex Arenas. Network data sets. http://deim.urv.cat/~aarenas/data/welcome.htm. Last accessed 31 January 2015.

[13] Alex Arenas, Leon Danon, Albert Díaz-Guilera, Pablo Gleiser, and Roger Guimerà. Community analysis in social networks. *The European Physical Journal B*, 38(2):373–380, 2004.

[14] Alex Arenas, Jordi Duch, Alberto Fernandez, and Sergio Gomez. Size reduction of complex networks preserving modularity. *New Journal of Physics*, 9(176), 2007.

[15] Vicente Arnau, Sergio Mars, and Ignacio Marín. Iterative cluster analysis of protein interaction data. *Bioinformatics*, 21(3):364–378, 2005.

[16] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing (STOC'04)*, pages 222–231. ACM Press, 2004.

[17] Mikhail Atallah, editor. *Algorithms and Theory of Computation Handbook*. CRC Press, 1999.

[18] James Bagrow. Evaluating local community methods in networks. *Journal of Statistical Mechanics: Theory and Experiment*, page P05001, 2008.

[19] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Journal of Machine Learning*, 56(1-3):89–113, July 2004.

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

[21] Srinka Basu, Debarka Sengupta, Ujjwal Maulik, and Sanghamitra Bandyopadhyay. A strong nash stability based approach to minimum quasi clique partitioning. In *Proceedings of the 6th International Conference on Communication Systems and Networks*, pages 1–6. IEEE, 2014.

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

[23] Hervé Baumann, Pierluigi Crescenzi, and Pierre Fraigniaud. Parsimonious flooding in dynamic graphs. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 260–269. ACM Press, 2009.

[24] Ronald I. Becker, Stephen R. Schach, and Yehoshua Perl. A shifting algorithm for min-max tree partitioning. *Journal of the ACM*, 29(1):58–67, 1982.

[25] Ehrhard Behrends. *Introduction to Markov Chains With Special Emphasis on Rapid Mixing*. Friedrick Vieweg & Son, October 2002.

[26] Jean-Paul Benzécri. Construction d'une classification ascendante hiérarchique par la recherche en chaîne des voisins réciproques. *Les cahiers de l'analyse des données*, 7(2):209–218, 1982.

[27] Tanya Berger-Wolf, David Kempe, and Chayant Tantipathananandth. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 717–726. ACM Press, 2007.

[28] Jonathan W. Berry, Bruce Hendrickson, Randall A. LaViolette, and Cynthia A. Phillips. Tolerating the community detection resolution limit with edge weighting. *Physical Review E*, 83:056119, May 2011.

[29] Andrea Bettinelli, Pierre Hansen, and Leo Liberti. Algorithm for parametric community detection in networks. *Physical Review E*, 86:016107, July 2012.

[30] Andrea Bettinelli, Pierre Hansen, and Leo Liberti. Community detection with the weighted parsimony criterion. *Journal of Systems Science and Complexity*, 2015. to appear.

[31] Charles-Edmond Bichot and Patrick Siarry, editors. *Graph Partitioning.* Wiley, 2011.

[32] 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.

[33] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, December 1996.

[34] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. Journal*, 51(3):255–269, May 2008.

[35] 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.

[36] Ulrik Brandes and Thomas Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *Lecture Notes in Computer Science*. Springer, February 2005.

[37] 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.

[38] 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.

[39] Ulrik Brandes and Martin Mader. A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In *Proceedings of the 19th International Symposium on Graph Drawing (GD'11)*, Lecture Notes in Computer Science, pages 99–110. Springer, 2012.

[40] Mauro Brunato, Holger Hoos, and Roberto Battiti. On effectively finding maximal quasi-cliques in graphs. In *Proceeding of the 2nd International Conference on Learning and Intelligent Optimization*, volume 5313 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2008.

[41] Dongbo Bu, Yi Zhao, Lun Cai, Hong Xue, Xiaopeng Zhu, Hongchao Lu, Jingfen Zhang, Shiwei Sun, Lunjiang Ling, Nan Zhang, Guojie Li, and Runsheng Chen. Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucleic Acids Research*, 31(9):2443–2450, 2003.

[42] Aidın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning, 2013. arXiv:1311.3144 [cs.DS].

[43] Ronald S. Burt. Positions in networks. *Social Forces*, 55:93–122, 1976.

