OPTIMIZATION FRAMEWORKS FOR GRAPH CLUSTERING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Nate Veldt

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. David F. Gleich, Chair
    Department of Computer Science, Purdue University
Dr. Gregery Buzzard
    Department of Mathematics, Purdue University
Dr. Guang Lin
    Department of Mathematics, Purdue University
Dr. Anthony Wirth
    School of Computing and Information Systems, The University of Melbourne

Approved by:
    Dr. Gregery Buzzard
        Head of the Departmental Graduate Program

Dedicated to my brother Nick,
whose eyes light up when I talk about math.

ACKNOWLEDGMENTS

My work over the past few years has involved many hours of solo activities: writing papers in coffee shops, programming, running numerical experiments, and of course scratching out mathematical proofs on pads of paper. As fun as all those activities are, they would not mean very much if my time at Purdue had not also been filled with productive and enriching interactions with fellow students, research colleagues, mentors, friends, and family.

Acknowledging and thanking my advisor David Gleich seems like the best place to start. None of the research I've done would have been possible without David's guidance, mentoring, and (of course) generous and consistent funding.[1] The papers I've written, conferences I've traveled to, and the research collaborations I've been a part of as a graduate student all happened because of opportunities that David pushed in my direction. I'm grateful to have an advisor who's not just incredibly knowledgeable, but is also so invested in helping his students succeed.

The second big thank you goes to Anthony Wirth at the University of Melbourne in Australia. Tony has been like a second advisor to me ever since the summer I spent working with him on correlation clustering back in 2016. He has helped me become a better researcher and better writer, and has always been an absolute pleasure to collaborate with. David and Tony have taught me an enormous amount about mathematics, computer science, and academic research in general. As a side bonus, I've learned a lot from both of them about how to appreciate really good coffee and high quality food (especially the coffee and food that can be found in Melbourne, Australia!).

I'm grateful for the wonderful research group I've been a part of while at Purdue. Let me thank Kyle Kloster, Nicole Eikmeier, Huda Nassar, Meng Liu, Tao Wu, Yanfei Ren, Yangyang Hou, Varun Vasudevan, Charlie Colley, and Rania Ibrahim for good conversations, compelling research presentations, help with paper proofreading, and all the other activities that have come along with being in the same lab. Thanks also to all my co-authors: Huda, Ananth Grama, Shahin Mohammadi, Michael Mahoney, James Saunderson, Christine Klymko, Cameron Ruggles, and of course David and Tony. It was a privilege and pleasure to collaborate with each of them.

Getting a solid liberal arts education at Wheaton College before heading off to graduate school was one of the best academic decisions I have ever made. There are a few faculty members I'd like to recognize in particular. Terry Perciante's contagious enthusiasm for math was the primary reason I was drawn to study math at Wheaton in the first place. Stephen Lovett was an excellent undergraduate advisor and gave me my first taste of research during a summer program on differential

---

[1] The research in this thesis was generously supported by NSF Awards CCF-1149756 and IIS-1546488.

geometry. I had the chance to take several great classes from Robert Brabenec, who has been teaching math at Wheaton since before the math department was established in 1967. Dr. Brabenec has inspired hundreds (if not thousands!) of students during his more than five decades of service to Wheaton College, and I'm grateful to be one of them. On the computer science side, Thomas VanDrunen (a Purdue CS graduate!) taught the phenomenal programming course I took in my junior year. That class introduced me to a wide range of foundational CS concepts that were extremely useful to me when I began studying computational mathematics. All of these professors were a big part of my decision to carry on in grad school. I also need to thank fellow Wheaton students Jeff Sommars and Nathan Bliss. Jeff and Nathan graduated a year ahead of me, and then coached and encouraged me through the process of getting ready to move on to graduate school, as they did before me. I remember the day that Jeff asked me, "You're going to go to graduate school, right? You should." At that point in my life, before I had started my junior year of college, I had no idea what I was going to do next. Without that vote of confidence, I'm not sure when (or if) it would have occurred to me to seriously pursue that direction.

Let me turn my attention now to all the friends and family members outside the University, who, whether directly or indirectly, encouraged me all along the way.

Resurrection Presbyterian has been a wonderful church home during my time in Indiana. The church community has been one of the best parts about life in Lafayette, and the thing I will probably miss the most when I'm gone. I'll miss Adam Brice's preaching and his mentorship. I'll miss playing music with Erin Thompson on Sundays. I'll miss conversations with Matt Eiles and Matt Thompson about topics relating to our respective PhD programs, as well as all the conversations we had that had nothing to do with our work. I've made so many other close friends through dinners, community groups, board game nights, and other events: Wenbo Wei, Ingrid Pierce, Emmeline "Hi!" Thompson, Emily Eiles, the McFarlins, Grants, Motts, Frenches, Slonims, Huntingtons, Woodens, Godwins, Moores, and so many others. Lauren and I will miss you all very much when we've moved away from Lafayette.

Getting regular exercise, taking mental breaks, and maintaining an active social life are all important ways to stay healthy, happy, and balanced in the midst of a busy graduate student schedule. The Purdue Running Club gave me an outlet for all those things, and they deserve a shout-out. I'm grateful in particular for my coach and good friend David Evans, and my main workout and running buddies Conner Sandt, Matt Sraders, Mike Howell, Thomas Hembree, Tyler Azbell, and Max Runningen.

Turning now to family members, I'll start by thanking my parents. There's no time or space to adequately acknowledge all the ways they've supported me throughout the years, but let me mention a few specific ways in which they prepared me to succeed in graduate school. For starters, my mom was my first (and by far the best) teacher I had growing up. We moved around a lot when I was younger, and for various reasons she home-schooled my siblings and me for a number of years. In addition to all the effort she spent preparing and teaching individual courses for all of us, she taught us self-motivation and discipline in our work. These served me well as the backbone of all of my subsequent academic endeavors. My dad has a master's degree in literature and instilled in all his kids the importance of good writing and proper grammar. Beyond that though, he was the one who gave me a strong curiosity for math when I was still very little. He taught me how to count up to 1023 on my fingers in binary, and proved to me why $0.\bar{9} = 1$. I'm not sure if he realized at the time that he was planting seeds that would eventually grow into enough love for math that I'd pursue a PhD in the subject.

My in-laws, Dave and Kim, have been incredibly supportive throughout my time in graduate school. In addition to letting me marry their daughter and welcoming me into the family, they have always shown a great deal of interest in my work and my progress through the PhD program. I also have quite a few siblings and siblings-in-law that deserve an acknowledgment for their support throughout the years. I'll list them alphabetically so that they don't read too much into the order of their names: Alli, Amber, Andrea, Anna, Ben, Daniel, Jacob, Michael, Nick and Tim. I do need to give a special acknowledgment to my younger brother Nick. His eyes light up, rather than glaze over, when I talk about anything related to my research. He's always been far smarter than I am, and I'm looking forward to seeing what he'll be up to next once he has wrapped up his double major in mathematics and mechanical engineering. As Wodehouse might say, I shall watch his future progress with considerable interest.

This brings me now to my wife Lauren. After we got married, Lauren gave up city life in Chicago to move to Lafayette, Indiana to be with me while I continued to pursue my degree at Purdue. She has been endlessly supportive of my work, and at the same time keeps me from getting tunnel vision when it comes to career and research. She has made every aspect of my life easier, more balanced, and more joyful. I owe her a lot more than I can cover in a paragraph of acknowledgment.

Above all things, I'm grateful to God. As a Christian, I believe the most important acknowledgment I can make is to acknowledge Jesus Christ as Lord and master over every aspect of my life, which certainly includes my graduate studies. Studying mathematics gives me the opportunity to explore just a small part of the truth and beauty that he has embedded into his creation, and I consider that a great privilege. I hope that in my work I have managed to follow (though I have done so imperfectly at best) the advice that Paul of Tarsus gave to his friends at Colossae: "Whatever you do, work at it with all your heart, as working for the Lord, not for human masters."

TABLE OF CONTENTS

Page

LIST OF TABLES xi

LIST OF FIGURES xii

ABSTRACT xv

1 INTRODUCTION 1
- 1.1 Principles of Graph Clustering 2
- 1.2 Graph Clustering as Combinatorial Optimization 4
- 1.3 Contributions 6

2 BACKGROUND 9
- 2.1 Graph Clustering Notation 9
- 2.2 Clustering, Partitioning, and Community Detection 10
- 2.3 Computational Complexity Terminology 11
- 2.4 Graph Clustering Objective Functions 12
  - 2.4.1 Cut-to-Size Ratio Objectives 12
  - 2.4.2 Modularity-Based Objectives 13
  - 2.4.3 Cluster Graph Modification Objectives 15
- 2.5 Correlation Clustering Background 16
  - 2.5.1 Formal Objectives 16
  - 2.5.2 Theoretical Results 17
  - 2.5.3 Applications and Scalable Algorithms 18
- 2.6 Graph Clustering Equivalence Results 19

3 THE LAMBDACC GRAPH CLUSTERING FRAMEWORK 21
- 3.1 Chapter Overview 21
- 3.2 Graph Clustering as Correlation Clustering 21
- 3.3 The LambdaCC Construction 22
  - 3.3.1 Alternative LambdaCC Construction 24
  - 3.3.2 Equivalent Formulations of the LambdaCC Objective 26
- 3.4 Theoretical Equivalence Results 28
  - 3.4.1 Equivalence with Hamiltonian 28
  - 3.4.2 Connection to Sparsest Cut 29
  - 3.4.3 Connection to Cluster Modification Objectives 31
  - 3.4.4 Summary of Equivalences 32
- 3.5 LambdaCC Clustering Heuristics 33
- 3.6 Experiments 36
  - 3.6.1 Algorithm Comparison with the LambdaCC Framework 37
  - 3.6.2 Social Network Analysis 41

4 COMPLEXITY RESULTS FOR LAMBDACC 45
- 4.1 Chapter Overview 45
- 4.2 Heavily Negative Weighted Correlation Clustering 46
  - 4.2.1 Deterministic Pivoting Theorem 46
  - 4.2.2 Approximations for Probability Weights 48
  - 4.2.3 Extending the Result 51

Page
4.3 LambdaCC Approximation Algorithms 53
4.3.1 Parameter Dependent Guarantees for LambdaCC 55
4.3.2 Two-Approximation for Cluster Deletion 60
4.4 Standard LambdaCC Integrality Gap 63

5 METRIC CONSTRAINED OPTIMIZATION FOR GRAPH CLUSTERING 67
5.1 Chapter Overview 67
5.2 Metric LPs and Graph Clustering 68
5.3 Background: Projection Methods and Quadratic Programming 74
5.3.1 Strictly Convex Quadratic Programming 75
5.3.2 The Best Approximation Problem 76
5.3.3 Dykstra's Method for BAP 76
5.3.4 Hildreth's Projection Method 77
5.3.5 Equivalence with Coordinate Descent 80
5.4 Dykstra's Method for Metric Constrained Optimization 81
5.4.1 Efficient local updates 82
5.4.2 Sparse storage of dual variables 83
5.4.3 Robust Stopping Criteria 84
5.4.4 Entrywise Rounding Procedure 86
5.5 Approximation Guarantees for Clustering Objectives 87
5.5.1 Cluster Deletion Approximation 87
5.5.2 Correlation Clustering 88
5.5.3 Sparsest Cut 89
5.6 Improved a Posteriori Approximations 91
5.6.1 A First Strategy for Improved Bounds 92
5.6.2 Improved Guarantees by Solving a Small LP 92
5.7 Experiments 95
5.7.1 Implementation Details for Convergence Check 96
5.7.2 Using Gurobi Software 96
5.7.3 Real-world Graphs 97
5.7.4 The Sparsest Cut Relaxation 98
5.7.5 Weighted Correlation Clustering 101
5.7.6 Maximum Modularity Clustering via LP Rounding 103

6 LEARNING GRAPH CLUSTERING RESOLUTION PARAMETERS 107
6.1 Chapter Overview 107
6.2 Generalized Clustering Objectives 108
6.2.1 Local Graph Clustering 108
6.2.2 Parametric Linear Programs 110
6.3 Theoretical Results 111
6.3.1 Problem Formulation 111
6.3.2 Parameter Fitness Function 112
6.3.3 Minimizing the Parameter Fitness Function 114
6.4 Application to Specific Objectives 116
6.4.1 Local Clustering 117
6.4.2 Global Clustering Approach 118
6.5 Experiments 120
6.5.1 Learning Parameters for Synthetic Datasets 121
6.5.2 Local Community Detection 121
6.5.3 Metadata and Global Clustering 124
6.5.4 Local Clustering in Social Networks 125

7 CONCLUSIONS 129

Page
7.1 Discussion and Open Questions 129
7.1.1 Improved Complexity Results for LambdaCC 130
7.1.2 $O(n^2)$-memory algorithms for metric optimization 133
7.1.3 Parallel projection methods for metric optimization 135
REFERENCES 142

LIST OF TABLES

Table Page

1.1 Intuitive definitions of graph clustering. 3

1.2 Chapter contributions. 7

5.1 We solve the LP relaxation for sparsest cut via DykstraSC on 19 graphs. Both DykstraSC and Gurobi (when it doesn't run out of memory) solve the problems to within a relative gap tolerance of 10^{-4}, and satisfy constraints to within machine precision. Time is given in seconds. 100

5.2 DykstraCC solves convex relaxations of correlation clustering with up to 700 billion constraints. The lazy-constraint Gurobi solver does very well for one very sparse graph, but runs out of memory on all other problems. We set $\gamma = 1$, and constraint tolerance to 0.01. Selecting a small γ leads to poorer approximation guarantees, but dramatically decreases the number of needed iterations until convergence. For problems on which we cannot optimally solve the LP with Gurobi to obtain an approximation ratio Δ, we report an upper bound. 104

5.3 We approximate the LP relaxation of modularity using our projection methods with $\gamma = 2$. Runtime for solving the relaxation is given in column 3. (UB) is an upper bound on the maximum modularity, computed using (5.44). We obtain clusterings with the Louvain method (Louv), a simple LP rounding technique (3LP), and by using the output of 3LP as input to Louvain. We report maximum and median modularity scores over 15 trials. Shown in bold are median and maximum scores that are higher than those achieved with only Louvain. 106

6.1 We list the number of nodes (n) and edges (m) in each SNAP network, along with average set size $|T|$ and set conductance $\mathbf{cond}(T)$ for the ten largest communities. 122

6.2 For experiments on SNAP datasets, we give F1 scores, conductance scores, runtimes, and output set sizes for finding the minimum conductance subset (mc), and for the set returned by learning a good resolution parameter (lr). Displayed is the average over results for the 10 largest communities in each network. 122

7.1 Results for running the parallel version of Dykstra's method in solving the metric constrained LP relaxation of correlation clustering. 136

LIST OF FIGURES

Figure Page

1.1 Two sets of nodes are displayed in Newman's Netscience graph. The red set embodies community structure, while the blue set does not. 2

3.1 We convert a toy graph (left) into a signed graph for standard (middle) and degree-weighted (right) LambdaCC. Dashed red lines indicate negative edges. Partitioning the signed graph via correlation clustering induces a clustering on the original unsigned graph. 23

3.2 LambdaCC is equivalent to several other objectives for specific values of $\lambda \in (0, 1)$. Values λ^* and ρ^* are not known a priori, but can be obtained by solving LambdaCC for increasingly smaller values of λ. 33

3.3 Each curve in each plot (except the Lambda-Louvain curve in black) represents a single clustering output by a certain algorithm. For each clustering, the top row of plots shows the ratio between the clustering's LambdaCC score and a lower bound on the optimal score determined by a solving an LP relaxation. The black curve represents the performance of Lambda-Louvain. This algorithm recomputes a new clustering for each value of λ. The second and third row show NMI and ARI scores between each clustering and the Lambda-Louvain clustering at each λ. 40

3.4 As we increase the number of clusters formed by Graclus (left), the algorithm does better for large values of λ and worse for small values. The algorithm seems particularly well-suited to optimize the LambdaCC objective for very small values of λ. Darker curves represent a larger number of clusters formed; the number of clusters formed ranges from 2 on the far left of the plot to just over 2200 for the right-most curve. In the right plot we vary the density ρ of quasi-cliques formed by RMQC from 0.5 to 0.85. In this plot, darker curves represent a larger density. As density increases, the curves converge to the performance of RMC, shown in blue. 41

3.5 On four university Facebook graphs, we illustrate that the dorm (red), graduation year (green), and student/faculty (S/F) status (blue) metadata attributes all correlate highly with the clustering found by Lambda-Louvain for each λ. Above the x-axis we show the number of clusters formed, which strictly increases with λ. The y-axis reports the probability that two nodes sharing a cluster also share an attribute value. Each attribute curve is compared against a null probability, shown as a dashed line of the same color. The dashed line reports the same probability computed for a related *fake* attribute, which was generated by randomly permuting the values of the corresponding true attribute. The large gaps between each attribute curve and its null probability indicate that the link structure of all networks is highly correlated with these attributes. In general, probabilities for *year* and *s/f* status are highest for small λ, whereas *dorm* has a higher correlation with smaller, denser communities in the network. Caltech is an exception to the general trend; see the main text for discussion. 43

Figure Page

4.1 We illustrate the 8 different types of triangles in the original graph G that could be mapped to a bad triangle $\{i, j, k\}$ in $\tilde{G}$ with two positive edges $\{(i, j), (j, k)\} \subset F^+$ and a negative edge $(i, k) \in F^-$. Negative edges are shown with dashed lines. We must check in each case that the inequality (4.24) holds. Cases 1-2 can be handled together, as can cases 5-6, due to the symmetric relationship between (i, j) and (j, k), the two positive edges in the bad triangle in $\tilde{G}$. 56

4.2 Case 1: We explicitly use the fact that $\lambda > 1/2$ here. The same set of steps works for Case 2, if we switch the roles of edges (i, j) and (j, k). This is the case that will make the proof fail if we deal with arbitrarily small λ, indicating as before that LambdaCC for small λ is somehow more challenging than large λ. 56

4.3 Case 3: When all edges of the triangle in G are negative, the bound is very loose since the left hand side of inequality (4.24) is just λ and we can easily bound the right hand side below. 57

4.4 Case 4: If the same type edges in G map to the same type edges in $\tilde{G}$, the inequality we are trying to prove is almost tight (if $x_{ik} = x_{jk} + x_{ij}$ and the variables are near their upper bound), and we must make explicit use of the fact that $\lambda > 1/2$. 57

4.5 Case 5: This case is easy to show, independent of λ. Exactly the same proof holds for case 6. 58

4.6 Case 7: The left hand side is zero, so the proof is trivial 58

4.7 Case 8: The key for this case is to use the fact that $1/3 \leq x_{ik} \leq x_{ij} + x_{jk}$. 59

5.1 Runtimes for DykstraSC on real-world graphs from Table 5.1. If n is the number of nodes in the graph, then DykstraSC solves for $n(n-1)/2$ distance scores. 99