[44] Sonia Cafieri, Pierre Hansen, and Leo Liberti. Loops and multiple edges in modularity maximization of networks. *Physical Review E*, 81(4):046102, 2010.

[45] Jean Cardinal and David Eppstein. Lazy algorithms for dynamic closest pair with arbitrary distance measures. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04)*, pages 112–119. SIAM, 2004.

[46] John J. Carrasco, Daniel C. Fain, Kevin J. Lang, and Leonid Zhukov. Clustering of bipartite advertiser-keyword graph. In *Workshop on Large Scale Clustering, ICDM 2003*, 2003.

[47] Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 112–124. ACM Press, 2004.

[48] Pak K. Chan, Martine D. Schlag, and Jason Y. Zien. Spectral k-way ratio-cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1088–1096, 1994.

[49] Frédéric Chataigner, Gordana Manic, Yoshiko Wakabayashi, and Raphael Yuster. Approximation algorithms and hardness results for the clique packing problem. *Electronic Notes in Discrete Mathematics*, 29:397–401, 2007.

[50] Jiyang Chen, Osmar R. Zaïane, and Randy Goebel. Detecting communities in large networks by iterative local expansion. In *Proceedings of the 2009 IEEE International Conference on Computational Aspects of Social Networks*, pages 105–112. IEEE Computer Society, 2009.

[51] David Cheng, Ravi Kannan, Santosh Vempala, and Grant Wang. A divide-and-merge methodology for clustering. *ACM Transactions on Database Systems*, 31(4):1499–1525, 2006.

[52] Aaron Clauset. Finding local community structure in networks. *Physical Review E*, 72(2):026132, August 2005.

[53] Aaron Clauset, Mark E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(066111), 2004.

[54] Andrea E. F. Clementi, Claudio Macci, Angelo Monti, Francesco Pasquale, and Riccardo Silvestri. Flooding time in edge-markovian dynamic graphs. *SIAM Journal on Discrete Mathematics*, 24(4):1694–1712, 2010.

[55] Brieuc Conan-Guez and Fabrice Rossi. Dissimilarity clustering by hierarchical multi-level refinement. In *Proceedings of the XXth European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2012)*, pages 483–488, 2012.

[56] Anne Condon and Richard M. Karp. Algorithms for graph partitioning on the planted partition model. *Randoms Structures and Algorithms*, 18(2):116–140, 2001.

[57] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing (STOC'71)*, pages 151–158. ACM Press, May 1971.

[58] 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.

[59] William H. Day and Herbert Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1(1):7–24, 1984.

[60] D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[61] 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.

[62] Imre Derényi, Gergely Palla, and Tamás Vicsek. Clique percolation in random networks. *Physical Review Letters*, 94:160202, 2005.

[63] Inderjit S. Dhillon, Juqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, November 2007.

[64] Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2010.

[65] Thang N. Dinh, Nam P. Nguyen, and My T. Thai. An adaptive approximation algorithm for community detection in dynamic scale-free networks. In *Proceedings of the 32th Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom)*, pages 55–59. IEEE Computer Society Press, 2013.

[66] Thang N. Dinh and My T. Thai. Towards optimal community detection: From trees to general weighted networks. *Internet Mathematics*. accepted pending revision.

[67] Luca Donetti and Miguel A. Muñoz. Detecting network communities: a new systematic and efficient algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 2004:P10012, October 2004.

[68] Haifeng Du, Marcus W. Feldman, Shuzhuo Li, and Xiaoyi Jin. An algorithm for detecting community structure of social networks based on prior knowledge and modularity. *Complexity*, 12(3):53–60, January 2007.

[69] Matthias Ehrgott. *Multicriteria Optimization*. Springer, 2005.

[70] David Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. *ACM Journal of Experimental Algorithmics*, 5:1–23, 2000.

[71] William H. Equitz. A new vector quantization clustering algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(10):1568–1575, October 1999.

[72] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd ACM SIGKDD international conference on Knowledge discovery and Data Mining*, pages 226–231. ACM Press, 1996.

[73] Tanja Falkowski, Anja Barth, and Myra Spiliopoulou. Dengraph: A density-based community detection algorithm. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 112–115. IEEE, 2007.

[74] C. T. Fan, Mervin E. Muller, and Ivan Rezucha. Development of sampling plans by using sequential (item by item) selection techniques and digital-computers. *Journal of the American Statistical Association*, 57(298):387–402, 1962.

[75] Ying Fan, Menghui Li, Peng Zhang, Jinshan Wu, and Zengru Di. Accuracy and precision of methods for community identification in weighted networks. *Physica A*, 377(1):363–372, 2007.

[76] Uriel Feige, Guy Kortsarz, and David Peleg. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

[77] Charles M. Fiduccia and Robert M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th ACM/IEEE Conference on Design Automation*, pages 175–181, 1982.

[78] Ronald A. Fisher and Frank Yates. *Statistical Tables for Biological, Agricultural and Medical Research*. Oliver and Boyd, London, 1948.

[79] Gary W. Flake, Robert E. Tarjan, and Kostas Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4):385–408, 2004.

[80] Tobias Fleck. Minimum-Contamination Probleme mit Bezug zu Graphclustern. Bachelor thesis, Karlsruhe Institute of Technology, 2013.

[81] Tobias Fleck, Andrea Kappes, and Dorothea Wagner. Graph clustering with surprise: Complexity and exact solutions. Technical Report arXiv:1310.60, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT), 2013.

[82] Tobias Fleck, Andrea Kappes, and Dorothea Wagner. Graph clustering with surprise: Complexity and exact solutions. In *Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'14)*, volume 8327 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2014.

[83] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3–5):75–174, 2010.

[84] 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.

[85] Pasi Fränti, Timo Kaukoranta, Day-Fann Shen, and Kuo-Shu Chang. Fast and memory efficient implementation of the exact PNN. *IEEE Transactions on Image Processing*, 9(5):773–777, May 2000.

[86] Pasi Fränti, Olli Virmajoki, and Ville Hautamäki. Fast agglomerative clustering using a k-nearest neighbor graph. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1875–1881, November 2006.

[87] 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.

[88] 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.

[89] Horst Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

[90] Corrado Gini. Measurement of inequality of incomes. *The Economic Journal*, 31(121):124–126, March 1921.

[91] Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Science of the United States of America*, 99(12):7821–7826, 2002.

[92] Philipp Glaser. Finding graph clusterings with varying coarseness. Studienarbeit, Karlsruhe Institute of Technology, 2013.

[93] 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.

[94] Robert Görke. *An Algorithmic Walk from Static to Dynamic Graph Clustering*. PhD thesis, Fakultät für Informatik, February 2010.

[95] Robert Görke, Marco Gaertler, Florian Hübner, and Dorothea Wagner. Computational aspects of lucidity-driven graph clustering. *Journal of Graph Algorithms and Applications*, 14(2):165–197, 2010.

[96] Robert Görke, Andrea Kappes, and Dorothea Wagner. Experiments on density-constrained graph clustering. *ACM Journal of Experimental Algorithmics*, 19:1.6:1.1–1.6:1.31, September 2014.

[97] Robert Görke, Roland Kluge, Andrea Schumm, Christian Staudt, and Dorothea Wagner. An efficient generator for clustered dynamic random networks. Technical report, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT), 2012. Karlsruhe Reports in Informatics 2012,17.

[98] Robert Görke, Roland Kluge, Andrea Schumm, Christian Staudt, and Dorothea Wagner. An efficient generator for clustered dynamic random networks. In *Proceedings of the 1st Mediterranean Conference on Algorithms*, pages 219–233. Springer, 2012.

[99] Robert Görke, Pascal Maillard, Andrea Schumm, Christian Staudt, and Dorothea Wagner. Dynamic graph clustering combining modularity and smoothness. *ACM Journal of Experimental Algorithmics*, 18(1):1.5:1.1–1.5:1.29, April 2013.

[100] Robert Görke, Pascal Maillard, Christian Staudt, and Dorothea Wagner. Modularity-driven clustering of dynamic graphs. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 436–448. Springer, May 2010.

[101] Robert Görke, Andrea Schumm, and Dorothea Wagner. Density-constrained graph clustering. In *Algorithms and Data Structures, 12th International Symposium (WADS'11)*, volume 6844 of *Lecture Notes in Computer Science*, pages 679–690. Springer, 2011.

[102] Robert Görke, Andrea Schumm, and Dorothea Wagner. Density-constrained graph clustering. Technical report, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT), 2011. Karlsruhe Reports in Informatics 2011-17.

[103] Robert Görke, Andrea Schumm, and Dorothea Wagner. Experiments on density-constrained graph clustering. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 1–15. SIAM, 2012.

[104] 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.

[105] J. C. Gower and G. J. Ross. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 18(1):54–64, 1969.