5.2 Convergence plots for DykstraSC on several graphs. 101

5.3 Convergence plots for DykstraCC on four graphs. 103

6.1 Left: Function $\mathcal{P}$ satisfies both properties (a) and (b) in Theorem 6.3.1. If $\mathcal{P}(\beta_1) = \mathcal{P}(\beta_2)$, querying $\mathcal{P}$ at any point $\beta_3 \in [\beta_1, \beta_2]$ gets us closer to a minimizer. Right: Function $\mathcal{Q}$ satisfies property (a) but not property (b). If $\mathcal{Q}(\beta_1) = \mathcal{Q}(\beta_2)$, we can get stuck evaluating $\mathcal{Q}$ in a flat region not near a minimizer. 114

6.2 We evaluate $\mathcal{P}$ at left and right bounds (blue points), and at a midpoint m (red point). Left: If $\mathcal{P}(\ell) < \mathcal{P}(m) < \mathcal{P}(r)$, then we know the minimizer of $\mathcal{P}$ is in $[\ell, m]$, and we recursively call the *one-branch* phase (Algorithm 1) with new bounds ℓ and m. Right: If $\mathcal{P}(m) < \mathcal{P}(\ell) < \mathcal{P}(r)$, we don't know if the minimizer is in the left branch $[\ell, m]$ or right branch $[m, r]$. Evaluating $\mathcal{P}$ at the midpoint of each branch (purple points), we rule out branch $[m, r]$ and recursively call Algorithm 2 with new endpoints ℓ and m and midpoint ℓ_{mid}. 115

6.3 Left: ARI scores for detecting ground truth in LFR graphs. Solid lines indicate median scores, and colored regions show the range of scores across five test graphs for each μ. Right: one of the five LFR test graphs for $\mu = 0.3$. Modularity ($\lambda = 1/(2|E|)$) makes mistakes by putting distinct ground truth clusters together (highlighted). For this example our approach perfectly detects the ground truth. 120

Figure Page

6.4 Left: F1 scores for detecting clusters X by learning α vs. the minimum of $\mathcal{P}_X$. Right: F1 scores obtained by finding $X = \text{argmin}_{S \subseteq R} \, \mathbf{cond}(S)$ vs. $\mathbf{cond}(X)$. The decreasing, nearly linear pattern in the first plot indicates that the minimum value of $\mathcal{P}_X$ tells us something about how well the targeted functional communities match a notion of structural communities in a network. The right plot indicates $\mathbf{cond}(X)$ does little to help us predict how detectable a cluster will be. 123

6.5 Minimizing $\mathcal{P}_X$ gives us refined information about the connectivity structure of different sets sharing metadata attributes in Facebook 100 networks. Plotting F1 detection scores against the minimum of $\mathcal{P}_X$ shows especially clear trends for the gender and residence metadata attributes. Plots for the graduation year attribute highlight an anomaly in the connectivity patterns of the 2009 graduating year classes. We explore this in further depth in the main text. 127

6.6 As graduation year increases, conductance scores on the whole tend to decrease. White dots indicate median value. The 2009 graduation year metadata sets tend to be much smaller in volume, but also have very small cut scores, indicating that freshman in 2005 were largely connecting on Facebook with people in their same class. 128

7.1 Ratios between optimal LambdaCC clustering score and the LambdaCC LP lower bound, as λ increases. The left plot shows this integrality gap for Karate (34 nodes), and the right plot shows the gap for Dolphins (62 nodes). 133

7.2 Convergence plots for Polbooks. 138

7.3 Convergence plots for Football. 139

7.4 Convergence plots for C Elegans Metabolic. 140

7.5 Convergence plots for Jazz. 141

ABSTRACT

Veldt, Nate PhD, Purdue University, May 2019. Optimization Frameworks for Graph Clustering. Major Professor: David F. Gleich.

In graph theory and network analysis, *communities* or *clusters* are sets of nodes in a graph that share many internal connections with each other, but are only sparsely connected to nodes outside the set. Graph clustering, the computational task of detecting these communities, has been studied extensively due to its widespread applications and its theoretical richness as a mathematical problem. This thesis presents novel optimization tools for addressing two major challenges associated with graph clustering.

The first major challenge is that there already exists a plethora of algorithms and objective functions for graph clustering. The relationship between different methods is often unclear, and it can be very difficult to determine in practice which approach is the best to use for a specific application. To address this challenge, we introduce a generalized discrete optimization framework for graph clustering called LambdaCC, which relies on a single tunable parameter. The value of this parameter controls the balance between the internal density and external sparsity of clusters that are formed by optimizing an underlying objective function. LambdaCC unifies the landscape of graph clustering techniques, as a large number of previously developed approaches can be recovered as special cases for a fixed value of the LambdaCC input parameter.

The second major challenge of graph clustering is the computational intractability of detecting the *best* way to cluster a graph with respect to a given NP-hard objective function. To address this intractability, we present new optimization tools and results which apply to LambdaCC as well as a broader class of graph clustering problems. We first of all develop polynomial-time approximation algorithms for LambdaCC and other related clustering objectives. In particular, we show how to obtain a 2-approximation for cluster deletion, which improves upon the previous best approximation factor of 3. We also present a new optimization framework for solving convex relaxations of NP-hard graph clustering problems, which are frequently used in the design of approximation algorithms. Finally, we develop a new framework for efficiently setting tunable parameters for graph clustering objective functions, so that practitioners can work with graph clustering techniques that are especially well suited to their application.

1. INTRODUCTION

A graph is a mathematical structure representing a set of objects, called nodes or vertices, which share pairwise relationships, called edges. The term ‘network’ is nearly synonymous, and is specifically used to refer to a complex system of interacting agents that, at its core, can be modeled and studied using a graph. Complex systems studied by network scientists include transportation networks (e.g., road networks, airline networks), technological networks (e.g., the Internet, cell phone networks), social networks (e.g., friendship networks), and biological networks (e.g., protein interaction networks, food webs, neural systems), to name only a select few.

One of the most fundamental problems in network science and graph theory is to identify sets of nodes that are more closely connected to each other than to the rest of the graph. This basic task goes by many (nearly synonymous) names, including graph partitioning, graph clustering, and community detection.[1] The ability to separate a graph into meaningful clusters of related vertices has, among many others, the following applications:

- finding related genes in a biological network,
- segmenting an image into shapes corresponding to objects in a picture,
- analyzing the community structure of a social network,
- partitioning a computational problem into tasks that can be performed more efficiently in parallel,
- identifying different regions and types of tissue in a clinical MRI scan,
- detecting anomalous behavior and fraudulent activity in healthcare networks and financial transaction networks.

In addition to its widespread applications, graph clustering is a very rich problem mathematically, and has been studied extensively from a theoretical perspective by mathematicians, physicists, computer scientists, statisticians, and experts from various other academic domains. Despite the wealth of known results and the vast landscape of already existing techniques, graph clustering remains a challenging problem. In this thesis we explore unifying frameworks and generalized algorithms for addressing two of the central problems in graph clustering: defining community structure in graphs, and algorithmically *detecting* that structure in practice.

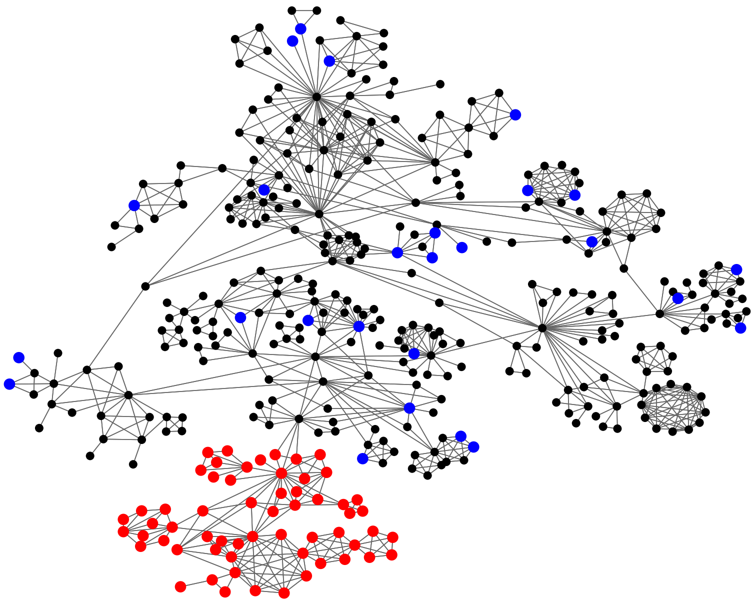
---

[1] This thesis will predominantly use the term graph clustering, but will interchangeably refer to ‘clusters’ and ‘communities’ in a graph. A more in depth discussion follows in Chapter 2.

1.1 Principles of Graph Clustering

Figure 1.1 is a picture of the largest connected component of the popular Netscience graph [Newman, 2006]. Nodes in this graph represent academic researchers, and edges indicate co-authorship on a paper on the topic of network science. Two sets of nodes have been highlighted, one in blue, the other in red. Without covering any mathematical background, one may easily guess that the red set is more likely to be considered a 'community' than the blue set. Blue nodes, which were selected at random, are scattered across the graph and share few connections with one another, whereas the red nodes appear tightly connected internally and are, for the most part, separated from the rest of the graph. Indeed, the red nodes do correspond to a community of collaborating researchers in academia.

Figure 1.1.: Two sets of nodes are displayed in Newman's Netscience graph. The red set embodies community structure, while the blue set does not.



Although Figure 1.1 provides an intuitive visual aid for understanding the task of graph clustering, it is oversimplified in a number of ways. To begin with, real-world graphs are usually much larger and do not come with a clean two-dimensional layout that visually highlights community structure. Furthermore, although it is clear that the red set in Figure 1.1 is a *better* community than the blue set, in practice it becomes very challenging to understand how to find the *best* partitioning of a graph into communities for a specific application. In this thesis we address these fundamental questions in depth, but this will of course require more than a picture of a small graph.

Defining a Community Graph clustering has been studied extensively across different disciplines, and there is significant general agreement about the fundamental characteristics of a graph community. Table 1.1 lists several quotes on the basic definition of a community, taken from widely-cited and well-established papers on the topic. All of these explanations can be applied to better understand why the red set in Figure 1.1 embodies community structure, whereas the blue set does not.

Table 1.1.: Intuitive definitions of graph clustering.

"A community is often thought of as a set of nodes that has more connections between its members than to the remainder of the network." [Leskovec et al., 2008]
"Graph clustering is the task of grouping the vertices of the graph into clusters taking into consideration the edge structure of the graph in such a way that there should be many edges within each cluster and relatively few between the clusters." [Schaeffer, 2007]
"One mesoscopic structure, called a community, consists of a group of nodes that are relatively densely connected to each other but sparsely connected to other dense groups in the network." [Porter et al., 2009]
"Communities, or clusters, are usually groups of vertices having higher probability of being connected to each other than to members of other groups, though other patterns are possible." [Fortunato and Hric, 2016]
"The most basic task of community detection, or graph clustering, consists in partitioning the vertices of a graph into clusters that are more densely connected." [Abbe, 2018]
"Generally speaking, techniques for graph partitioning and graph clustering aim at the identification of vertex subsets with many internal and few external edges." [Bader et al., 2013]
"A property that seems to be common to many networks is community structure, the division of network nodes into groups within which the network connections are dense, but between which they are sparser." [Newman and Girvan, 2004]

These quotes highlight two basic guidelines for graph clustering:

1. Communities should have a high internal density, i.e., there should be many edges between nodes in the same community.

2. Communities should have sparse external connections, i.e., there should not be too many edges connecting a community to the rest of the graph.

Nearly every approach to graph clustering rests, in one way or another, on the tension between these two (often conflicting) design goals. In order to find clusters with higher internal density, it may become necessary to exclude nodes that share many, but perhaps not "enough" edges within a community. The denser we require a community to be, the more we exclude such nodes. This in turn leads to a higher number of edges between communities. On the other hand, if we want to simply avoid edges between communities, we could place all nodes in a single cluster. This, however, would be mathematically trivial and unhelpful in applications. Therefore, understanding how to strike the right balance between internal density and external sparsity is one of the most essential decisions in graph clustering. At this point, widespread agreement about the high-level definition of community detection quickly turns into a plethora of different ways to formalize and study the problem in theory and practice.

1.2 Graph Clustering as Combinatorial Optimization

We approach the task of graph clustering from the common strategy of combinatorial optimization. In other words, the problem is formalized by introducing an objective function that assigns a numerical score to each discrete clustering of a graph, measuring how well it embodies community structure. Optimizing the objective function over all possible clusterings will then return "the best" partitioning of the graph with respect to a well-defined measure. As an example, one way to partition a graph G into two pieces is to identify a set of nodes S that minimizes the following function f, referred to as the *normalized cut* objective [Shi and Malik, 2000]:

$$f(S) = \frac{\text{number of edges leaving } S}{\text{number of edge endpoints in } S} + \frac{\text{number of edges leaving } S}{\text{number of edge endpoints \textit{not} in } S}\,. \tag{1.1}$$

Minimizing f will produce two clusters (nodes in S, and nodes not in S) that are both nontrivial in size and share few edges with each other. The denominators in expression (1.1) encourage clusters with a large number of internal edge endpoints, i.e., a high internal density of edges. Including the "number of edges leaving S" in the numerators ensures that the optimal output will share few edges with the rest of the graph, i.e. external sparsity is encouraged. Looking back to Figure 1.1, if we let S_r represent the set of red nodes and use S_b to denote blue nodes, a quick calculation shows that $f(S_r) \approx 0.0119$ and $f(S_b) \approx 1.0319$. The fact that the normalized cut score of S_r is so much lower than the score of S_b matches our intuition that S_r is a good cluster, and S_b is not.

Related Strategies for Graph Clustering Optimizing an objective function is not the only existing strategy for graph clustering. Fortunato and Hric [2016] provide an overview of optimization based methods in their survey on graph clustering. In addition to this, they present an overview of methods based on statistical inference, and methods based on dynamics. In short, statistical inference methods posit the existence of some underlying distribution of graphs with community structure. Clustering a graph is then equivalent to inferring the parameters of the distribution and learning the cluster assignment which maximizes the likelihood of observing the given graph. Dynamics-based methods for graph clustering rely on the idea that a dynamic process in the network, such as a random walk or another type of diffusion, will stay localized when it encounters a tightly connected cluster of nodes. More intuitively, if a random walker visits nodes in a graph by following edges at random, they are likely to get "trapped" inside communities. We would expect the walker to wander around the (many) internal connections of a cluster for a long while, before following one of the (few) external edges and escaping the cluster. Observing when a dynamic process gets trapped in this way is one way to detect communities.

Each of these approaches constitutes a slightly different way of thinking about graph clustering. However, they are by no means contradictory nor conflicting, but are in fact closely related to each other on a fundamental level. Random walks on graphs are known to be intimately related to spectral relaxations of graph clustering objective functions that one might wish to optimize [Chung, 1997]. Furthermore, many diffusion-based methods are known to output clusters that satisfy strong guarantees with respect to objectives like conductance [Andersen et al., 2006, Chung, 2009, Wang et al., 2017]. There are also known equivalences between assigning communities based on the stochastic block model [Abbe, 2018] (a method based on statistical inference) and common graph clustering objectives such as minimum cut [Newman, 2013] and maximum modularity [Newman, 2016]. Finally, clustering with the dynamics-based personalized PageRank was shown to share strong connections with inferring community structure based on the stochastic block model [Kloumann et al., 2017]. These results are only a sample of the deep connections that exist between optimization-based methods and other existing approaches.

Algorithms and Computational Complexity Defining community structure is only the first step in actually detecting communities. Given an objective function, the next crucial step is to develop reasonable techniques for optimizing it. Algorithm development and the study of computational complexity are therefore key components in graph clustering research.

The majority of objective functions for graph clustering that have been introduced and analyzed are known to be NP-hard, meaning that it is infeasible to optimize them exactly in practice. A major focus in graph clustering is to develop efficient algorithms for community detection that run quickly and can detect meaningful clustering structure in theory and practice. Many of these algorithms are designed to heuristically optimize a specific function, often by making greedy local moves to cluster assignments in order to improve the objective function score. However, these techniques typically do not come with any guarantees for how well the underlying objective is approximated, and may perform poorly in the worst case. A more rigorous approach to graph clustering is to develop efficient approximation algorithms for NP-hard objectives. These are algorithms that run in polynomial time, and output a clustering that provably approximates the optimal solution to within a certain multiplicative factor. Approximation algorithms are widely studied in the field of theoretical computer science, and well-known algorithms exist for many graph clustering objectives.

Understanding the computational complexity of a given graph clustering problem also involves proving theoretical results about fundamental limits associated with solving or approximating an objective function. The fact that most clustering objectives are NP-hard indicates that the only known algorithms for optimally solving these objectives run in exponential time. There also exist more refined notions of computational complexity, which can be used to show that even obtaining certain types of approximations is challenging. We provide more formal details in the next chapter.

1.3 Contributions

This thesis presents novel optimization frameworks for graph clustering. This includes a unifying combinatorial (NP-hard) optimization framework, as well as continuous and convex optimization tools that are useful for graph clustering applications. These frameworks and tools are designed to address two major challenges: (1) understanding the tradeoffs between existing graph clustering techniques, to determine which approach best fits a given application, and (2) dealing with the computational intractability of graph clustering.

The central and unifying contribution of this thesis is a novel framework for graph clustering, called LambdaCC. Chapter 3 introduces this framework and proves a number of equivalence results between LambdaCC and several previously studied objective functions for graph clustering. The subsequent chapters build on the LambdaCC framework by providing algorithms, hardness results, and practical optimization tools for applying the framework in practice. All of the tools developed in these chapters are motivated strongly by LambdaCC. However, they can be viewed more broadly as frameworks for solving key convex and continuous optimization problems that arise often in graph clustering applications. Table 1.2 provides a short overview of each chapter, including each chapter's specific relationship to the LambdaCC framework, as well as its broader contributions to the graph clustering literature on the whole.

Many of the results presented in this thesis were first shown in a sequence of previous publications (see references in Table 1.2). This thesis provides a unified and expanded presentation of these results, along with several additional contributions. All of the results in Chapters 3 through 6 were

Table 1.2.: Chapter contributions.