[106] Derek Greene, Dónal Doyle, and Pádraig Cunningham. Tracking the evolution of communities in dynamic social networks. In *Proceedings of the 2010 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 176–183. IEEE Computer Society, 2010.

[107] Ilan Gronau and Shlomo Moran. Optimal implementations of UPGMA and other common clustering algorithms. *Information Processing Letters*, 104:205–210, 2007.

[108] Martin Grötschel and Yoshiko Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1-3):59–96, 1989.

[109] Martin Grötschel and Yoshiko Wakabayashi. Facets of the clique partitioning polytope. *Mathematical Programming*, 47(1-3):367–387, 1990.

[110] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. *ACM SIGMOD Record*, 27(2):73–84, 1998.

[111] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. In *Proceedings of the 15th International Conference on Data Engineering*, page 512. IEEE Computer Society, 1999.

[112] Roger Guimerà and Luís A. Nunes Amaral. Functional cartography of complex metabolic networks. *Nature*, 433:895–900, February 2005.

[113] Roger Guimerà, Marta Sales-Pardo, and Luís A. Nunes Amaral. Module identification in bipartite and directed networks. *Physical Review E*, 76:036102, September 2007.

[114] Inc. Gurobi Optimization. Gurobi optimizer reference manual. http://www.gurobi.com. Last accessed 31 January 2015.

[115] Lars Hagen and Andrew B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085, September 1992.

[116] David Harel and Yehuda Koren. Clustering spatial data using random walks. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 281–286. ACM Press, 2001.

[117] Tanja Hartmann, Andrea Kappes, and Dorothea Wagner. Clustering evolving networks. Technical Report arXiv:1401.3516, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT), 2014.

[118] Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4-6):175–181, 2000.

[119] Klaus Holzapfel, Sven Kosub, Moritz G. Maaß, and Hanjo Täubig. The complexity of detecting fixed-density clusters. In *Proceedings of the 5th Italian Conference on Algorithms and Complexity (CIAC'03)*, volume 2653 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2003.

[120] Holger Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier/Morgan Kaufmann, 2004.

[121] John E. Hopcroft, Omar Khan, Brian Kulis, and Bart Selman. Natural communities in large linked networks. In *Proceedings of the nineth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 541–546, 2003.

[122] Falk Hüffner, Christian Komusiewicz, Adrian Liebtrau, and Rolf Niedermeier. Partitioning biological networks into highly connected clusters with maximum edge coverage. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(3):455–467, 2014.

[123] Pan Hui, Jon Crowcroft, and Eiko Yoneki. Bubble rap: Social-based forwarding in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 10(11):1576–1589, November 2011.

[124] Takehiro Ito, Xiao Zhou, and Takao Nishizeki. Partitioning a graph of bounded tree-width to connected subgraphs of almost uniform size. *Journal of Discrete Algorithms*, 4(1):142–154, 2006.

[125] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[126] Ellis L. Johnson, Anuj Mehrotra, and Georg L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62(1-3):133–151, 1993.

[127] J. Juan. Programme de classification hiérarchique par l'algorithme de la recherche en chaîne des voisins réciproques. *Les cahiers de lanalyse des donnees*, 7(2):219–225, 1982.

[128] Björn H. Junker and Falk Schreiber, editors. *Analysis of Biological Networks*. Wiley Series on Bioinformatics. Wiley, April 2008.

[129] 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.

[130] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad, spectral. *Journal of the ACM*, 51(3):497–515, May 2004.

[131] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32(8):68–75, August 1999.

[132] George Karypis, Eui-Hong Han, and Vipin Kumar. Multilevel refinement for hierarchical clustering. Technical report, University of Minesota, 1999. UMN CS 99-020.

[133] Timo Kaukoranta, Pasi Fränti, and Olli Nevalainen. Vector quantization by lazy pairwise nearest neighbor method. *Optical Engineering*, 38(11):1862–1868, November 1999.

[134] Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, February 1970.

[135] Min-Soo Kim and Jiawei Han. A particle-and-density based evolutionary clustering method for dynamic networks. In *Proceedings of the 35th International Conference on Very Large Databases (VLDB 2009)*, pages 622–633, 2009.

[136] Andrew D. King, Natasa Pržulj, and Igor Jurisica. Protein complex prediction via cost-based clustering. *Bioinformatics*, 20(17):3013–3020, 2004.