	Implications for LambdaCC	**Broader contributions**
Chapter 3 *The LambdaCC Graph Clustering Framework* [Veldt et al., 2018b]	Defines and proves equivalence results for LambdaCC.	Unifies a large number of previously studied graph clustering objective functions.
Chapter 4 *Complexity Results for LambdaCC* [Veldt et al., 2018b, Gleich et al., 2018]	Gives polynomial-time approximation algorithms for LambdaCC. Proves an $\Omega(\log n)$ integrality gap for the LP relaxation.	Improves approximation guarantees for a weighted version of correlation clustering studied by Puleo and Milenkovic [2015], which generalizes LambdaCC when $\lambda > 0.5$.
Chapter 5 *Metric-Constrained Optimization for Graph Clustering Relaxations* [Veldt et al., 2018a]	Details a new approach for solving the LambdaCC LP relaxation, needed for implementing the lowest-factor approximation algorithms.	Establishes a memory-efficient optimization framework for solving the *metric constrained* relaxations of NP-hard graph clustering objectives, which are frequently used in the design of approximation algorithms.
Chapter 6 *Learning Graph Clustering Resolution Parameters* [Veldt et al., 2019b]	Develops an approach for setting the LambdaCC resolution parameter λ to capture different types of community structure.	Presents a general framework for learning clustering resolution parameters for graph clustering objectives, when given a single example of a "good" clustering in a particular application domain.

developed in collaboration with Professor David Gleich (Purdue University) and Professor Anthony Wirth (The University of Melbourne). The results in Chapter 5 were additionally developed in collaboration with Professor James Saunderson (Monash University).

Experiments and Code In addition to theoretical contributions, Chapters 3, 5, and 6 each contain a section on experiments results. (Chapter 4 is primarily focused on theoretical approximation algorithms.) These experimental sections demonstrate how each developed method leads to improved results for community detection problems in both synthetic and real-world networks (e.g., social networks, collaboration networks, biological networks, etc.). Code for all algorithmic implementations has been made publicly available online at `https://github.com/nveldt/`.

2. BACKGROUND

This chapter establishes key terminology and notation for optimization-based graph clustering. This includes an overview of several well-known objective functions that have been studied from the perspective of combinatorial optimization. For more extensive background on graph clustering, the interested reader may refer to any one of several helpful overviews and surveys [Porter et al., 2009, Schaeffer, 2007, Fortunato, 2010, Fortunato and Hric, 2016].

In the second half of the chapter we cover relevant technical background on correlation clustering, a framework for partitioning datasets characterized by positive and negative pairwise relationships. A major contribution of this thesis is a unifying framework for graph clustering that is based on a new specially weighted version of correlation clustering called LambdaCC. The algorithms and complexity results we prove for LambdaCC rely heavily on algorithms and techniques developed in the correlation clustering literature, so we provide necessary technical background here, as well as a short survey of previous results.

2.1 Graph Clustering Notation

Graphs and Cuts Throughout the manuscript, we will use $G = (V, E)$ to denote an unweighted, undirected graph with $n = |V|$ nodes and $m = |E|$ edges. We always assume loopless graphs, and with few exceptions consider only connected graphs. The degree of a node $v \in V$ is the number of edges incident on v and is denoted by d_v. For two disjoint sets of nodes $S, T \subseteq V$, $\text{cut}(S, T)$ represents the number of edges between them. For every set of nodes $S \subseteq V$ we let $\bar{S} = V \backslash S$ denote its complement set, and $E_S \subset E$ denotes the set of edges in the induced subgraph of S. We also have the following measures:

$$\begin{aligned} \text{cut}(S) &= \text{cut}(S, \bar{S}) \\ \text{vol}(S) &= \sum_{v \in S} d_v \\ \text{density}(S) &= \frac{|E_S|}{\binom{|S|}{2}} \end{aligned}$$

For a single node $v \in V$, by convention we let $\text{density}(v) = 1$. Since each set $S \subset V$ is uniquely identified with a set of edges between itself and the rest of the graph, we will frequently refer to a set of nodes as a *cut* in a graph.

Graph Clusterings We use $\mathcal{C}$ to denote a graph clustering. Although overlapping community detection is also important and well-studied, in this thesis we focus on equivalence results, frameworks, and algorithms for non-overlapping clustering. Thus, for our purposes a clustering is synonymous with a partition of the nodes, i.e., $\mathcal{C} = \{S_1, S_2, \ldots, S_k\}$ is a k-clustering if $S_i \cap S_j = \varnothing$ for every $i \neq j$, and $\bigcup_{i=1}^{k} S_i = V$. We encode a clustering $\mathcal{C}$ of a network using a clustering indicator function $\delta^{\mathcal{C}} = (\delta^{\mathcal{C}}_{ij})$ where

$$\delta^{\mathcal{C}}_{ij} = \begin{cases} 1 & \text{if nodes } i,j \text{ are clustered together} \\ 0 & \text{if nodes } i,j \text{ are in different clusters.} \end{cases} \tag{2.1}$$

We frequently also encode clusterings using a set of variables $\mathbf{x}^{\mathcal{C}} = (x^{\mathcal{C}}_{ij})$ where $x^{\mathcal{C}}_{ij} = 1 - \delta^{\mathcal{C}}_{ij}$. These represents the binary distance between two nodes: i, j have distance 1 if they are clustered apart, and 0 otherwise. Throughout the manuscript, graph clustering is cast as the task of optimizing an objective function $f : \mathscr{C} \to \mathbb{R}$, where $\mathscr{C}$ denotes a set of valid clusterings. For every clustering $\mathcal{C} \in \mathscr{C}$, the function f outputs a real-valued objective *score* $f(\mathcal{C}) \in \mathbb{R}$ which gives a measure of how well $\mathcal{C}$ embodies community structure.

2.2 Clustering, Partitioning, and Community Detection

The terms *graph clustering*, *community detection*, and *graph partitioning* all show up in the quotes in Table 1.1. The words *cluster* and *community* are typically used interchangeably to refer to densely connected nodes that are only loosely connected to the rest of the graph. Following the standards in the literature, we will use the terms *community detection* and *graph clustering* synonymously in this thesis. Graph partitioning, although very similar, is sometimes contrasted from these terms in a few key ways. We highlight these before moving on to more formal definitions.

Graph partitioning is often presented in the context of parallel computing [Newman and Girvan, 2004, Schloegel et al., 2003, Schulz, 2013], where nodes in a graph represent computational tasks, and edges indicate some form of data dependence. In this case, if there are k processors for accomplishing the work, it is beneficial to split up the tasks into exactly k clusters in a way that minimizes communication between processors. In order for the work to be balanced among the k processors, it is important to separate nodes into blocks of roughly the same size. Consistent with this example, most formalizations of graph partitioning specify the number of clusters k to form, and include a balance constraint or penalty on cluster sizes [Schaeffer, 2007, Schulz et al., 2016, Fortunato, 2010, Van Dongen, 2000, Falkner et al., 1994]. The goal is then to find the k-way clustering that satisfies the balance constraint and minimizes the number of edges between clusters. Given the emphasis on minimizing communication, graph partitioning often places a higher priority on external sparsity than on internal density.

The design goals of graph partitioning occasionally conflict with the motivation behind certain applications of graph clustering and community detection. In many real-world networks (e.g., biological or social networks), it is unhelpful to assume there are a fixed number of clusters. Furthermore, it may very well be the case that natural communities in the graph vary significantly in size. For these and other reasons, graph partitioning is usually associated, for example, with applications in high performance computing [Schloegel et al., 2003], sparse matrix multiplication [Gupta, 1997], and VSLI layout [Alpert and Kahng, 1995]. In many of these applications, the underlying graph is mesh-like or possesses a very geometric structure, unlike the complex and high-dimensional networks that are often the objective of study in graph clustering.

The subtle differences between graph clustering and graph partitioning are discussed at length in a number of other surveys [Fortunato, 2010, Schaeffer, 2007, Schloegel et al., 2003] and theses [Van Dongen, 2000, Schulz, 2013]. Despite these existing differences, there is a significant overlap between techniques for graph partitioning and tools for community detection, and the problems are frequently presented and studied together in the literature [Bader et al., 2013, Newman, 2013, Wang et al., 2015, Zhou et al., 2017, Schulz et al., 2016]. The broad goal of this thesis is to develop techniques for finding well-connected sets of nodes in a graph under a variety of different assumptions and conditions. Although we focus primarily on graph clustering and community detection, many of the results will directly apply to graph partitioning as well.

2.3 Computational Complexity Terminology

Given our emphasis on optimization-based graph clustering, we briefly review standard terminology on approximations algorithms and hardness results for computational problems. A *decision problem* is a computational problem requiring simply a *yes* or *no* answer. The decision version for any graph clustering optimization problem asks whether, for a fixed objective score β, there exists some clustering $\mathcal{C}$ with objective score less than β. A decision problem is in P if it can be solved in polynomial time. A problem is in NP if it can be *checked* in polynomial time, i.e. given a clustering $\mathcal{C}$ and a score β, does clustering $\mathcal{C}$ have score less than β? A problem is NP-hard if every problem in NP can be reduced to it in polynomial time. NP-complete problems are those which are both NP-hard and are in NP.

There are several more refined notions of complexity that are useful to mention. We will define these specifically for minimization problems, though simple alterations can be made for maximization variants. A *constant*-factor algorithm for a problem is one that runs in polynomial time and outputs a solution within a constant factor of the optimum, i.e., for a minimization problem, a c-approximation algorithm returns a solution that is at most $c \cdot OPT$ where OPT is the optimum score. An algorithm is a polynomial time approximation scheme (PTAS) if, for every fixed $\varepsilon > 0$, there is a polynomial time algorithm (with runtime typically dependent on ε) that returns a $(1+\varepsilon)$-approximation. A problem is said to be APX-hard if there exists some constant $c > 1$ such that it is NP-hard to approximate it to within a factor smaller than c. Hence there is no PTAS unless P = NP. The unique games conjecture is a challenging open problem in computer science presented by Khot [2002] regarding the NP-completeness of certain *unique games* problems. A problem is said to be UG-hard if it is NP-hard, assuming the unique games conjecture is true.

2.4 Graph Clustering Objective Functions

Numerous graph clustering objective functions have been introduced and thoroughly analyzed. Here we consider three categories of objective functions (with examples), each of which strikes a different balance between the internal density and external sparsity of formed clusters.

2.4.1 Cut-to-Size Ratio Objectives

Many objective functions are defined specifically to measure the community structure of a single set $S \subset V$, based on a ratio between $\text{cut}(S)$ and some measure of the set's size. The *sparsity* of a cut $S \subset V$ is defined by

$$\mathbf{scut}(S) = \frac{\text{cut}(S)}{|S|} + \frac{\text{cut}(\bar{S})}{|\bar{S}|} = n \cdot \frac{\text{cut}(S)}{|S||\bar{S}|}. \tag{2.2}$$

Finding the set S with minimum sparsity is known as the *sparsest cut* problem, and is a popular problem in theoretical computer science. Not only is the problem NP-hard, it is known to be UG-hard to approximate to within every constant factor [Chawla et al., 2015]. For many years, the best known approximation ratio was an $O(\log n)$ approximation due to Leighton and Rao [1999]. Later this was improved to $O(\sqrt{\log n})$ by Arora et al. [2009] by rounding a semidefinite programming relaxation. Often it is convenient to work with a scaled version of the objective that measures the total number of edges between S and $\bar{S}$ (i.e. $\text{cut}(S)$), divided by the maximum possible number of edges between the two sets (i.e. $|S||\bar{S}|$). We refer to this as the *scaled sparsity* of S, which we will denote

$$\mathbf{sscut}(S) = \frac{\text{cut}(S)}{|S||\bar{S}|} = \frac{1}{n}\mathbf{scut}(S). \tag{2.3}$$

A number of other objectives are closely related to sparsest cut. The *expansion* of a set is defined by

$$\mathbf{expan}(S) = \frac{\text{cut}(S)}{\min\{|S|, |\bar{S}|\}}, \tag{2.4}$$

and is also frequently studied from the perspective of computational complexity. The degree-weighted versions of these problems are the normalized cut and conductance scores, defined respectively as follows:

$$\mathbf{ncut}(S) = \frac{\text{cut}(S)}{\text{vol}(S)} + \frac{\text{cut}(\bar{S})}{\text{vol}(\bar{S})} \tag{2.5}$$

$$\mathbf{cond}(S) = \frac{\text{cut}(S)}{\min\{\text{vol}(S), \text{vol}(\bar{S})\}}. \tag{2.6}$$

Normalized cut was introduced by Shi and Malik [2000] in the context of image segmentation, while conductance is recognized as one of the most commonly used objectives for community detection [Schaeffer, 2007]. From a theoretical perspective, all of these objectives are treated as being nearly identical. For degree-regular graphs, conductance and expansion are the same up to a constant multiplicative factor, as are sparsest cut and normalized cut. Furthermore, conductance and normalized cut differ by at most a factor two, as do cut expansion and cut sparsity.

Relationship to Multi-cluster Objectives Although the objectives above are defined specifically for a single set S, they are still viewed as global clustering objectives, since identifying a single community S is equivalent to forming a two-clustering of the network: $\mathcal{C} = \{S, \bar{S}\}$. Multi-cluster generalizations of the objectives have also been considered. For example, when Shi and Malik [2000] introduced the normalized cut score, they presented a k-way version of the objective defined by

$$\mathbf{ncut}_k(\mathcal{C}) = \frac{\text{cut}(S_1, \bar{S}_1)}{\text{vol}(S_1)} + \frac{\text{cut}(S_2, \bar{S}_2)}{\text{vol}(S_2)} + \cdots + \frac{\text{cut}(S_k, \bar{S}_k)}{\text{vol}(S_k)} \tag{2.7}$$

for a k-clustering $\mathcal{C} = \{S_1, S_2, \cdots, S_k\}$.

2.4.2 Modularity-Based Objectives

Arguably the most widely applied objective for community detection is the *modularity* score of Newman and Girvan [2004]. Intuitively, a clustering is said to have high modularity if clusters have higher internal edge density than would be expected at *random*. Randomness in this context is defined by a pre-specified null model defining the probability P_{ij} that an edge exists between nodes i, j. Formally, the modularity of a clustering $\mathcal{C}$ is defined by

$$\mathbf{mod}(\mathcal{C}) = \frac{1}{2m} \sum_{i=1}^{n} \sum_{j=1}^{n} (A_{ij} - P_{ij}) \delta_{ij}^{\mathcal{C}}. \tag{2.8}$$

A number of different choices for the modularity null model have been considered in previous work. The most basic is the Bernoulli random graph model in which the expectation of an edge is $P_{ij} = p$ for some fixed $p \in (0, 1)$ and for all pairs of nodes $i \neq j$. However, a much more popular approach is to use the Chung-Lu null model [Chung and Lu, 2002], defined by setting $P_{ij} = d_i d_j / (2m)$. Choosing the latter model ensures that the number of *expected* edges is the same as the number of actual edges in the observed graph G, since $\sum_{i \neq j} P_{ij} = \sum_{i \neq j} A_{ij}$. Furthermore, the degree distribution is preserved. In other words, one can show that $\sum_{j \neq i} P_{ij} = d_i$ for each node i, so the expected degree of node i in the random graph equals the degree of node i in G.

Limitations of Modularity Modularity has been applied extensively and variations of the objective have been introduced for weighted graphs, bipartite graphs, multi-slice networks, and a number of other cases. Despite its widespread use, however, it is known to exhibit several limitations. First of all, even random graphs may exhibit high maximum modularity scores, making it hard to discern when a high modularity score is indicative of meaningful structure [Guimerà et al., 2004]. Modularity is also known to suffer from an inherent resolution limit [Fortunato and Barthélemy, 2007], meaning that it may fail to detect communities that are smaller than a certain size, which depends on the size of the network. Finally, modularity is very challenging to optimize. Not only is the objective NP-hard, Dinh et al. [2016] showed that it is NP-hard to even approximate to within any constant factor. Thus, although many heuristic methods have been introduced, none of them come with provable approximation guarantees.

Generalizations of Modularity Reichardt and Bornholdt introduced a generalization of modularity based on finding minimum energy states of a spin glass model [Reichardt and Bornholdt, 2004, 2006]. For our purposes it is enough to understand that their approach to graph clustering corresponds to minimizing the following Hamiltonian objective function:

$$\textbf{Hamiltonian}(\mathcal{C}) = -\sum_{i \neq j} (A_{ij} - \gamma P_{ij})\, \delta_{ij}^{\mathcal{C}}\,, \tag{2.9}$$

where γ is a tunable clustering resolution parameter. When $\gamma = 1$, minimizing the Hamiltonian is equivalent to maximizing modularity.

Delvenne et al. [2010] introduced another generalization of modularity called the *stability* of a clustering. The stability of a partition measures the probability that a random walker in a graph will end up in the cluster it started in after a random walk of length t. The walk length t is another type of tunable resolution parameter that can be changed to detect different types of clusters in a graph. As t increases, the random walker will "wander" farther from its initial starting point, and the objective will tend to reward the detection of larger clusters in the graph. Delvenne et al. [2010] proved that a linearized version of stability is equivalent to the Hamiltonian objective (2.9) of Reichardt and Bornholdt.

Both the stability objective and the Hamiltonian objective (2.9) generalize modularity and provide a way to overcome the resolution limit by allowing users to set a resolution parameter (γ or

t). However, optimizing these objectives remains very challenging and all known algorithms are heuristics that come with no approximation guarantees.

2.4.3 Cluster Graph Modification Objectives

An idealized definition of a community in a graph is a set of completely connected nodes (i.e. a *clique*), which shares no edges with the rest of the graph. A graph $G = (V, E)$ made up entirely of disjoint cliques is referred to as a *cluster graph.* Based on this idealized notion of community structure, *cluster graph modification* objectives measure the number of edges that need to be changed in a graph in order to convert it into a cluster graph. A variant of the problem was first studied by Ben-Dor et al. [1999] in the context of clustering gene expression patterns. Shamir et al. [2004] later formalized three related objectives: cluster completion, cluster editing, and cluster deletion.

Formal Definitions *Cluster completion* seeks the minimum number of edges to add to a graph to turn it into a cluster graph. This problem can be solved in polynomial time since it amounts to finding connected components in a graph. *Cluster deletion* seeks the minimum number of edges to remove from a graph to turn it into a disjoint union of cliques. This is equivalent to finding a clustering $\mathcal{C}$ in which all clusters are cliques, and the number of edges between these cliques is minimized. Formally, the objective can be written:

$$\begin{array}{ll} \min_{\mathcal{C}} & \sum_{i<j} A_{ij}(1 - \delta_{ij}^{\mathcal{C}}) \\ \text{subject to} & \delta_{ij}^{\mathcal{C}} = 0 \text{ if } (i,j) \notin E \end{array} \tag{2.10}$$

Natanzon [1999] showed that cluster deletion is NP-hard to optimize, while Shamir et al. [2004] proved that it is in fact APX-hard.

Cluster editing allows both the addition and deletion of edges, and was shown to be NP-complete by Shamir et al. [2004]. Formally, the objective is:

$$\min_{\mathcal{C}} \sum_{i<j} A_{ij}(1 - \delta_{ij}^{\mathcal{C}}) + (1 - A_{ij})\delta_{ij}^{\mathcal{C}}. \tag{2.11}$$

For a fixed clustering $\mathcal{C}$, we will use $\mathbf{cedit}(\mathcal{C})$ and $\mathbf{cdel}(\mathcal{C})$ to denote the number of edges that must be changed in the graph, for cluster editing and cluster deletion, respectively, in order for $\mathcal{C}$ to turn into a disjoint set of cliques. For cluster deletion, we assign a score $\mathbf{cdel}(\mathcal{C}) = \infty$ if any cluster in $\mathcal{C}$ is not a clique.

Complexity Results A number of parameterized complexity results have been shown for cluster editing and cluster deletion [Böcker and Damaschke, 2011, Gramm et al., 2003, 2005, Damaschke, 2009]. Parameterized algorithms consider a fixed budget k and seek a *yes* or *no* answer for whether a graph can be turned into a cluster graph using at most k modifications. Böcker and Baumbach [2013] provide an overview of fixed parameter tractability results specifically for cluster editing. Many polynomial time algorithms and hardness results for special classes of graphs have also been presented [Natanzon et al., 1999, Dessmark et al., 2007b, Gao et al., 2013, Bonomo et al., 2015a,b].

Cluster editing is equivalent to unweighted correlation clustering [Bansal et al., 2004], which we discuss in depth in the next section. Many correlation clustering results therefore directly transfer over to cluster editing. In particular, the problem was proved to be APX-hard to optimize [Charikar et al., 2005], and the best approximation factor is slightly better than 2.06 [Chawla et al., 2015]. Cluster deletion can be viewed as a constrained version of correlation clustering. Charikar et al. [2005] showed that a 4-approximation algorithm for unweighted correlation clustering (i.e. cluster editing) can be adapted to obtain a 4-approximation for cluster deletion. Later, van Zuylen and Williamson [2009] proved a 3-approximation for constrained variants of correlation clustering, which directly implies a 3-approximation for cluster deletion.

2.5 Correlation Clustering Background

Correlation clustering is framework for clustering datasets characterized by *positive* and *negative* pairwise relationships between data objects. The problem is typically formalized as a partitioning problem on *signed* graphs. In this context, a negative edge in the graph represents evidence that two nodes should be clustered apart, and a positive edge is evidence that two nodes belong together. Because individual pieces of evidence may conflict with each other, the goal is to find a clustering of the data which *correlates* as much as possible with the evidence. Correlation clustering was formalized and introduced to the theoretical computer science community by Bansal et al. [2004]. Since then, the problem has been studied extensively from a theoretical perspective, and has also been used in a large number of data science applications.

2.5.1 Formal Objectives

The most general instance of correlation clustering is given by a graph $G = (V, W^+, W^-)$ in which every pair of nodes $(i, j) \in V \times V$ possesses two nonnegative weights, $w_{ij}^+ \in W^+$ and $w_{ij}^- \in W^-$. These weights indicate how similar and how dissimilar i and j are, respectively. Often, but not always, only one of these weights is nonzero for each pair (i, j), to indicate that each pair of nodes is either strictly similar or strictly dissimilar. An *agreement* occurs when two similar nodes are clustered together, or two dissimilar nodes are separated. A *disagreement* is defined by two similar nodes that are separated, or two dissimilar nodes that are clustered together.

There are two commonly studied objective functions for correlation clustering: maximizing the weight of agreements and minimizing the weight of disagreements. When minimizing disagreements, placing nodes i and j in the same cluster comes with a penalty of w_{ij}^-, whereas separating them gives a penalty of w_{ij}^+. The objective can then be cast as an integer linear program (ILP):

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{i<j} w_{ij}^+ x_{ij} + w_{ij}^-(1 - x_{ij}) \\ \text{subject to} \quad & x_{ij} \leq x_{ik} + x_{jk} \text{ for all } i, j, k \\ & x_{ij} \in \{0, 1\} \text{ for all } i, j. \end{aligned} \tag{2.12}$$

In this formulation, x_{ij} represents "distance": $x_{ij} = 0$ indicates that nodes i and j are clustered together, while $x_{ij} = 1$ indicates they are separated. Including triangle inequality constraints ($x_{ij} \leq x_{ik} + x_{jk}$) ensures the output will define a valid clustering of the nodes. The first summation in the objective counts all penalties associated with placing nodes together, and the second summation counts penalties from placing nodes in separate clusters. One of the key features of correlation clustering is that the number of clusters to form is not specified in advance. Rather, the appropriate number of clusters to form will arise naturally by optimizing (2.12).

Maximizing agreements can be cast as an ILP that differs from (2.12) by an additive constant. Because of this, both problems are equivalent when optimized exactly. However, from the perspective of approximations, minimizing disagreements is significantly more challenging.

2.5.2 Theoretical Results

The complete unweighted version of correlation clustering considers a signed graph in which every two nodes share either a positive edge or a negative edge. Equivalently, $(w_{ij}^+, w_{ij}^-) \in \{(0,1), (1,0)\}$ for all $(i, j) \in V \times V$). Bansal et al. [2004] gave a PTAS for maximizing agreements and an $O(1)$-approximation for minimizing agreements. The latter result served mainly as an illustration that constant factor approximations are possible. Charikar et al. [2005] shortly thereafter proved that minimizing disagreements is APX-hard, but significantly improved the approximation ratio to 4 by rounding a linear programming relaxation of (2.12). The approximation factor for minimizing disagreements was later lowered to 2.5 by Ailon et al. [2008], who also gave an extremely fast and elegant 3-approximation based on a technique called *pivoting*. The best known approximation factor for minimizing disagreements is slightly less than 2.06, due to Chawla et al. [2015].

In graphs with arbitrary weights, several groups proved simultaneously that minimizing disagreements can be approximated to within $O(\log n)$ by rounding an LP relaxation [Charikar et al., 2005, Emanuel and Fiat, 2003, Demaine and Immorlica, 2003]. For maximizing agreements, a simple half-approximation is achieved by either placing all nodes in one cluster or separating each node into its own cluster. The best known approximation for maximizing agreements is 0.7666, due to Swamy

[2004], which is only slightly better than the 0.7664-approximation due to Charikar et al. [2005]. Both approximations are based on rounding a semidefinite programming relaxation.

Minimizing disagreements in arbitrarily weighted graphs is known to be equivalent to minimum multicut [Emanuel and Fiat, 2003, Demaine and Immorlica, 2003, Demaine et al., 2006]. This holds also in the case where all nodes share either a positive edge, a negative edge, or no edge. This equivalence immediately implies that in general, minimizing disagreements is UG-hard to approximate to within any constant factor [Chawla et al., 2015]. Furthermore, Emanuel and Fiat [2003] showed that the linear programming relaxation of the problem has an $\Omega(\log n)$ integrality gap, proving that the $O(\log n)$ approximation via LP-rounding is tight.

Numerous other theoretical results have been proven for special variants of correlation clustering. Without attempting to provide an exhaustive summary, we note that this includes results for bipartite graphs [Asteris et al., 2016], edge-colored graphs [Bonchi et al., 2015, Anava et al., 2015], and hypergraphs [Li et al., 2017, Gleich et al., 2018, Fukunaga, 2018]). Previous research has also addressed restrictions on the number of clusters [Giotis and Guruswami, 2006, Coleman et al., 2008], alternative objective functions [Puleo and Milenkovic, 2018, Charikar et al., 2017], special weighted variants [Ailon et al., 2008, Puleo and Milenkovic, 2015, Veldt et al., 2017, 2018b], and streaming algorithms [Ahn et al., 2015, Bhaskara et al., 2018].

2.5.3 Applications and Scalable Algorithms

Correlation clustering has been used in numerous application domains including image segmentation [Kim et al., 2011, Beier et al., 2014], bioinformatics [Bhattacharya and De, 2008, Hou et al., 2016], cross-lingual link detection [Van Gael and Zhu, 2007], and community detection [Sharma and Singh, 2016, Wang et al., 2013]. However, many of the theoretical approximation algorithms that have been developed do not apply particularly well in these cases. To begin with, many of the best theoretical results apply only to variants of the problem that arise infrequently in practice (e.g. complete, unweighted graphs). Another challenge is that the linear and semidefinite programming relaxations of correlation clustering have high memory requirements and are hard to solve in practice, even if they are theoretically polynomial-time solvable. Therefore, a common focus in the literature has been to present scalable techniques for correlation clustering.

Scalable algorithms for correlation clustering take a number of different forms. Chierichetti et al. [2014], and later Pan et al. [2015] developed parallel versions of the pivoting algorithm of Ailon et al. [2008]. These come with a priori approximation guarantees, but only apply to the complete unweighted case. At the other end of the spectrum, many fast heuristic techniques have been introduced for the general weighted case [Bagon and Galun, 2011, Wang et al., 2013, Beier et al., 2014, 2015]. These have been shown to perform well in certain applications, but do not come with any sort of approximation guarantee. Much of the literature on scalable algorithms for correlation clustering is focused specifically on sparse problems in which the vast majority of node pairs do not share an edge. In this context, weighted correlation clustering is often referred to as the *multicut partitioning* problem. A number of techniques have been designed for computing lower bounds for the multicut objective in practice, which provide a way to obtain a posteriori approximation guarantees [Lange et al., 2018, Nowozin and Jegelka, 2009, Swoboda and Andres, 2017]. However, these do not solve the canonical LP relaxation of correlation clustering, and thus do not enable one to implement the best a priori approximation algorithms.

2.6 Graph Clustering Equivalence Results

We call two objective functions *equivalent* if they produce the same output when solved optimally, even if they are not the same from the perspective of approximations. Several graph clustering objectives have been shown to be equivalent or at least closely related. Newman [2016] demonstrated that maximizing modularity with a resolution parameter is equivalent to maximizing a log-likelihood function for the degree-corrected stochastic block model. Several authors independently showed that a normalized version of modularity with a restricted number of clusters is known to be equivalent to the k-way generalization of normalized cut (2.7) [Bolla, 2011, Yu and Ding, 2010, Wang et al., 2015]. Delvenne et al. [2010] also observed a relationship between modularity and normalized cut, which can both be viewed as special cases of the *stability* clustering objective.

As noted in Section 2.4.3, the objective of minimizing disagreements for correlation clustering in unweighted graphs is equivalent to cluster editing. The objective functions are in fact identical: counting the number of edges to add or delete in a graph in order to turn it into a cluster graph is exactly the same as counting the number of negative edge mistakes and positive edge mistakes in a clustering of a signed graph. Similarly, cluster deletion can be viewed as a constrained version of correlation clustering, in which one is prohibited from making mistakes at negative edges. Finally, Agarwal and Kempe [2008] observed that the LP-rounding technique of Charikar et al. [2005] for correlation clustering can be adapted and applied to obtain bounds and approximate solutions for modularity. Although Agarwal and Kempe did not focus on proving equivalence results, this result provides a first glimpse at a relationship between the two objectives.

3. THE LAMBDACC GRAPH CLUSTERING FRAMEWORK

3.1 Chapter Overview

This chapter introduces the LambdaCC framework for graph clustering. This framework models graph clustering as a specially weighted version of correlation clustering, in which positive and negative edges are weighted with respect to a resolution parameter λ. When the LambdaCC objective function is optimized, the chosen value of λ implicitly controls the trade-off between forming clusters with a high internal edge density, and forming clusters with a low between-cluster edge density.

In addition to giving practitioners the ability to strike their own desired balance between internal density and external sparsity, LambdaCC generalizes a number of previously studied objectives for graph clustering. These objectives include modularity, cluster deletion, and sparsest cut, all of which can be captured as special cases of LambdaCC for a specific choice of λ. In this way LambdaCC serves as a flexible and versatile objective function, which is able to interpolate between previous objectives and highlight community structure at different resolutions in the same network. At the end of the chapter we show two sets of experiments that highlight the benefits of the versatility and generality of LambdaCC in practice.

The results of this chapter are based on a shorter presentation of the LambdaCC framework published at the 2018 World Wide Web Conference [Veldt et al., 2018b].

3.2 Graph Clustering as Correlation Clustering

Correlation clustering is often presented as a framework for partitioning *signed* graphs. On a surface level, this may appear significantly different from detecting communities in unsigned and unweighted networks. However, these problems share several deep connections. A broader and more helpful view of correlation clustering is that it is a framework for partitioning a dataset characterized by pairwise pieces of *advice* or *evidence* about how to cluster a dataset. It is typically impossible to satisfy all the advice at once, since the evidence provided is often inconsistent. Thus, the goal is to find a clustering that agrees with the advice as much as possible.

Graph clustering fits naturally within the framework of clustering based on pairwise advice. In a good clustering, edges should be dense inside clusters, and sparse between them. The existence of an edge between two nodes, while it does not guarantee they will be clustered together, provides some evidence that the two nodes should be clustered together. Similarly, the absence of an edge does not guarantee two nodes will be separated, but this provides at least some indication that they may not belong together. In this way, edges and non-edges in a graph can be interpreted as localized advice or evidence for whether or not two nodes should be clustered together.

Given this view, a natural approach to graph clustering is to take a graph and convert it into an instance of correlation clustering by introducing positive and negative edges. If we assign a unit weight to every edge in the resulting signed graph, performing correlation clustering on the signed graph is exactly equivalent to solving cluster editing on the unsigned input graph [Shamir et al., 2004]. This equivalence between cluster editing and unweighted correlation clustering [Bansal et al., 2004] has long been known. However, setting all weights equal to one will not be the best choice in all contexts. In some cases it may make more sense to follow the two different types of advice to a differing degree. To address this, we formalize the LambdaCC framework, which converts an unsigned network into a specially weighted version of correlation clustering in which positive and negative edges are weighted unevenly, based on the value of a parameter λ. We will show that tuning the value of this parameter enables practitioners to strike their own desired balance between internal density and external sparsity.

3.3 The LambdaCC Construction

Recall that a general instance of correlation clustering is given by a set of non-negative weights (w_{ij}^+, w_{ij}^-) for each $(i,j) \in V \times V$ where V is a set of nodes. A clustering $\mathcal{C}$ of the nodes has the following weight of disagreements:

$$\textbf{CC-Disagree}(\mathcal{C}) = \sum_{i<j} w_{ij}^+(1 - \delta_{ij}^{\mathcal{C}}) + w_{ij}^-\delta_{ij}^{\mathcal{C}} \,. \tag{3.1}$$

Let $G = (V, E)$ be an unsigned graph that we wish to cluster. The LambdaCC framework converts G into an instance of correlation clustering $G' = (V, W^+, W^-)$ on the same set of nodes, V, by assigning weights (w_{ij}^+, w_{ij}^-) based on the edge structure of G. Solving the resulting correlation clustering problem induces a clustering of the original graph G. We will present two variants: *standard* LambdaCC defines weights based only on the edges E and a resolution parameter $\lambda \in (0,1)$. The *degree-weighted* version also takes into account the degree d_v for each node $v \in V$. The conversion from G to G' is illustrated in Figure 3.1, and explained in detail below.

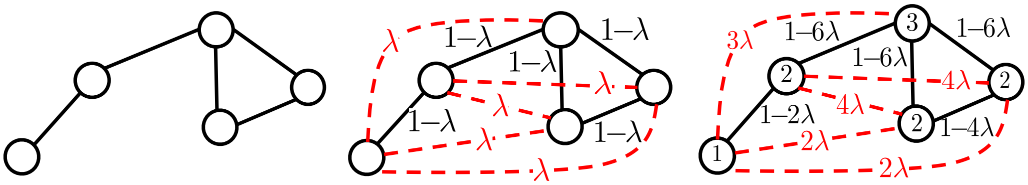


Figure 3.1.: We convert a toy graph (left) into a signed graph for standard (middle) and degree-weighted (right) LambdaCC. Dashed red lines indicate negative edges. Partitioning the signed graph via correlation clustering induces a clustering on the original unsigned graph.

Standard LambdaCC Standard LambdaCC converts each edge in G into a positive edge in a signed graph G' with weight $(1-\lambda)$. Each non-edge in G' is mapped to a negative edge of weight λ in G'. Equivalently, we assign correlation clustering weights as follows:

$$(w_{ij}^{+}, w_{ij}^{-}) = \begin{cases} (1-\lambda, 0) & \text{if } (i,j) \in E \\ (0, \lambda) & \text{if } (i,j) \notin E\,. \end{cases} \tag{3.2}$$

The correlation clustering objective (3.1) for the above weighted case can be expressed as a graph clustering objective on the original graph $G = (V, E)$:

$$\textbf{LamCC}(\mathcal{C}) = \sum_{(i,j)\in E} (1-\lambda)(1-\delta_{ij}^{\mathcal{C}}) + \sum_{(i,j)\notin E} \lambda \delta_{ij}^{\mathcal{C}}. \tag{3.3}$$

Degree-Weighted LambdaCC For degree-weighted LambdaCC, we first consider the case $\lambda \in (0, 1/d_{max}^2]$ where d_{max} is the maximum node degree in $G = (V, E)$. As before, non-edges in G map to negative edges in a signed graph G', and edges in G map to positive edges in a signed graph G'. This time, the edge weight for $(i,j) \in E$ is given by $\lambda d_i d_j$, and the weight for an edge $(i,j) \notin E$ is $(1-\lambda d_i d_j)$ (see the rightmost picture in Figure 3.1). Since $\lambda \leq 1/d_{max}^2$, we know that $1 - \lambda d_i d_j \geq 0$ for all $(i,j) \in E$.

In some cases it will be useful to consider values of $\lambda > 1/d_{max}^2$, in which case it is possible for an edge in G to map to a negative edge in G'. In general, the weights for degree-weighted LambdaCC are given by:

$$(w_{ij}^{+}, w_{ij}^{-}) = \begin{cases} (1-\lambda d_i d_j, 0) & \text{if } (i,j) \in E \text{ and } 1 \geq \lambda d_i d_j \\ (0, \lambda d_i d_j - 1) & \text{if } (i,j) \in E \text{ and } 1 < \lambda d_i d_j \\ (0, \lambda d_i d_j) & \text{if } (i,j) \notin E\,. \end{cases} \tag{3.4}$$

Inserting these weights into the correlation clustering objective (3.1) yields the degree-weighted LambdaCC objective for graph clustering:

$$\textbf{DW-LamCC}(\mathcal{C}) = \sum_{\substack{(i,j)\in E \\ 1 \geq \lambda d_i d_j}} (1-\lambda d_i d_j)(1-\delta_{ij}^{\mathcal{C}}) + \sum_{\substack{(i,j)\in E \\ 1 < \lambda d_i d_j}} (\lambda d_i d_j - 1)\delta_{ij}^{\mathcal{C}} + \sum_{(i,j)\notin E} \lambda d_i d_j \delta_{ij}^{\mathcal{C}}\,. \tag{3.5}$$

3.3.1 Alternative LambdaCC Construction

The standard LambdaCC objective (3.3) has a straightforward interpretation and can be written cleanly in terms of edges and non-edges in G. However, the degree-weighted objective (3.5) is complicated by the fact that positive edges in G can map to either positive *or* negative edges in G'. Here we present an alternative LambdaCC construction which differs from these objectives by only an additive constant. The alternative LambdaCC construction is slightly more general, and in some cases will be easier to work with when proving results about the LambdaCC framework.