[137] Ton Kloks. *Treewidth–Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[138] Roland Kluge. An efficient generator for large clustered dynamic random networks. bachelor thesis, Karlsruhe Institute of Technology, 2011.

[139] Hans-Peter Kriegel, Petra Mutzel, and Till Schäfer. SAHN clustering in arbitrary metric spaces using heuristic nearest neighbor search. In *Proceedings of the 3rd Workshop on Algorithms and Computation (WALCOM'09)*, volume 8344 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2014.

[140] Ravi Kumar, Jasmine Novak, and Andrew S. Tomkins. Structure and evolution of online social networks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 611–617. ACM Press, 2006.

[141] Jussi M. Kumpula, Jari Saramäki, Kimmo Kaski, and János Kertész. Limited resolution in complex network community detection with potts model approach. *The European Physical Journal B*, 56(1):41–45, March 2007.

[142] Sukhamay Kundu and Jayadev Misra. A linear tree partitioning algorithm. *SIAM Journal on Computing*, 6(1):151–154, 1977.

[143] Takio Kurita. An efficient agglomerative clustering algorithm using a heap. *Pattern Recognition*, 24(3):205–209, 1991.

[144] Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, 80(1):016118, 2009.

[145] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, October 2008.

[146] Kevin J. Lang and Satish Rao. A flow-based method for improving the expansion or conductance of graph cuts. In *Proceedings of the 10th Integer Programming and Combinatorial Optimization Conference*, volume 3064 of *Lecture Notes in Computer Science*, pages 325–337. Springer, 2004.

[147] Elizabeth A. Leicht and Mark E. J. Newman. Community structure in directed networks. *Physical Review Letters*, 100(11):118703+, March 2008.

[148] Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[149] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew S. Tomkins. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 462–470. ACM Press, 2008.

[150] 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.

[151] Angsheng Li and Linqing Tang. The complexity and approximability of minimum contamination problems. In *Proceedings of the 8th annual conference on Theory and applications of models of computation*, Lecture Notes in Computer Science, pages 298–307. Springer, 2011.

[152] Yu-Ru Lin, Yun Chi, Shenghuo Zhu, Hari Sundaram, and Belle L. Tseng. Analyzing communities and their evolutions in dynamic social networks. *ACM Transactions on Knowledge Discovery from Data*, 3(2):8:1–8:31, April 2009.

[153] Guimei Liu and Limsoon Wong. Effective pruning techniques for mining quasi-cliques. In *Proceedings of the 2008 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, volume 5212 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2008.

[154] Stuart Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[155] Anand Louis and Konstantin Makarychev. Approximation algorithm for sparsest k-partitioning. In *Proceedings of the 25th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 1244–1255. SIAM, 2014.

[156] R. Duncan Luce. Connectivity and generalized cliques in sociometric group structure. *Psychometrika*, 15:169–190, 1950.

[157] R. Duncan Luce and Albert Perry. A method of matrix analysis of group structure. *Psychometrika*, 14:95–116, 1949.

[158] Feng Luo, James Z. Wang, and Eric Promislow. Exploring local community structures in large networks. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 233–239. IEEE, 2006.

[159] 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.

[160] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 245–254, 2010.

[161] Claire P. Massen and Jonathan P. K. Doye. Identifying communities within energy landscapes. *Physical Review E*, 71(4):046101, 2005.

[162] H. Matsuda, T. Ishihara, and A. Hashimoto. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theoretical Computer Science*, 210:305–325, 1999.

[163] David W. Matula and Farhad Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics*, 27(1-2):113–123, May 1990.

[164] Robert J. Mokken. Cliques, clubs, and clans. *Quality and Quantity*, 13:161–173, 1979.

[165] Chris Muelder and Kwan-Liu Ma. Rapid graph layout using space filling curves. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1301–1308, 2008.

[166] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms, 2011. arXiv:1109.2378.

[167] Daniel Müllner. fastcluster: Fast hierarchical, agglomerative clustering routines for r and python. *Journal of Statistical Software*, 23(9):1–18, May 2013.

[168] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–259, 1983.

[169] F. Murtagh. *Multidimensional clustering algorithms*, volume 4 of *Compstat Lectures*. Physica-Verlag GmbH, 1985.

[170] Marc Newman. Network data. http://www-personal.umich.edu/~mejn/netdata/. Last accessed 31 January 2015.

[171] Mark E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[172] Mark E. J. Newman. Analysis of weighted networks. *Physical Review E*, 70(056131):1–9, 2004.