The edge weights given in (3.2) and (3.4) are chosen specifically so that every two nodes in a signed graph have either a strictly positive or a strictly negative relationship. For the alternative construction, we allow some nodes to possess both a positive and a negative edge with nonzero weight. Formally, start by defining a node weight ω_v for each node $v \in V$, and for each $(i,j) \in V \times V$, define edge weights

$$(w_{ij}^{+}, w_{ij}^{-}) = (A_{ij}, \lambda \omega_i \omega_j),$$

where $A_{ij} \in \{0,1\}$ is the ij entry of the adjacency matrix for $G = (V,E)$. In other words, the alternative LambdaCC construction maintains the original edges in $G = (V,E)$ as positive edges with weight 1. It then introduces a weighted negative edge between *each* pair of nodes. The weight of disagreements for the resulting correlation clustering problem defines the alternative objective:

$$\textbf{AltLamCC}(\mathcal{C}) = \sum_{i<j} A_{ij}(1-\delta_{ij}^{\mathcal{C}}) + \lambda \omega_i \omega_j \delta_{ij}^{\mathcal{C}}. \tag{3.6}$$

Objective (3.6) is general, succinct, and in some cases will be slightly easier to reason about than the original LambdaCC construction. Choosing $\omega_v = 1$ for all $v \in V$ leads to an objective that differs from standard LambdaCC (3.3) by a positive additive constant:

$$\begin{aligned}
\textbf{S-AltLamCC}(\mathcal{C}) &= \sum_{i<j} A_{ij}(1-\delta_{ij}^{\mathcal{C}}) + \lambda\delta_{ij}^{\mathcal{C}} && (3.7)\\
&= \sum_{(i,j)\notin E} \lambda\delta_{ij}^{\mathcal{C}} + \sum_{(i,j)\in E} (1-\delta_{ij}^{\mathcal{C}}) + \lambda\delta_{ij}^{\mathcal{C}} && (3.8)\\
&= \sum_{(i,j)\notin E} \lambda\delta_{ij}^{\mathcal{C}} + \sum_{(i,j)\in E} (1-\lambda)(1-\delta_{ij}^{\mathcal{C}}) + \lambda && (3.9)\\
&= \left(\sum_{(i,j)\notin E} \lambda\delta_{ij}^{\mathcal{C}} + \sum_{(i,j)\in E} (1-\lambda)(1-\delta_{ij}^{\mathcal{C}})\right) + \lambda|E| && (3.10)\\
&= \textbf{LamCC}(\mathcal{C}) + \lambda|E|. && (3.11)
\end{aligned}$$

Similarly, setting $\omega_v = d_v$ for each $v \in V$ produces an objective that differs from the degree-weighted objective (3.5) by a positive added constant, and can be written succinctly as:

$$\textbf{DW-AltLamCC}(\mathcal{C}) = \sum_{i<j} A_{ij}(1-\delta_{ij}^{\mathcal{C}}) + \lambda d_i d_j \delta_{ij}^{\mathcal{C}}. \tag{3.12}$$

The differences between original LambdaCC objectives (3.3) and (3.4) and alternative objectives (3.7) and (3.12) are subtle, but important. What is most important is that the original and alternative objectives for LambdaCC differ by at most a positive constant, so they will have the same set of optimal solutions. Objectives (3.7) and (3.12) in some cases are slightly easier to work when when proving results about the optimal solutions to the LambdaCC objectives. However, in practical applications and approximation algorithms, we will apply the original LambdaCC construction in which every pair of nodes has either a strictly positive or strictly negative relationship. In any cases where we use the alternative construction and objective functions, we will explicitly use the term *alternative.*

Note that the original versions of the LambdaCC objective are harder to approximate than the alternative LambdaCC objectives. We prove this for the standard version; the same approach yields an analogous result for degree-weighted LambdaCC.

Theorem 3.3.1 *Let $\mathcal{C}^*$ be the optimal clustering for the standard LambdaCC objective* (3.3)*. If $\hat{\mathcal{C}}$ is a p-approximation for the original objective* (3.3) *(for some $p \geq 1$), then it is also a p-approximation for the alternative LambdaCC objective* (3.7)*.*

Proof Clustering $\mathcal{C}^*$ is optimal for both versions of the objective. Notice from (3.11) that

$$\textbf{S-AltLamCC}(\mathcal{C}) = \textbf{LamCC}(\mathcal{C}) + \lambda|E| \text{ for any } \mathcal{C}$$

and we are assuming that $\textbf{LamCC}(\hat{\mathcal{C}}) \leq p\,\textbf{LamCC}(\mathcal{C}^*)$. Thus:

$$\begin{aligned}
\textbf{S-AltLamCC}(\hat{\mathcal{C}}) &= \textbf{LamCC}(\hat{\mathcal{C}}) + \lambda|E| \\
&\leq p\,\textbf{LamCC}(\mathcal{C}^*) + \lambda|E| \\
&\leq p(\textbf{LamCC}(\mathcal{C}^*) + \lambda|E|) \\
&\leq p(\textbf{S-AltLamCC}(\mathcal{C}^*)).
\end{aligned}$$

■

3.3.2 Equivalent Formulations of the LambdaCC Objective

The different variants of the LambdaCC objective function can also be expressed directly in terms of cuts, volumes, and set sizes in the original graph G. We will prove this using the alternative LambdaCC objectives presented in the previous section. We begin with a simple example for a bipartition (i.e., a two-clustering) of a graph.

Theorem 3.3.2 *For a two-clustering $\mathcal{C} = \{S, \bar{S}\}$, the standard (respectively, degree-weighted) version of the alternative LambdaCC objective given in* (3.7) *(respectively, given in* (3.12)*) for $\mathcal{C}$ can be written as follows:*

$$\textbf{S-AltLamCC}(\mathcal{C}) = \text{cut}(S) - \lambda|S||\bar{S}| + c_1 \tag{3.13}$$

$$\textbf{DW-AltLamCC}(\mathcal{C}) = \text{cut}(S) - \lambda\,\text{vol}(S)\,\text{vol}(\bar{S}) + c_2 \tag{3.14}$$

where $c_1 = \lambda\binom{n}{2}$ and $c_2 = \lambda\,\text{vol}(V)^2 - \lambda\sum_{i\in V} d_i^2$ are constant with respect to a fixed graph and parameter λ.

Proof For the alternative construction of LambdaCC, positive edges always have weight 1. Therefore, the weight of positive edge mistakes for both the standard and degree-weighted objectives is just cut(S), the number of edges crossing from S to $\bar{S}$. To compute the negative edge mistakes, we start with the weight of all negative edges, and subtract the weight of edges from S to $\bar{S}$, since these are the negative edges where a mistake was *not* made:

$$\begin{aligned}
\text{(Weight of negative edge mistakes)} &= \text{(Total weight of negative edges)} \\
&\quad - \text{(weight of negative edges between } S \text{ and } \bar{S}) \\
&= \sum_{i<j} \lambda\omega_i\omega_j - \sum_{i\in S}\sum_{j\in\bar{S}} \lambda\omega_i\omega_j \\
&= \lambda\left[\sum_{i\in V}\sum_{j\in V}\omega_i\omega_j - \sum_{i\in V}\omega_i^2 - \sum_{i\in S}\sum_{j\in\bar{S}}\omega_i\omega_j\right]
\end{aligned}$$

Inserting $\omega_i = 1$ for the standard case, and $\omega_i = d_i$ for degree-weighted, we see

$$\text{(Weight of negative mistakes for degree-weighted LamCC)} = \lambda \left[\operatorname{vol}(V)^2 - \operatorname{vol}(S)\operatorname{vol}(\bar{S}) - \sum_{i \in V} d_i^2 \right]$$

$$\text{(Weight of negative mistakes for standard LamCC)} = \lambda \left[\binom{n}{2} - |S||\bar{S}| \right]$$

Adding together the weight of positive edge mistakes ($\operatorname{cut}(S)$) and the weight of negative mistakes in each case, yields (3.13) and (3.14) ■

We can quickly generalize this for clusterings with an arbitrary number of clusters.

Theorem 3.3.3 *Let $\mathcal{C} = \{S_1, S_2, \dots, S_k\}$ be a k-clustering of $G = (V, E)$. The standard and degree-weighted versions of the alternative LambdaCC objective for $\mathcal{C}$ can be written as follows:*

$$\textbf{S-AltLamCC}(\mathcal{C}) = \frac{1}{2}\sum_{i=1}^{k} \operatorname{cut}(S_i) - \frac{\lambda}{2}\sum_{i=1}^{k} |S_i||\bar{S}_i| + c_1 \tag{3.15}$$

$$\textbf{DW-AltLamCC}(\mathcal{C}) = \frac{1}{2}\sum_{i=1}^{k} \operatorname{cut}(S_i) - \frac{\lambda}{2}\sum_{i=1}^{k} \operatorname{vol}(S_i)\operatorname{vol}(\bar{S}_i) + c_2 \tag{3.16}$$

where $c_1 = \lambda\binom{n}{2}$ and $c_2 = \lambda \operatorname{vol}(V)^2 - \lambda \sum_{i \in V} d_i^2$ are constant with respect to a fixed graph and parameter λ.

Proof If we consider a single cluster S_i, the number of positive edge mistakes associated with S_i is $\operatorname{cut}(S_i) = \operatorname{cut}(S_i, \bar{S}_i)$. The *total* weight of positive edge mistakes across all clusters is therefore:

$$\text{(Weight of positive edge mistakes)} = \frac{1}{2}\sum_{i=1}^{k} \operatorname{cut}(S_i)\,.$$

Dividing by two is necessary in order to avoid counting each positive mistake twice, since each of these edges is incident to two of the k clusters. The weight of negative mistakes can be again computed by adding up the weight of *all* negative edges, and then subtracting the weight of negative edges crossing between two clusters. The weight of all negative edges is:

$$\text{(Weight of all negative edges)} = \sum_{u<v} \lambda \omega_u \omega_v = \lambda \left[\sum_{u \in V}\sum_{v \in V} \omega_u \omega_v - \sum_{u \in V} \omega_u^2 \right],$$

which evaluates to the constants c_1 and c_2 when $\omega_v = 1$ and $\omega_v = d_v$ respectively. To get the weight of negative edges where a mistake was *not* made, we consider the negative edges leaving each cluster S_i, and then we sum these up across all k clusters, and divide by 2 in order to avoid double counting. Thus,

$$\text{(Weight of negative edges crossing between clusters)} = \frac{1}{2}\sum_{i=1}^{k}\sum_{u \in S_i}\sum_{v \in \bar{S}_i} \lambda \omega_u \omega_v.$$

This equals $\frac{1}{2}\sum_{i=1}^{k} |S_i||\bar{S}_i|$ when $\omega_v = 1$ and $\frac{1}{2}\sum_{i=1}^{k} \operatorname{vol}(S_i)\operatorname{vol}(\bar{S}_i)$ when $\omega_v = d_v$. Putting all the terms together yields the desired results for both standard and degree-weighted LambdaCC. ■

We finish the section with a corollary that applies to the original version of the standard LambdaCC objective in which positive edges have weight $(1-\lambda)$ and negative edges have weight λ.

Corollary 3.3.4 *Let $G = (V, E)$ be a graph and fix a parameter λ.*

1. *Let $S \subset V$ be a set of nodes and $\mathcal{C} = \{S, \bar{S}\}$ define a two-clustering of G. The standard LambdaCC objective score* (3.3) *for $\mathcal{C}$ can be re-written:*

$$\mathbf{LamCC}(\mathcal{C}) = \text{cut}(S, \bar{S}) - \lambda|S||\bar{S}| + \lambda\binom{n}{2} - \lambda|E|\,.$$

2. *The standard LambdaCC objective* (3.3)*, which is optimized over clusterings $\mathcal{C}$ with arbitrarily many clusters, can be written:*

$$\min_{\mathcal{C}} \frac{1}{2}\sum_{S\in\mathcal{C}} \text{cut}(S) - \frac{\lambda}{2}\sum_{S\in\mathcal{C}} |S||\bar{S}| + \lambda\binom{n}{2} - \lambda|E|\,. \tag{3.17}$$

Proof This follows directly from the results of Theorems (3.3.2) and (3.3.3), combined with the fact that $\mathbf{S\text{-}AltLamCC}(\mathcal{C}) = \mathbf{LamCC}(\mathcal{C}) + \lambda|E|$ for any clustering $\mathcal{C}$. ■

3.4 Theoretical Equivalence Results

Having motivated LambdaCC from first principles, we now turn to several surprising connections between this new clustering objective and techniques that have been studied in past work. Throughout the section, the term *equivalent* refers to objective functions that have the same set of optimal solutions, even if they are different from the perspective of approximations.

3.4.1 Equivalence with Hamiltonian

Despite a significant difference in approach and interpretation, the clustering that minimizes disagreements is the same clustering that minimizes the Hamiltonian objective (2.9), for a standard choice of parameters.

Theorem 3.4.1 *Let $m = |E|$. If we define the graph null model for the Hamiltonian objective to be $P_{ij} = \omega_i\omega_j/(2m)$ and set its resolution parameter to $\gamma = 2m\lambda$, then the clustering $\mathcal{C}$ which minimizes the Hamiltonian, also minimizes the LambdaCC objective.*

Proof We begin with the alternative LambdaCC objective in its general form (3.6) and perform a few steps of algebra:

$$\begin{aligned}\textbf{AltLamCC}(\mathcal{C}) &= \sum_{i<j} A_{ij}(1-\delta_{ij}^{\mathcal{C}}) + \lambda\omega_i\omega_j\delta_{ij}^{\mathcal{C}} \\ &= \sum_{i<j} A_{ij} + \sum_{i<j} -A_{ij}\delta_{ij}^{\mathcal{C}} + \lambda\omega_i\omega_j\delta_{ij}^{\mathcal{C}} \\ &= \frac{|V|(|V|-1)}{2} - \sum_{i<j}(A_{ij}-\lambda\omega_i\omega_j)\delta_{ij}^{\mathcal{C}}\end{aligned}$$

Choosing the null model and resolution parameter given in the theorem statement, we obtain the equivalence:

$$\textbf{AltLamCC}(\mathcal{C}) = \frac{|V|(|V|-1)}{2} + \frac{1}{2}\textbf{Hamiltonian}(\mathcal{C})\,. \tag{3.18}$$

■

The choice $P_{ij} = \omega_i\omega_j/(2m)$ is reminiscent of the Chung-Lu null model most commonly used for modularity and the Hamiltonian objectives. Thus we see that degree-weighted LambdaCC with $\lambda = 1/(2m)$ is equivalent with modularity.

3.4.2 Connection to Sparsest Cut

While degree-weighted LambdaCC is more closely related to modularity and the Hamiltonian, the standard LambdaCC objective (3.7) can be viewed as a generalization of sparsest cut (2.2).

In Corollary 3.3.4 we showed that for a two-clustering $\mathcal{C} = \{S, \bar{S}\}$, the standard LambdaCC objective score is

$$\text{cut}(S,\bar{S}) - \lambda|S||\bar{S}| + c\,, \tag{3.19}$$

where $c = \lambda\binom{n}{2} - \lambda|E|$ is constant with respect to G and λ. Note that if we minimize (3.19) over all two-clusterings, we solve the decision version of the minimum *scaled* sparsity objective (2.3). A few steps of algebra confirm that there is some set $S \subseteq V$ with $\textbf{sscut}(S) = \text{cut}(S)/(|S||\bar{S}|) < \lambda$ if and only if (3.19) is less than c.

Since LambdaCC is a special case of correlation clustering, we do not place a restriction on the number of clusters that are formed. The second part of Corollary 3.3.4 shows that in general, when we minimize over clusterings $\mathcal{C}$ with arbitrarily many clusterings, the standard LambdaCC objective can be written as follows:

$$\min\ \frac{1}{2}\sum_{S\in\mathcal{C}}\text{cut}(S) - \frac{\lambda}{2}\sum_{S\in\mathcal{C}}|S||\bar{S}| + c\,. \tag{3.20}$$

In this case, optimally solving objective (3.20) will tell us whether we can find a clustering $\mathcal{C} = \{S_1, S_2, \dots, S_k\}$ with some number of clusters k such that

$$\frac{\sum_{i=1}^{k} \operatorname{cut}(S_i, \bar{S}_i)}{\sum_{j=1}^{k} |S_j||\bar{S}_j|} < \lambda.$$

Hence LambdaCC can be viewed as a multi-cluster generalization of the decision version of sparsest cut. The following theorem proves an even deeper connection between sparsest cut and standard LambdaCC. Using degree-weighted LambdaCC yields an analogous result for normalized cut.

Theorem 3.4.2 *Let λ^* be the minimum scaled sparsity for a graph G.*

(a) For all $\lambda > \lambda^*$, the optimal solution to (3.20) partitions G into two or more clusters, each of which has scaled sparsity bounded above by λ. Furthermore, there exists some $\lambda' > \lambda^*$ such that the optimal clustering for LambdaCC is the minimum sparsest cut partition.

(b) For $\lambda \leq \lambda^*$, it is optimal to place all nodes into a single cluster.

Proof **Statement (a)** Let S^* be some optimal sparsest cut-inducing set in G, i.e.,

$$\textbf{sscut}(S^*) = \frac{\operatorname{cut}(S^*)}{|S^*||\bar{S}^*|} = \lambda^*.$$

The LambdaCC objective score for clustering $\mathcal{C} = \{S^*, \bar{S}^*\}$ is

$$\operatorname{cut}(S^*) - \lambda|S^*||\bar{S}^*| + \lambda\binom{n}{2} - \lambda|E|\,. \tag{3.21}$$

When minimizing objective (3.20), we can always obtain a score of $\lambda\binom{n}{2} - \lambda|E|$ by placing all nodes into a single cluster. Note however that the score of clustering $\{S^*, \bar{S}^*\}$ in expression (3.21) is strictly less than $\lambda\binom{n}{2} - \lambda|E|$ for all $\lambda > \lambda^*$. Even if $\{S^*, \bar{S}^*\}$ is not optimal, this means that when $\lambda > \lambda^*$, we can do strictly better than placing all nodes into one cluster. In this case let $\mathcal{C}^*$ be the optimal LambdaCC clustering and consider two of its clusters: S_i and S_j. The weight of disagreements between S_i and S_j is equal to the number of positive edges between them times the weight of a positive edge: $(1-\lambda)\operatorname{cut}(S_i, S_j)$. Should we form a new clustering by merging S_i and S_j, these positive disagreements will disappear; in turn, we would introduce $\lambda|S_i||S_j| - \lambda\operatorname{cut}(S_i, S_j)$ new mistakes, as this is the weight of negative edges between the clusters. Because we assumed $\mathcal{C}^*$ is optimal, we know that we cannot decrease the objective by merging two of the clusters. This implies that

$$(1-\lambda)\operatorname{cut}(S_i, S_j) - (\lambda|S_i||S_j| - \lambda\operatorname{cut}(S_i, S_j)) = \operatorname{cut}(S_i, S_j) - \lambda|S_i||S_j| \leq 0\,.$$

Given this, we fix an arbitrary cluster S_i and perform a sum over all other clusters to see that

$$\sum_{j\neq i} \operatorname{cut}(S_i, S_j) - \sum_{j\neq i} \lambda|S_i||S_j| \leq 0$$

$$\implies \mathrm{cut}(S_i, \bar{S}_i) - \lambda |S_i||\bar{S}_i| \leq 0 \implies \frac{\mathrm{cut}(S_i, \bar{S}_i)}{|S_i||\bar{S}_i|} \leq \lambda \,,$$

proving the desired upper bound on the scaled sparsity of each cluster S_i.