[173] Mark E. J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(026113):1–16, 2004.

[174] Vincenzo Nicosia, Giuseppe Mangioni, V. Carchiolo, and Michele Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(03):p03024 (23pp), 2009.

[175] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, 2005.

[176] Vilfredo Pareto. *Manual d'économie politique*. F. Rouge, 1896.

[177] Rob Patro, Geet Duggal, Emre Sefer, Hao Wang, Darya Filippova, and Carl Kingsford. The missing models: A data-driven approach for learning how networks grow. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 42–50. ACM Press, 2012.

[178] Jian Pei, Daxin Jiang, and Aidong Zhang. On mining cross-graph quasi-cliques. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 228–238. ACM Press, 2005.

[179] Yehoshua Perl and Stephen R. Schach. Max-min tree partitioning. *Journal of the ACM*, 28(1):5–15, 1981.

[180] R Development Core Team. R: A Language and Environment for Statistical Computing. http://www.r-project.org. Last accessed 31 January 2015.

[181] Jörg Reichardt and Stefan Bornholdt. Detecting fuzzy community structures in complex networks with a potts model. *Physical Review Letters*, 93(21):218701, November 2004.

[182] Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74(016110):1–16, 2006.

[183] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

[184] Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. An introduction to exponential random graph (p*) models for social networks. *Social Networks*, 29(2):173–191, 2007.

[185] F. James Rohlf. Hierarchical clustering using the minimum spanning tree. *ACM Computing Surveys*, 16:93–95, 1973.

[186] Peter Ronhovde and Zoran Nussinov. Multiresolution community detection for megascale networks by information-based replica correlations. *Physical Review E*, 80(1):016109, 2009.

[187] Peter Ronhovde and Zoran Nussinov. Local resolution-free potts model for community detection. *Physical Review E*, 81(4):046114, April 2010.

[188] Martin Rosvall, Daniel Axelsson, and Carl T. Bergstrom. The map equation. *The European Physical Journal Special Topics*, 178(1):13–23, 2009.

[189] Randolf Rotta and Andreas Noack. Multilevel local search algorithms for modularity clustering. *ACM Journal of Experimental Algorithmics*, 16:2.3:2.1–2.3:2.27, July 2011.

[190] Jianhua Ruan and Weixiong Zhang. Identifying network communities with a high resolution. *Physical Review E*, 77:016104, January 2008.

[191] Erin N. Sawardecker, Marta Sales-Pardo, and Luís A. Nunes Amaral. Detection of node group membership in networks with group overlap. *The European Physical Journal B*, 67:277–284, 2009.

[192] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, August 2007.

[193] Philipp Schuetz and Amedeo Caflisch. Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Physical Review E*, 77(046112), 2008.

[194] Stephen B. Seidman. Network structure and minimum degree. *Social Networks*, 5:269–287, 1983.

[195] Stephen B. Seidman and Brian L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6:139–154, 1978.

[196] 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.

[197] Jamshid Shanbehzadeh and Philip O. Ogunbona. On the computational complexity of the LBG and PNN algorithms. *IEEE Transactions on Image Processing*, 6(4):614–616, April 1997.

[198] Huawei Shen, Xueqi Cheng, Kai Cai, and Mao-Bin Hu. Detect overlapping and hierarchical community structure in networks. *Physica A: Statistical Mechanics and its Applications*, 388(8):1706–1712, 2009.

[199] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[200] Maria A. Sibaldo, Tiago B. A. de Carvalho, Tsang Ing Ren, and George Cavalcanti. Recommender system based on modularity. In *Proceedings of the 2014 Recommender Systems Challenge*, pages 58:58–58:61. ACM Press, 2014.

[201] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, January 1973.

[202] Jiří Šíma and Satu Elisa Schaeffer. On the NP-completeness of some graph cluster measures. In *Proceedings of the 32rd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'06)*, Lecture Notes in Computer Science, pages 530–537. Springer, 2006.

[203] Peter H. A. Sneath. The application of computers to taxonomy. *Microbiology*, 17(1):201–226, 1957.

[204] Peter H. A. Sneath and Robert R. Sokal. *Numerical Taxonomy: The Principles and Practice of Numerical Classification*. W. H. Freeman and Company, 1973.

[205] Tom A.B. Snijders. The statistical evaluation of social network dynamics. *Sociological Methodology*, 31(1):361–395, 2001.