Since G is a finite graph, there are a finite number of scaled sparsity scores that can be induced by a subset of V. Let $\tilde{\lambda}$ be the second-smallest scaled sparsity score achieved, so $\tilde{\lambda} > \lambda^*$. If we set $\lambda' = (\lambda^* + \tilde{\lambda})/2$, then the optimal LambdaCC clustering produces at least two clusters, since $\lambda' > \lambda^*$, and each cluster has scaled sparsity at most $\lambda' < \tilde{\lambda}$. By our selection of $\tilde{\lambda}$, all clusters returned must have scaled sparsity exactly equal to λ^*, which is only possible if the clustering returned has two clusters. Hence this clustering is a minimum sparsity partition of the network.

Statement (b) If $\lambda < \lambda^*$, forming a single cluster must be optimal, otherwise we could invoke Statement (a) to assert the existence of some nontrivial cluster with scaled sparsity less than or equal to $\lambda < \lambda^*$, contradicting the minimality of λ^*. If $\lambda = \lambda^*$, forming a single cluster or using the clustering $\mathcal{C} = \{S^*, \bar{S}^*\}$ yield the same objective score, which is again optimal for the same reason. ■

3.4.3 Connection to Cluster Modification Objectives

For large λ, standard LambdaCC is closely related to cluster modification problems. Recall that the cluster editing objective, which is equivalent to unweighted correlation clustering, is given by

$$\textbf{cedit}(\mathcal{C}) = \sum_{i<j} A_{ij}(1 - \delta_{ij}^{\mathcal{C}}) + (1 - A_{ij})\delta_{ij}^{\mathcal{C}}. \tag{3.22}$$

We can generalize this by including a weight α on the term which penalizes placing two non-adjacent nodes in the same cluster:

$$\textbf{cedit}_\alpha(\mathcal{C}) = \sum_{i<j} A_{ij}(1 - \delta_{ij}^{\mathcal{C}}) + \alpha(1 - A_{ij})\delta_{ij}^{\mathcal{C}}. \tag{3.23}$$

If $\alpha \to \infty$, the penalty on clustering non-adjacent nodes together becomes prohibitively expensive, and the problem becomes equivalent to cluster deletion. If we substitute $\alpha = \lambda/(1-\lambda)$, this differs from standard LambdaCC (3.3) only by a constant factor:

$$\textbf{cedit}_{\lambda/(1-\lambda)}(\mathcal{C}) = \sum_{(i,j)\in E} (1 - \delta_{ij}^{\mathcal{C}}) + \sum_{(i,j)\notin E} \frac{\lambda}{1-\lambda}\delta_{ij}^{\mathcal{C}} = \frac{1}{(1-\lambda)} \textbf{LamCC}(\mathcal{C})\,.$$

When $\alpha > 1$ (i.e., $\lambda > 1/2$), putting non-adjacent nodes together will be more expensive than cutting positive edges, so we would expect that the clustering which optimizes the LambdaCC objective will separate G into dense clusters that are "nearly" cliques. We formalize this with a simple theorem and corollary.

Theorem 3.4.3 *If $\mathcal{C}$ minimizes the standard LambdaCC objective for the unsigned graph $G = (V, E)$, then the edge density of every cluster in $\mathcal{C}$ is at least λ.*

Proof Take a cluster $S \in \mathcal{C}$ and consider what would happen if we broke apart S so that each node in S were instead placed into its own singleton cluster. This means we are now making mistakes at every positive edge previously in S, which increases the weight of disagreements by $(1-\lambda)|E_S|$. On the other hand, there are no longer negative mistakes between nodes in S, so the LambdaCC objective would simultaneously decrease by $\lambda\left(\binom{|S|}{2} - |E_S|\right)$. The total change in the objective made by breaking S into singletons is

$$(1-\lambda)|E_S| - \lambda\left(\binom{|S|}{2} - |E_S|\right) = |E_S| - \lambda\binom{|S|}{2},$$

which must be nonnegative, since $\mathcal{C}$ is optimal, so

$$|E_S| - \lambda\binom{|S|}{2} \geq 0 \implies \text{density}(S) = |E_S|/\binom{|S|}{2} \geq \lambda.$$

■

Corollary 3.4.4 *Let* $G = (V, E)$ *be a graph with* $m = |E|$*. For every* $\lambda > m/(m+1)$*, optimizing standard LambdaCC is equivalent to optimizing cluster deletion, and the minimum LambdaCC objective score is exactly* $(1-\lambda)$ *times the minimum cluster-deletion score.*

Proof From Theorem 3.4.3, all output clusters have density greater than $m/(m+1)$. This is only possible if the density is always 1, otherwise a cluster with even a single negative edge would have to contain more than the total number of edges in the graph to reach the minimum density, which is impossible. Therefore all clusters are cliques, and the LambdaCC objective is exactly $(1-\lambda)$ times the cluster deletion objective. ■

3.4.4 Summary of Equivalences

We summarize the equivalence relationships between LambdaCC and other objectives in Figure 3.2. We additionally note that the results of Newman [2016] imply that LambdaCC is also equivalent to the log-likelihood function for the stochastic block model (SBM). This holds both in the case of degree-corrected SBM (which is equivalent to degree-weighted LambdaCC) and the standard SBM (corresponding to standard LambdaCC).

LambdaCC is the first generalized objective function which simultaneously captures relationships among all these objectives. However, we highlight several previous results which have shown connections between pairs of the objectives we consider here. As mentioned previously, the stability measure introduced by Delvenne et al. [2010] generalizes both the modularity and normalized cut objective. Additionally, a link between modularity and correlation clustering was noted by Agarwal and Kempe [2008] who, inspired by the result of Charikar et al. [2005], considered a linear programming relaxation for modularity related to the correlation clustering relaxation. The similarity

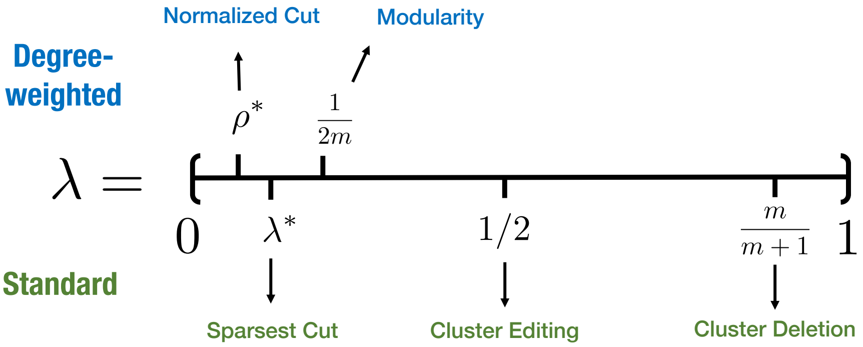


Figure 3.2.: LambdaCC is equivalent to several other objectives for specific values of $\lambda \in (0, 1)$. Values λ^* and ρ^* are not known a priori, but can be obtained by solving LambdaCC for increasingly smaller values of λ.

between cluster deletion and correlation clustering has also been noted previously [Shamir et al., 2004, Dessmark et al., 2007a, Puleo and Milenkovic, 2015].

3.5 LambdaCC Clustering Heuristics

We address the computational complexity of LambdaCC and develop approximation algorithms for it in the next chapter. In this section we outline two heuristics methods for optimizing it in practice. The first algorithm starts with all nodes as singletons, select a node at random, and grows a cluster around it in a way that greedily improves the LambdaCC objective. The second algorithm is a more sophisticated technique based on an adaptation of the Louvain method of Blondel et al. [2008], which was originally developed for the modularity objective. Based on the equivalence between LambdaCC and a generalization of modularity which includes a resolution parameter, we can directly apply previously developed software for a generalized Louvain method [Jeub et al., 2011-2017]. The novelty in this section is a proof which gives bounds (in terms of λ) on the internal density and external sparsity of clusters formed by applying these heuristics specifically within the LambdaCC framework.

Regarding the scalability of these algorithms, we note that in practice we do not explicitly form the signed graph G', which in theory has $O(n^2)$ (positive or negative) edges. Given an initial sparse graph G, it suffices to store positive-edge relationships between nodes, and implicitly apply penalties due to negative edges in G' by considering non-edges in G.

GrowCluster The first heuristic we develop is GrowCluster (Algorithm 1), which iteratively selects an unclustered node uniformly at random and forms a cluster around it by greedily aggre-

Algorithm 1 GrowCluster

```
Input: $G' = (V, E^+, E^-)$
Output: a clustering $\mathcal{C}$ of $G'$
$W \leftarrow V$, $C \leftarrow \emptyset$
while $W \neq \emptyset$ do
    1. Choose a uniformly random $u \in W$, set $S \leftarrow \{u\}$
    2. For all $v \in W \backslash S$, compute benefit from merging $v$ into $S$:
            (For standard LambdaCC: $\Delta_v = \text{cut}(S, \{v\}) - \lambda|S|$)
            (For degree-weighted: $\Delta_v = \text{cut}(S, \{v\}) - \lambda w_v \, \text{vol}(S)$)
    3. Set $m \leftarrow \max_v \Delta_v$, $v' \leftarrow \arg\max \Delta_v$
    while $m > 0$ do
        $S \leftarrow S \cup \{v'\}$
        Update $\Delta_v$, $m$, and $v'$
    Add cluster $S$ to $\mathcal{C}$, update $W \leftarrow W \backslash S$
```

gating adjacent nodes, until there is no more improvement to the LambdaCC objective. A variant of this, called GrowClique (Algorithm 2), is specifically designed for cluster deletion. It monotonically improves the LambdaCC objective, but differs in that at each iteration it randomly selects k unclustered nodes, and greedily grows cliques around each of these seeds. The resulting cliques may overlap: at each iteration we select only the largest of such cliques.

Lambda-Louvain The *Louvain* method is an algorithm developed by Blondel et al. [2008] for modularity clustering. It iteratively visits each node in the graph and moves it to an adjacent cluster, if such a move gives a locally maximum increase in the modularity score. This continues until no move increases modularity, at which point the clusters are aggregated into super-nodes and the entire process is repeated on the aggregated network. By adapting the original Louvain method to make greedy local moves based on the LambdaCC objective, rather than modularity, we obtain a scalable algorithm that is known to provide good approximations for a related objective, and additionally adapts well to changes in our parameter λ. We refer to this as Lambda-Louvain. Both standard and degree-weighted versions of the algorithm can be achieved by employing the existing GenLouvain algorithm of Jeub et al. [2011-2017]).

Output Guarantees Because Lambda-Louvain and GrowCluster are based simply on greedy heuristics, they satisfy no output guarantees with respect to the optimal objective. However, the

Algorithm 2 GrowClique

```
Input: $G' = (V, E^+, E^-)$
Output: a clustering $\mathcal{C}$ of $G'$ where all clusters are cliques
$W \leftarrow V$, $C \leftarrow \emptyset$
while $W \neq \emptyset$ do
    for $i = 1$ to $k$ do
        Select a random seed node $u \in W$, set $S \leftarrow \{u\}$
        Set $N \leftarrow \{v \in W \backslash S : v$ neighbors all nodes in S$\}$
        while $N \neq \emptyset$ do
            $S \leftarrow S \cup \{v\}$ for any $v \in N$
            $N \leftarrow \{v \in W \backslash S : v$ neighbors all nodes in S$\}$
        $S_i \leftarrow S$
    $S_{\max} = \arg\max_i |S_i|$
    Add cluster $S_{\max}$ to $\mathcal{C}$, update $W \leftarrow W \backslash S_{\max}$
```

following theorem shows that these heuristics always form clusters that satisfy specific structural properties with respect to edge density and cut sparsity.

Theorem 3.5.1 *For every λ, standard (respectively, degree-weighted)* Lambda-Louvain *either places all nodes in one cluster, or produce clusters that have scaled sparsity (respectively, scaled normalized cut) bounded above by λ. The same holds true for* GrowCluster*.*

Proof Note that by design, when Lambda-Louvain terminates there will be no two clusters which can be merged to yield a better objective score. Just as in the proof of statement (1) for Theorem 3.4.2, for the standard LambdaCC objective this means that for any pair of cluster S_i and S_j we have

$$\text{cut}(S_i, S_j) - \lambda |S_i||S_j| \leq 0\,. \tag{3.24}$$

We then fix S_i, perform a sum over all other clusters, and get the desired result:

$$\sum_{j \neq i} \text{cut}(S_i, S_j) - \sum_{j \neq i} \lambda |S_i||S_j| \leq 0 \implies \text{cut}(S_i, \bar{S}_i) - \lambda |S_i||\bar{S}_i| \leq 0 \implies \frac{\text{cut}(S_i, \bar{S}_i)}{|S_i||\bar{S}_i|} \leq \lambda\,.$$

If we are using degree-weighted Lambda-Louvain, when the algorithm terminates we know that all pairs of clusters S_i, S_j satisfy

$$\text{cut}(S_i, S_j) - \lambda\, \text{vol}(S_i)\, \text{vol}(S_j) \leq 0$$

and the corresponding result for scaled normalized cut holds.

Though slightly less obvious, it is also true that none of the output clusters of GrowCluster (if there are at least two) could be merged to yield a better objective score. Notice that this is certainly true of the first cluster S_1 formed by GrowCluster: we stop growing S_1 when we find that (for standard-weighted LambdaCC)

$$\text{cut}(S_1, v) - \lambda|S_1| \leq 0$$

for all other nodes v in the graph. Therefore, given *any* other subset of nodes S (including sets of nodes making up other clusters that the algorithm will output), we see

$$\sum_{v \in S} (\text{cut}(S_1, v) - \lambda|S_1|) = \text{cut}(S_1, S) - \lambda|S_1||S| \leq 0.$$

Therefore when we form the second cluster S_2 with GrowCluster, we already know that $\text{cut}(S_1, S_2) - \lambda|S_1||S_2| \leq 0$, and similar reasoning shows that $\text{cut}(S_2, S_j) - \lambda|S_2||S_j| \leq 0$ will hold for any cluster S_j with $j > 2$ that will be subsequently formed. In this way we see that inequality (3.24) will also hold between all pairs of clusters output by GrowCluster, so the rest of the result follows. The same steps will also work for degree-weighted LambdaCC. ■

3.6 Experiments

To end the chapter we consider two sets of experiments which highlight the utility of applying a flexible and generalized clustering framework such as LambdaCC. The first set of experiments demonstrates that many existing algorithms for graph clustering implicitly optimize LambdaCC in different parameter regimes. From this we see that the similarities and differences among existing algorithms for graph clustering can be better understood and evaluated within the context of the LambdaCC framework. These experiments also demonstrates that finding the "best" algorithm for clustering in different contexts can be reduced to the question of finding a proper value of a resolution parameter.

In the second set of experiment we cluster social networks based on the LambdaCC framework. In particular, we find a range of clusterings for each network by optimizing the LambdaCC objective for a range of parameters λ. We then compare the relationship between community membership and metadata attributes associated with the networks. These experiments uncover new insights into the community structure of these social networks, which would be much more challenging to detect if we were to optimize a more rigid objective function.

Code for our algorithms and experiments are available online at `https://github.com/nveldt/LamCC`.

3.6.1 Algorithm Comparison with the LambdaCC Framework

In this section we consider the performance of several previously existing graph clustering algorithms on two synthetic graphs and one real-world network. Each algorithm produces a different type of output clustering, which strikes a certain balance between forming clusterings with sparse external connections and dense internal connections. We then evaluate the performance of each algorithm with respect to the LambdaCC framework. To do so we first obtain lower bounds for the LambdaCC objective by solving a linear program relaxation. We use these lower bounds to measure how well a clustering produced by an algorithm approximates the LambdaCC objective for different values of λ. We find that different algorithms are effectively optimizing the LambdaCC framework in different parameter regimes.

Networks We run experiments on the following three graphs.

1. **ca-GrQc** [Leskovec et al., 2007]. Each node in this network represents an author, and edges represent scientific collaboration as determined by papers on the arXiv e-print website in the category of General Relativity and Quantum Cosmology. The dataset is publicly available on the SNAP network database [Leskovec and Krevl, 2014]. We run experiments on the largest connected component of the network, which has 4158 nodes.

2. **BTER graph**. We generate a graph from the Block Two-Level Erdős Renyi (BTER) model of Seshadhri et al. [2012]. Code for generating BTER graphs is available at `http://www.sandia.gov/~tgkolda/feastpack/`. Graphs can be generated by setting parameters for number of nodes n, target average degree k, max degree bound $maxd$, target maximum clustering coefficient mcc, and target average clustering coefficient $avgcc$. The graph for which we display results was generated with parameters $n = 1000, k = 15, maxd = 50, mcc = 0.95$, and $avgcc = 0.15$.

3. **LFR graph**. Lancichinetti-Fortunato-Radicchi (LFR) graphs are synthetic networks generated with power-law distributions for both node degree and ground truth community size [Lancichinetti et al., 2008]. We include results for a 1000-node LFR graph. This graph was generated using code downloaded from `https://sites.google.com/site/santofortunato/inthepress2` with the following parameters: $n = 1000$ nodes, average degree $k = 15$, mixing parameter $\mu = 0.1$, a maximum degree of 50, a minimum community size of 20, and a maximum community size of 50.

Algorithms For each algorithm we use, we include a brief description, relevant sources, and a link to the implementation we used in our experiments.

1. **Graclus**. Graclus is a multilevel graph partitioning algorithm developed by Dhillon et al. [2007] for optimizing objectives such as normalized cut and ratio cut without computing eigenvectors. Code can be obtained at `http://www.cs.utexas.edu/users/dml/Software/graclus.html`. This algorithm requires the user to specify the exact number of clusters to form. In the main text we show results for partitioning the graph into just two clusters, since this gives very good LambdaCC results for small λ.

2. **Louvain**. For the local moving algorithm of Blondel et al. [2008], we use the implementation available at `https://github.com/ayanonagon/mkse312_project/tree/master/Community_BGLL_Matlab`.

3. **InfoMap**. The InfoMap algorithm is similar to Louvain, but instead optimizes the map equation [Rosvall et al., 2009, Bohlin et al., 2014]. Source code is available at `http://www.mapequation.org/code.html`.

4. **Recursive Maximum Quasi-Clique** (RMQC). We employ the Quick algorithm of Liu and Wong [2008] for finding maximum quasi-cliques; their code is available at `https://www.comp.nus.edu.sg/~wongls/projects/pattern-spaces/quick-v1/`. We run this procedure for a specified density ρ, and extract the largest clique in the network. We then remove the quasi-clique and recurse on the remaining nodes.

5. **Recursive Maximum Clique** (RMC). For recursively extracting maximum cliques we use the Parallel Maximum Clique (PMC) library [Rossi et al., 2015] (`http://maximumclique.com/`). Interestingly, this procedure is known to give a 2-approximation to the optimal cluster deletion objective [Dessmark et al., 2007a]. However, finding the maximum clique problem is NP-complete, which disqualifies this procedure from being considered a constant-factor approximation algorithm for cluster deletion.

6. **Lambda-Louvain** To greedily optimize the LambdaCC objective, we use the GenLouvain algorithm Jeub et al. [2011-2017]. When applying the GenLouvain algorithm to the LambdaCC framework, we have referred to this approach as Lambda-Louvain.

LP relaxation for LambdaCC In order to measure how well we each algorithm approximately optimizes the LambdaCC objective, we compute lower bounds for the optimal LambdaCC objective score by solving the following linear programming relaxation:

$$\begin{array}{ll} LP(\lambda) = \min & \sum_{(i,j)\in E}(1-\lambda)x_{ij} + \sum_{(i,j)\notin E}\lambda(1-x_{ij}) \\ \text{subject to} & x_{ij} \leq x_{ik} + x_{jk} \text{ for all } i,j,k \\ & 0 \leq x_{ij} \leq 1 \text{ for all } i,j. \end{array} \tag{3.25}$$

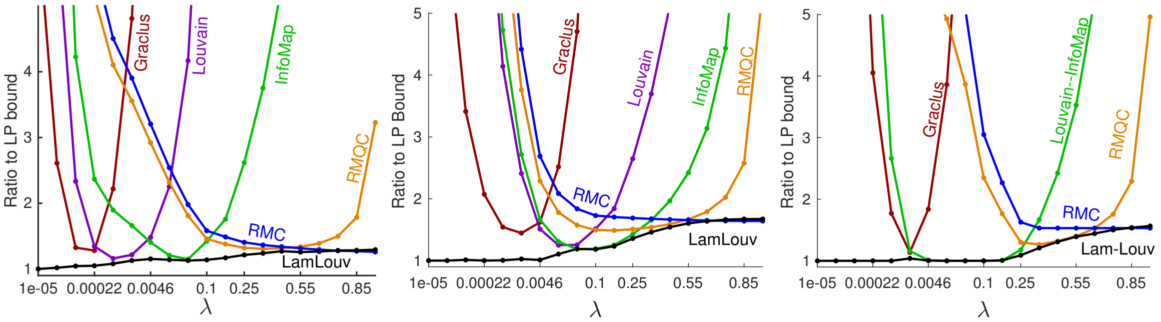
This linear program is central in many results we develop in subsequent chapters for LambdaCC. Later we will consider memory-efficient techniques for solving this LP relaxation in practice (Chapter 5), and techniques for rounding this LP in order to obtain approximation guarantees for LambdaCC (Chapter 4). We will also show how to use lower bounds obtained from this LP in order to "learn" good values of the parameter λ when applying the LambdaCC framework in practice (Chapter 6). For now, it is enough to note that even though it is infeasible to solve the NP-hard objective exactly, solving the LP relaxation for a fixed value of λ provides a good lower bound on the optimal LambdaCC objective for that λ. Therefore, in order to bound how well a clustering $\mathcal{C}$ approximates LambdaCC, we compute the ratio between the clustering's objective score $\textbf{LamCC}_\lambda(\mathcal{C})$ and the LP lower bound:

$$\textbf{ApproxRatio}(\mathcal{C}, \lambda) = \frac{\textbf{LamCC}_\lambda(\mathcal{C})}{LP(\lambda)}. \tag{3.26}$$

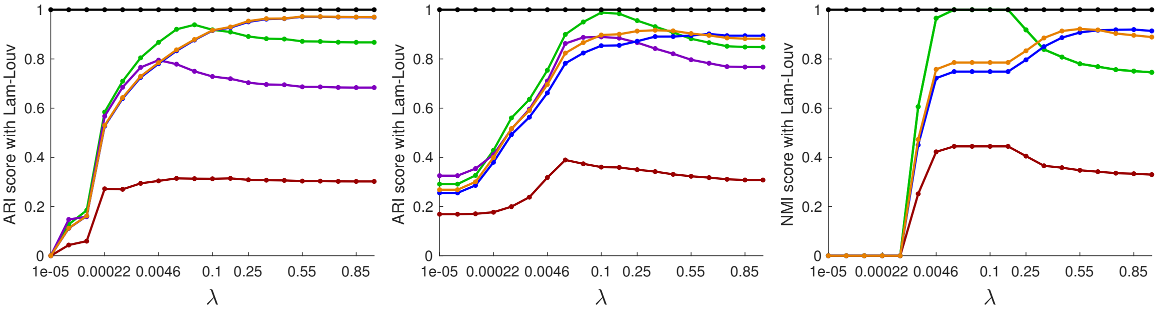
If a clustering $\mathcal{C}$ output by an algorithm satisfies $\textbf{ApproxRatio}(\mathcal{C}, \lambda) = \Delta$ for a fixed λ, then this means that the algorithm returned a clustering that is at most a Δ-approximation to the optimal LambdaCC objective for that resolution parameter.

Experimental Results In Figures 3.3a, 3.3b, and 3.3c, for each algorithm tested we plot the ratio between the LambdaCC objective score and the LambdaCC lower bound determined by solving the LP relaxation. For these plots, we have Graclus form only two clusters. For RMQC, we have the method return maximum quasi-cliques with a density of at least 0.6. The bowl-shaped curves in these plots indicate that different algorithms are effectively optimizing the LambdaCC objective in different parameter regimes. Consider for example the clustering returned by Graclus on ca-GrQc (Figure 3.3a). Note that for very small values of λ (e.g. λ around 0.00022), the objective score for the Graclus clustering (shown in dark red) is well below a factor 2-approximation for the LambdaCC objective. As λ increases, the Graclus clustering no longer is a good approximation for LambdaCC, but in each parameter regime at least one of the off-the-shelf algorithms has a good approximation guarantee. Lambda-Louvain is the only algorithm in Figures 3.3a, 3.3b, and 3.3c that forms a new clustering for each distinct value of λ. This method therefore always does a good job approximating the LambdaCC objective and interpolates the performance of other algorithms.

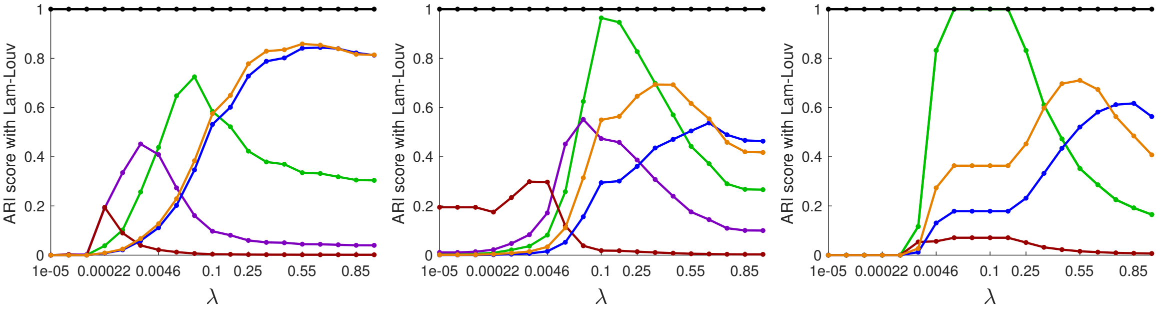
In Figure 3.3 we also display normalized mutual information (NMI) and adjusted rand index (ARI) scores between the Lambda-Louvain clustering and the output of other algorithms. We note that the NMI and ARI scores peak in the same regime where each algorithm best optimizes LambdaCC. Often, the peaks for these scores are higher for larger values of λ. This can be explained by realizing that when λ is small, fewer clusters are formed. It is natural to expect there to be many ways to partition the graph into a small number of clusters such that different clusterings share a very similar structure, even if the individual clusters themselves do not match. On the whole, the plots



(a) ca-GrQc LP Ratio (b) BTER graph LP Ratio (c) LFR graph LP Ratio



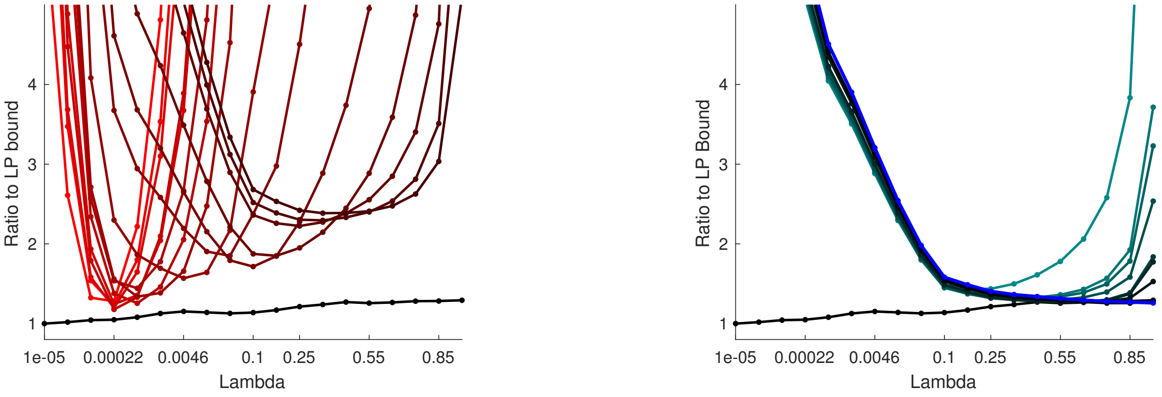
(d) ca-GrQc NMI scores (e) BTER NMI scores (f) LFR NMI scores



(g) ca-GrQc ARI scores (h) BTER ARI scores (i) LFR ARI scores

Figure 3.3.: Each curve in each plot (except the LAMBDA-LOUVAIN curve in black) represents a single clustering output by a certain algorithm. For each clustering, the top row of plots shows the ratio between the clustering's LambdaCC score and a lower bound on the optimal score determined by a solving an LP relaxation. The black curve represents the performance of LAMBDA-LOUVAIN. This algorithm recomputes a new clustering for each value of λ. The second and third row show NMI and ARI scores between each clustering and the LAMBDA-LOUVAIN clustering at each λ.

in Figure 3.3 illustrate that our framework effectively interpolates between several well-established strategies in graph clustering, and can serve as a good proxy for any clustering task for which any one of these algorithms is known to be effective.



(a) Multiple runs of Graclus on the caGrQc network, with increasingly many clusters.

(b) Multiple runs of RMQC on the caGrQc network, with increasingly higher quasi-clique density.

Figure 3.4.: As we increase the number of clusters formed by Graclus (left), the algorithm does better for large values of λ and worse for small values. The algorithm seems particularly well-suited to optimize the LambdaCC objective for very small values of λ. Darker curves represent a larger number of clusters formed; the number of clusters formed ranges from 2 on the far left of the plot to just over 2200 for the right-most curve. In the right plot we vary the density ρ of quasi-cliques formed by RMQC from 0.5 to 0.85. In this plot, darker curves represent a larger density. As density increases, the curves converge to the performance of RMC, shown in blue.

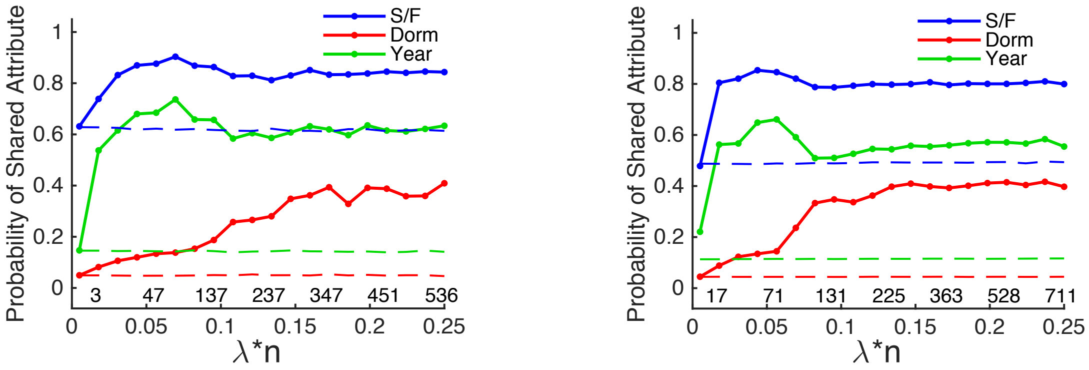
Parametric Graclus and RMQC By performing multiple runs of Graclus and varying the number of partitions formed by this algorithm, we can show that Graclus can approximately optimize different parameter regimes of LambdaCC. In Figure 3.4a we show how the Graclus objective scores change as we increase the number of clusters from 2 to over 2000. As the number of clusters increases, the algorithm performs better and better for large λ and worse for smaller λ. Figure 3.4b shows that something similar occurs for RMQC when we vary the minimum density of quasi-cliques from 0.5 to 0.85. As the inner-edge density increases, the performance of RMQC essentially converges to the performance of RMC.

3.6.2 Social Network Analysis

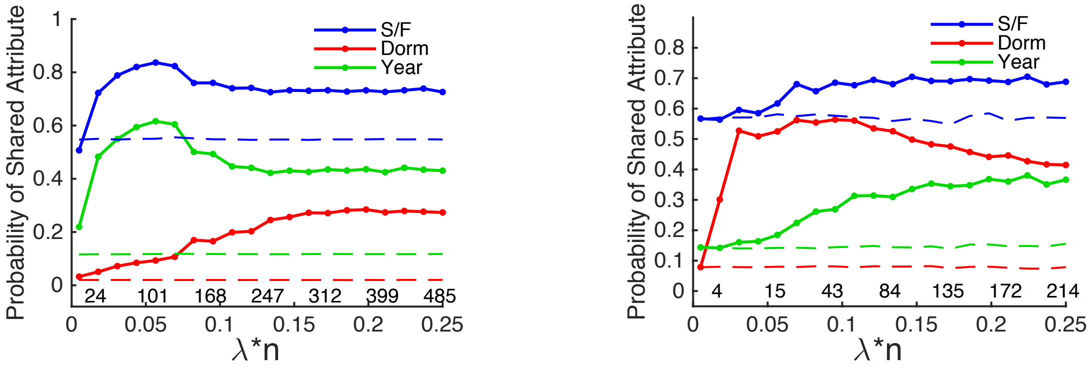
Clustering a social network using a range of resolution parameters can reveal valuable insights about how links are formed in the network. Here we examine several graphs from the Facebook100 dataset [Traud et al., 2012], each of which represents the induced subgraph of the Facebook network corresponding to a US university at some point in 2005. The networks come with anonymized metadata, reporting attributes such as major and graduation year for each node. While metadata attributes are not expected to correspond to ground-truth communities in the network [Peel et al., 2017], we do expect them to play a role in how friendship links and communities are formed. In this experiment we illustrate strong correlations between the edge structure of the networks and the dorm, graduation year, and student/faculty status metadata attributes. We also see how these correlations are revealed, to different degrees, depending on our choice of λ.

Given a Facebook subgraph with n nodes, we cluster it with degree-weighted LAMBDA-LOUVAIN for a range of λ values between $0.005/n$ and $0.25/n$. In this clustering, we refer to two nodes in the same cluster as an *interior pair*. We measure how well a metadata attribute M correlates with the clustering by calculating the proportion of interior pairs that share the same value for M. This value, denoted by $P(M)$, can also be interpreted as the probability of selecting an interior pair uniformly at random and finding that they agree on attribute M. To determine whether the probability is meaningful, we compare it against a null probability $P(\tilde{M})$: the probability that a random interior pair agree at a *fake* metadata attribute $\tilde{M}$. We assign to each node a value for the fake attribute $\tilde{M}$ by performing a random permutation on the vector storing values for true attribute M. In this way, we can compare each true attribute M against a fake attribute $\tilde{M}$ that has the same exact proportion of nodes with each attribute value, but does not impart any true information regarding each node.

We plot results for each of the three attributes $M \in \{\textit{dorm}, \textit{year}, s/f \text{ (student/faculty)}\}$ on four Facebook networks in Figure 3.5, as λ is varied. In all cases, we see significant differences between $P(M)$ and $P(\tilde{M})$. In general, $P(\textit{year})$ and $P(s/f)$ reach a peak at small values of λ when clusters are large, whereas $P(\textit{dorm})$ is highest when λ is large and clusters are small. This indicates that the first two attributes are more highly correlated with large sparse communities in the network, whereas sharing a dorm is more correlated with smaller, denser communities. Caltech, a small residential university, is an exception to these trends and exhibits a much stronger correlation with the dorm attribute, even for very small λ.



(a) Swarthmore $n = 1659$ (b) Yale $n = 8578$



(c) Cornell $n = 18660$ (d) Caltech $n = 769$

Figure 3.5.: On four university Facebook graphs, we illustrate that the dorm (red), graduation year (green), and student/faculty (S/F) status (blue) metadata attributes all correlate highly with the clustering found by Lambda-Louvain for each λ. Above the x-axis we show the number of clusters formed, which strictly increases with λ. The y-axis reports the probability that two nodes sharing a cluster also share an attribute value. Each attribute curve is compared against a null probability, shown as a dashed line of the same color. The dashed line reports the same probability computed for a related *fake* attribute, which was generated by randomly permuting the values of the corresponding true attribute. The large gaps between each attribute curve and its null probability indicate that the link structure of all networks is highly correlated with these attributes. In general, probabilities for *year* and *s/f* status are highest for small λ, whereas *dorm* has a higher correlation with smaller, denser communities in the network. Caltech is an exception to the general trend; see the main text for discussion.

4. COMPLEXITY RESULTS FOR LAMBDACC

4.1 Chapter Overview

This chapter presents an improved approximation algorithm for a specially weighted version of correlation clustering that was first considered by Puleo and Milenkovic [2015]. These authors showed that rounding a linear programming relaxation can yield a $(5 - 1/\tau)$-approximation for a generalization of the *probability weights* version of correlation clustering, where $\tau \geq 1$ is a parameter associated with the weights. We improve this to a 3-approximation using a new rounding technique, adapted from the results of van Zuylen and Williamson [2009]. Significantly, the weighted variant considered here is a generalization of LambdaCC for $\lambda \geq 1/2$. Thus an approximation algorithm for LambdaCC (when $\lambda \geq 1/2$) is obtained as an immediate corollary. Additionally, we show that a variation of the algorithm produces a 2-approximation specifically for cluster deletion, which is now the best known approximation for this problem. For values of λ smaller than 1/2, we give parameter dependent approximation guarantees.