[206] Robert R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *The University of Kansas Science Bulletin*, 38:1409–1438, January 1958.

[207] Daniel A. Spielman and Shang-Hau Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS'96)*, pages 96–106, October 1996.

[208] Aravind Srinivasan, Tanmoy Chakraborty, and Sanjukta Bhowmick. Identifying base clusters and their application to maximizing modularity. In *Graph Partitioning and Graph Clustering: Tenth DIMACS Implementation Challenge*, volume 588 of *DIMACS Book*, pages 141–156. American Mathematical Society, 2013.

[209] Christian Staudt and Henning Meyerhenke. Engineering high-performance community detection heuristics for massive graphs. In *Proceedings of the 2013 International Conference on Parallel Processing*. Conference Publishing Services (CPS), 2013.

[210] Yizhou Sun, Jie Tang, Jiawei Han, Manish Gupta, and Bo Zhao. Community evolution detection in dynamic heterogeneous information networks. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, pages 137–146. ACM Press, 2010.

[211] Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, August 1997.

[212] V. A. Traag, P. Van Dooren, and Y. Nesterov. Narrow scope for resolution-limit-free community detection. *Physical Review E*, 84(1):016114, July 2011.

[213] V. A. Traag, G. Krings, and P. Van Dooren. Significant scales in community structure. *Nature Scientific Reports*, 3(2930), 2013.

[214] Takeaki Uno. An efficient algorithm for enumerating pseudo cliques. In *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC'07)*, volume 4835 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 2007.

[215] Stijn M. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[216] Stijn M. van Dongen. Graph clustering via a discrete uncoupling process. *SIAM Journal on Matrix Analysis and Applications*, 30(1):121–141, 2008.

[217] Alexei Vázquez. Growing network with local rules: Preferential attachment, clustering hierarchy, and degree correlations. *Physical Review E*, 67:056104, May 2003.

[218] Thomas Voice, Maria Polukarov, and Nicolas R. Jennings. Coalition structure generation over graphs. *Journal of Artificial Intelligence Research*, 45(1):165–196, September 2012.

[219] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, December 2007.

[220] Ken Wakita and Toshiyuki Tsurumi. Finding community structure in mega-scale social networks. In *Proceedings of the 16th International World Wide Web Conference*, pages 1275–1276. ACM Press, 2007.

[221] Joe H. Ward, Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, March 1963.

[222] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[223] Duncan J. Watts. Networks, dynamics, and the small-world phenomenon. *American Journal of Sociology*, 105:493–527, 1999.

[224] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[225] Yen-Chuen Wei and Chung-Kuan Cheng. Towards efficient hierarchical designs by ratio cut partitioning. In *Proceedings of the 1989 IEEE International Conference on Computer-Aided Design*, pages 298–301. IEEE, 1998.

[226] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger. SCAN: A structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 824–833. ACM Press, 2007.

[227] Tianbao Yang, Yun Chi, Shenghuo Zhu, and Rong Jin. Detecting communities and their evolutions in dynamic social networks – a bayesian approach. *Machine Learning*, 82(2):157–189, February 2011.

[228] Luh Yen, Denis Vanvyve, Fabien Wouters, François Fouss, Michel Verleysen, and Marco Saerens. Clustering using a random walk based distance measure. In *Proceedings of the 13th European Symposium on Artifical Neural Networks*, pages 317–324, 2005.

[229] Kai Yu, Shipeng Yu, and Volker Tresp. Soft clustering on graphs. In *Advances in Neural Information Processing Systems 18*, page 05. MIT Press, 2006.

[230] Ümit Yüceer. Discrete convexity: convexity for functions defined on discrete spaces. *Discrete Applied Mathematics*, 119(3):297–304, 2002.

[231] yWorks GmbH. yFiles for Java. http://www.yworks.com/en/products_yfiles_about.html. Last accessed 31 January 2015.

[232] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

[233] Zhiping Zeng, Jianyong Wang, Lizhu Zhou, and George Karypis. Coherent closed quasi-clique discovery from large dense graph databases. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 797–802. ACM Press, 2006.

[234] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data (SIGMOD'96)*, pages 103–114. ACM Press, 1996.

[235] Elena Zheleva, Hossam Sharara, and Lise Getoor. Co-evolution of social and affiliation networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1007–1016. ACM Press, 2009.

[236] Haijun Zhou. Network landscape from a brownian particle's perspective. *Physical Review E*, 67:041908, 2003.

List of Publications

Journal Articles

1. Reinhard Bauer, Gianlorenzo D'Angelo, Daniel Delling, Andrea Schumm, and Dorothea Wagner. The Shortcut Problem – Complexity and Algorithms. *Journal of Graph Algorithms and Applications*, 16(2):447–481, 2012.

2. Robert Görke, Pascal Maillard, Andrea Schumm, Christian Staudt, and Dorothea Wagner. Dynamic Graph Clustering Combining Modularity and Smoothness. *ACM Journal of Experimental Algorithmics*, 18(1):1.5:1.1–1.5:1.29, April 2013.

3. Robert Görke, Andrea Kappes, and Dorothea Wagner. Experiments on Density-Constrained Graph Clustering. *ACM Journal of Experimental Algorithmics*, 19:1.6:1.1–1.6:1.31, September 2014.

Conference Articles

1. Robert Görke, Andrea Schumm, and Dorothea Wagner. Density-Constrained Graph Clustering. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures, 12th International Symposium (WADS'11)*, volume 6844 of *Lecture Notes in Computer Science*, pages 679–690. Springer, 2011.

2. Robert Görke, Andrea Schumm, and Dorothea Wagner. Experiments on Density-Constrained Graph Clustering. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 1–15. SIAM, 2012.

3. Christian Staudt, Andrea Schumm, Henning Meyerhenke, Robert Görke, and Dorothea Wagner. Static and Dynamic Aspects of Scientific Collaboration Networks. In *Proceedings of the 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 522–526. IEEE Computer Society, 2012.

4. Robert Görke, Roland Kluge, Andrea Schumm, Christian Staudt, and Dorothea Wagner. An Efficient Generator for Clustered Dynamic Random Networks. In *Proceedings of the 1st Mediterranean Conference on Algorithms*, pages 219–233. Springer, 2012.

5. Tobias Fleck, Andrea Kappes, and Dorothea Wagner. Graph Clustering with Surprise: Complexity and Exact Solutions. In *Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science*

(SOFSEM'14), volume 8327 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2014.

6. Patricia Iglesias Sánchez, Emmanuel Müller, Uwe Leo Korn, Christian Böhm, Andrea Kappes, Tanja Hartmann, and Dorothea Wagner. Efficient Algorithms for a Robust Modularity-Driven Clustering of Attributed Graphs. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 2015. to appear.

Book Chapters and Other Publications

1. David A. Bader, Henning Meyerhenke, Peter Sanders, Christian Schulz, Andrea Schumm, and Dorothea Wagner. Benchmarking for Graph Clustering and Partitioning. In Jon Rokne and Reda Alhajj, editors, *Encyclopedia of Social Network Analysis and Mining*, pages 71–82. Springer, 2013.

2. Tanja Hartmann, Andrea Kappes, and Dorothea Wagner. Identifikation von Clustern in Graphen. *Informatik Spektrum*, 36(2):144–152, 2013.

Technical Reports

1. Robert Görke, Andrea Schumm, and Dorothea Wagner. Density-Constrained Graph Clustering. Technical report, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT), 2011. Karlsruhe Reports in Informatics 2011-17.

2. Christian Staudt, Andrea Schumm, Henning Meyerhenke, Robert Görke, and Dorothea Wagner. Static and Dynamic Aspects of Scientific Collaboration Networks. Technical Report, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT), 2012. Karlsruhe Reports in Informatics 2012-11.

3. Robert Görke, Roland Kluge, Andrea Schumm, Christian Staudt, and Dorothea Wagner. An Efficient Generator for Clustered Dynamic Random Networks. Technical report, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT), 2012. Karlsruhe Reports in Informatics 2012,17.

4. Tobias Fleck, Andrea Kappes, and Dorothea Wagner. Graph Clustering with Surprise: Complexity and Exact Solutions. Technical Report arXiv:1310.6019, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT), 2013.

5. Tanja Hartmann, Andrea Kappes, and Dorothea Wagner. Clustering Evolving Networks. Technical Report arXiv:1401.3516, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology (KIT), 2014.

Erklärung

Ich versichere, diese Dissertation selbstständig angefertigt und alle benutzten Hilfsmittel vollständig angegeben zu haben. Weiterhin versichere ich, kenntlich gemacht zu haben, was aus Arbeiten anderer und eigenen Veröffentlichungen unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, Februar 2015

Andrea Kappes