After proving approximation results, we show that for a certain small choice of λ, the LambdaCC LP relaxation has an $\Omega(\log n)$ integrality gap. This indicates that LambdaCC is much more challenging to approximate for small values of the resolution parameter. This result does *not* prove that obtaining constant factor approximations for arbitrarily small λ is NP-hard. However, the best approximation factors for minimizing disagreements for a wide range of correlation clustering variants all rely on rounding the linear programming relaxation. The only other lower bound that is consistently used in proving approximations for minimizing disagreements is a triangle packing lower bound [Bansal et al., 2004, Ailon et al., 2008], which is not as tight as the LP relaxation. Thus, our result strongly suggests that constant factor approximations for arbitrary λ are unlikely, or at least would require the design of completely new techniques for correlation clustering.

The proof of a 3-approximation for LambdaCC when $\lambda \geq 1/2$, as well as the 2-approximation for cluster deletion, were first shown in the conference paper which introduced the Lambda framework [Veldt et al., 2018b]. The $\Omega(\log n)$ integrality gap for the linear program, and the parameter dependent approximation guarantees for LambdaCC, were published in the proceedings of the 29th International Symposium on Algorithms and Computation [Gleich et al., 2018]. This thesis provides the first proof of the 3-approximation for the generalized weighted version of correlation clustering considered by Puleo and Milenkovic [2015].

4.2 Heavily Negative Weighted Correlation Clustering

Puleo and Milenkovic [2015] were the first to consider weighted versions of correlation clustering satisfying the following properties for every pair of nodes (i, j):

$$w_{ij}^{+} \leq 1 \tag{4.1}$$

$$w_{ij}^{-} \leq \tau \text{ for some } \tau \in [1, \infty] \tag{4.2}$$

$$w_{ij}^{+} + w_{ij}^{-} \geq 1\,. \tag{4.3}$$

These authors proved a $5 - 1/\tau \geq 4$ approximation for this variant by altering the LP rounding procedure of Charikar et al. [2005]. Note that scaling all weights by a constant factor will not change the approximation guarantees for the problem. This observation allows us to immediately obtain approximation guarantees for LambdaCC. If we begin with an instance of standard LambdaCC and divide all weights by $(1-\lambda)$, we obtain a problem in which weights satisfy $(w_{ij}^{+}, w_{ij}^{-}) \in \{(1, 0), (0, \lambda/(1-\lambda))\}$. If $\lambda \geq 1/2$, inequalities (4.1), (4.2), and (4.3) all hold, and we can apply the results of Puleo and Milenkovic [2015] to obtain a 5-approximation for LambdaCC in this parameter regime.

In this section we show that the $5 - \tau \geq 4$ approximation can be improved to a 3-approximation for the extended weight bounds considered by Puleo and Milenkovic [2015]. Here we will in fact drop the parameter τ, since it unnecessary for our analysis and we prove an approximation that is independent of this value. We focus on the first and third bound given above, i.e., we consider weighted instances that satisfy

$$w_{ij}^{+} \leq 1 \tag{4.4}$$

$$w_{ij}^{+} + w_{ij}^{-} \geq 1 \tag{4.5}$$

for all pairs (i, j). We will refer to this as *Heavily Negative Weighted* correlation clustering. We will apply the results of van Zuylen and Williamson [2009] to obtain a 3-approximation for the problem based on a *pivoting* technique for LP-rounding.

4.2.1 Deterministic Pivoting Theorem

Before proceeding we review the results of van Zuylen and Williamson [2009] for obtaining deterministic pivoting algorithms for correlation clustering. Algorithm 3 gives the CC-Pivot method, which repeatedly selects an unclustered node and clusters it with all of its positive neighbors that have not yet been clustered. If nodes are chosen uniformly at random, then this corresponds to the fast randomized 3-approximation for unweighted complete correlation clustering developed by Ailon et al. [2008]. The same authors also gave an better 2.5 approximation that relies on first solving the

Algorithm 3 CC-Pivot

```
Input: Signed graph $G = (V, E^+, E^-)$
Output: Clustering $\mathcal{C} = \text{CC-Pivot}(G)$
Select a pivot node $k \in V$
Form cluster $S = \{v \in V : (k, v) \in E^+\}$
Output clustering $\mathcal{C} = \{S, \text{CC-Pivot}(G - S)\}$
```

LP relaxation. Later, van Zuylen and Williamson provided a powerful framework for de-randomizing pivoting algorithms for correlation clustering. Not only does this allow one to obtain deterministic approximation guarantees, but it also provides a simple framework for developing new approximation algorithms for new variants of correlation clustering. We restate a key theorem; a detailed proof is included in the work of van Zuylen and Williamson [2009].

Theorem 4.2.1 *(Theorem 3.1 in [van Zuylen and Williamson, 2009]) Let* $G = (V, W^+, W^-)$ *be a signed, weighted graph where each pair of nodes* (i, j) *has positive and negative weights* $w_{ij}^+ \in W^+$ *and* $w_{ij}^- \in W^-$. *Given a set of budgets* $\{c_{ij} : i \in V, j \in V, i \neq j\}$, *and an unweighted graph* $\tilde{G} = (V, F^+, F^-)$ *satisfying the following assumptions:*

1. $w_{ij}^- \leq \alpha c_{ij}$ *for all* $(i, j) \in F^+$ *and*

   $w_{ij}^+ \leq \alpha c_{ij}$ *for all* $(i, j) \in F^-$,

2. $w_{ij}^+ + w_{jk}^+ + w_{ik}^- \leq \alpha \left(c_{ij} + c_{jk} + c_{ik}\right)$

   for every triplet $\{i, j, k\}$ *in* $\tilde{G}$ *with* $(i, j), (j, k) \in F^+$, $(i, k) \in F^-$,

then applying CC-Pivot *on* $\tilde{G}$ *will return a solution that costs at most* $\alpha \sum_{i<j} c_{ij}$ *if we choose a pivot* k *that minimizes:*

$$\frac{\sum_{(i,j) \in T_k^+(G)} w_{ij}^+ + \sum_{(i,j) \in T_k^-(G)} w_{ij}^-}{\sum_{(i,j) \in T_k^+(G) \cup T_k^-(G)} c_{ij}}.$$

where

$$T_k^+(G) = \{(i, j) \in F^+ : (k, j) \in F^-, (k, i) \in F^+\}$$

$$T_k^-(G) = \{(i, j) \in F^- : (k, j) \in F^+, (k, i) \in F^+\}.$$

We note, as the original authors did, that the given approximation guarantee holds in expectation if pivot nodes are chosen uniformly at random. A common approach to applying the above theorem is

Algorithm 4 LP-Round-and-Pivot

```
Input: Instance of correlation clustering: $G = (V, W^+, W^-)$
Output: Clustering $\mathcal{C}$ of $G$
Solve the CC LP-relaxation (5.2), obtaining distances $(x_{ij})$
Construct $\tilde{G} = (V, F^+, F^-)$ by assigning edges in the following way:
    If $x_{ij} < \beta$, assign $(i, j) \in F^+$ in $\tilde{G}$
    If $x_{ij} > \beta$, assign $(i, j) \in F^-$ in $\tilde{G}$
    If $x_{ij} = \beta$, arbitrarily assign $(i, j)$ to be a positive or negative edge in $\tilde{G}$.
Return the clustering from running CC-Pivot on $\tilde{G}$.
```

to obtain budgets $c_{ij} = w_{ij}^+ x_{ij} + w_{ij}^-(1 - x_{ij})$ for each pair $(i, j) \in V \times V$ by solving the LP-relaxation for correlation clustering:

$$\begin{array}{ll} \text{minimize} & \sum_{i<j} w_{ij}^+ x_{ij} + w_{ij}^-(1 - x_{ij}) \\ \text{subject to} & x_{ij} \leq x_{ik} + x_{jk} \text{ for all } i, j, k \\ & 0 \leq x_{ij} \leq 1 \text{ for all } i, j. \end{array} \tag{4.6}$$

After this, one can round the relaxed distances to construct an unweighted signed graph, on which to apply the CC-pivot algorithm. A generic version of this technique is shown in Algorithm 4.

4.2.2 Approximations for Probability Weights

In their work, van Zuylen and Williamson [2009] showed that running Algorithm 4 with $\beta = 1/2$ returns a 3-approximation when weights satisfy probability constraints: $w_{ij}^+ + w_{ij}^- = 1$ for all $(i, j) \in V \times V$. Here we use a very similar proof technique to show that the same result will also hold if we set $\beta = 1/3$.

Theorem 4.2.2 *If $w_{ij}^+ + w_{ij}^- = 1$ and $w_{ij}^+ \geq 0, w_{ij}^- \geq 0$ for every pair of nodes (i, j) in an instance of correlation clustering, then Theorem 4.2.1 is satisfied with $\alpha = 3$ if we partition the edge set so that for every $(i, j) \in F^+$ we have $x_{ij} \leq 1/3 = \beta$ and for every $(i, j) \in F^-$ we have $x_{ij} \geq 1/3 = \beta$.*

Proof The first condition holds because if $(i, j) \in F^+$, then $x_{ij} \leq 1/3 \implies (1 - x_{ij}) \geq 2/3$ so $3c_{ij} = 3(w_{ij}^+ x_{ij} + w_{ij}^-(1 - x_{ij})) \geq 3(2/3 w_{ij}^-) \geq w_{ij}^-$. Similarly, if $(i, j) \in F^-$ then $x_{ij} \geq 1/3$ and $3c_{ij} = 3(w_{ij}^+ x_{ij} + w_{ij}^-(1 - x_{ij})) \geq w_{ij}^+$.

To prove the second condition, let $\{i, j, k\}$ be a "bad" triplet of nodes where $(i, j), (j, k) \in F^+$ and $(i, k) \in F^-$. This means that $x_{ij} \leq 1/3$, $x_{jk} \leq 1/3$, and $x_{ik} \geq 1/3$. Set $w_{ij} = w_{ij}^+ \in [0, 1]$, and $\bar{w}_{ij} = w_{ij}^- = (1 - w_{ij})$. Similarly, define $w_{jk} = w_{jk}^+ \in [0, 1]$, $\bar{w}_{jk} = 1 - w_{jk}$, $w_{ik} = w_{ik}^- \in [0, 1]$, and

$\bar{w}_{ik} = 1 - w_{ik}$. In other words, w_{ab} denotes what is often called the "backward" cost of edge (a, b) and $\bar{w}_{ab}$ denotes the "forward" cost. Next we introduce variables associated with the distance scores for each edge:

$$\begin{aligned} y_{ij} &= 1 - x_{ij} \geq 2/3 \\ y_{jk} &= 1 - x_{jk} \geq 2/3 \\ y_{ik} &= x_{ik} \geq 1/3\,. \end{aligned}$$

Introducing these new variables, which are defined differently for positive and negative edges, allows us to express the value of LP-budgets in a uniform manner:

$$c_{ij} = w^+_{ij} x_{ij} + w^-_{ij}(1 - x_{ij}) = w_{ij}(1 - y_{ij}) + \bar{w}_{ij} y_{ij} = w_{ij} + y_{ij}(\bar{w}_{ij} - w_{ij}) \tag{4.7}$$

$$c_{jk} = w^+_{jk} x_{jk} + w^-_{jk}(1 - x_{jk}) = w_{jk}(1 - y_{jk}) + \bar{w}_{jk} y_{jk} = w_{jk} + y_{jk}(\bar{w}_{jk} - w_{jk}) \tag{4.8}$$

$$c_{ik} = w^+_{ik} x_{ik} + w^-_{ik}(1 - x_{ik}) = w_{ik}(1 - y_{ik}) + \bar{w}_{ik} y_{ik} = w_{ik} + y_{ik}(\bar{w}_{ik} - w_{ik})\,. \tag{4.9}$$

Our goal is to show that

$$(c_{ij} + c_{jk} + c_{ik}) \geq \frac{w_{ij} + w_{jk} + w_{ik}}{3}. \tag{4.10}$$

Note that

$$c_{ij} + c_{jk} + c_{ik} = w_{ij} + w_{jk} + w_{ik} + y_{ij}(\bar{w}_{ij} - w_{ij}) + y_{jk}(\bar{w}_{jk} - w_{jk}) + y_{ik}(\bar{w}_{ik} - w_{ik}). \tag{4.11}$$

Given the symmetric relationship between variables associated with pairs (i, j) and (j, k), we can assume without loss of generality that

$$(\bar{w}_{ij} - w_{ij}) \leq (\bar{w}_{jk} - w_{jk}).$$

We consider three different cases to prove the final result. We will repeatedly use the fact that $y_{ij} + y_{jk} + y_{ik} \leq 2$, which holds because of the triangle inequality on x_{ij}, x_{jk}, and x_{ik}.

Case 1: $0 \leq \bar{w}_{ij} - w_{ij} \leq \bar{w}_{jk} - w_{jk}$ and $0 \leq \bar{w}_{ik} - w_{ik}$. In this case note that $c_{ij} + c_{jk} + c_{ik} \geq w_{ij} + w_{jk} + w_{jk}$, since we can simply drop the last three rightmost positive terms of (4.11).

Case 2: $\bar{w}_{ij} - w_{ij} \leq \bar{w}_{ik} - w_{ik}$ and $\bar{w}_{ij} - w_{ij} < 0$. If $\bar{w}_{ij} - w_{ij}$ is negative and smaller than both $\bar{w}_{jk} - w_{jk}$ and $\bar{w}_{ik} - w_{ik}$, then (4.11) will be minimized when y_{ij} is a large as possible, i.e. $y_{ij} = 1$. To see why, assume that $y_{ij} < 1$ and we will show the expression is not yet minimized. If the constraint $y_{ij} + y_{ik} + y_{jk} \leq 2$ is not tight, then we could just increase y_{ij} and the value of (4.11) would be smaller. If the equality is tight, then regardless of the signs or relative order of $(\bar{w}_{ik} - w_{ik})$ and $(\bar{w}_{jk} - w_{jk})$, expression (4.11) would be smaller if we decreased either y_{ik}, y_{jk}, or both, and increased y_{ij} to 1.

Next observe that if $y_{ij} = 1$, then y_{ik} and y_{jk} must be tight against their lower bounds, $y_{jk} = 2/3$ and $y_{ik} = 1/3$, so that the constraint $y_{ij} + y_{jk} + y_{ik} \leq 2$ is not violated. Thus

$$\begin{aligned} c_{ij} + c_{jk} + c_{ik} &\geq w_{ij} + w_{jk} + w_{ik} + (\bar{w}_{ij} - w_{ij}) + \frac{2}{3}(\bar{w}_{jk} - w_{jk}) + \frac{1}{3}(\bar{w}_{ik} - w_{ik}) \\ &= \bar{w}_{ij} + \frac{1}{3}(\bar{w}_{jk} + w_{jk}) + \frac{1}{3}\bar{w}_{jk} + \frac{1}{3}(\bar{w}_{ik} + w_{ik}) + \frac{1}{3}w_{ik} \\ &\geq \frac{1}{3} + \frac{1}{3} + \frac{1}{3}w_{ik} \\ &\geq \frac{w_{ij}}{3} + \frac{w_{jk}}{3} + \frac{w_{ik}}{3}, \end{aligned}$$

since $\bar{w}_{jk} + w_{jk} = \bar{w}_{ik} + w_{ik} = 1$. So we have shown (4.10) holds. a

Case 3: $\bar{w}_{ik} - w_{ik} \leq \bar{w}_{ij} - w_{ij}$ and $\bar{w}_{ik} - w_{ik} < 0$. If $\bar{w}_{ik} - w_{ik}$ is negative and smaller than both $\bar{w}_{jk} - w_{jk}$ and $\bar{w}_{ij} - w_{ij}$, then (4.11) will be minimized when y_{ik} is as large as possible without violating the constraint $y_{ij} + y_{jk} + y_{ik} \leq 2$. Since y_{ij} and y_{jk} both have a lower bound of 2/3, y_{ik} has an upper bound of 2/3, so (4.11) is minimized when $y_{ik} = y_{ij} = y_{jk} = 2/3$. Thus

$$\begin{aligned} c_{ij} + c_{jk} + c_{ik} &\geq w_{ij} + w_{jk} + w_{ik} + \frac{2}{3}(\bar{w}_{ij} - w_{ij}) + \frac{2}{3}(\bar{w}_{jk} - w_{jk}) + \frac{2}{3}(\bar{w}_{ik} - w_{ik}) \\ &= \frac{\bar{w}_{ij} + \bar{w}_{jk} + \bar{w}_{ik}}{3} + \frac{1}{3}(\bar{w}_{jk} + w_{jk}) + \frac{1}{3}(\bar{w}_{jk} + w_{jk}) + \frac{1}{3}(\bar{w}_{ik} + w_{ik}) \\ &\geq \frac{w_{ij} + w_{jk} + w_{ik}}{3}, \end{aligned}$$

so again (4.10) holds. These cases cover all possibilities, so the full result is shown.

■

In order to extend the result to the *heavily negative weighted* case where $w^+_{ij} + w^-_{ij}$ can be greater than 1, we will extract a useful fact directly from the proof of Theorem 4.2.2.

Lemma 4.2.3 *Consider a set of real numbers $x_{ij} \in [0, 1/3]$, $x_{jk} \in [0, 1/3]$, $x_{ik} \in [1/3, 1]$, $w_{ij} \in [0, 1]$, $w_{jk} \in [0, 1]$, and $w_{ik} \in [0, 1]$. If (x_{ij}, x_{ik}, x_{jk}) satisfy triangle inequality constraints, then*

$$2 - \frac{4}{3}w_{ij} - x_{ij} + 2w_{ij}x_{ij} - \frac{4}{3}w_{jk} - x_{jk} + 2w_{jk}x_{jk} + (1 - w_{ik})x_{ik} + w_{ik}\left(\frac{2}{3} - x_{ik}\right) \geq 0. \tag{4.12}$$

In particular, the above expression holds when $w_{ik} = 1$, i.e.

$$2 - \frac{4}{3}w_{ij} - x_{ij} + 2w_{ij}x_{ij} - \frac{4}{3}w_{jk} - x_{jk} + 2w_{jk}x_{jk} + \left(\frac{2}{3} - x_{ik}\right) \geq 0 \tag{4.13}$$

Proof This lemma is simply a rearrangement of a fact that was proven in Theorem 4.2.2. In particular, given the input from the statement of the lemma, define

$$(w^+_{ij}, w^-_{ij}) = (w_{ij}, 1 - w_{ij}), \qquad (w^+_{jk}, w^-_{jk}) = (w_{jk}, 1 - w_{jk}), \qquad (w^+_{ik}, w^-_{ik}) = (1 - w_{ik}, w_{ik}).$$

Also define

$$\begin{aligned} c_{ij} &= w^+_{ij}x_{ij} + w^-_{ij}(1 - x_{ij}) = w_{ij}x_{ij} + (1 - w_{ij})(1 - x_{ij}) \\ c_{jk} &= w^+_{jk}x_{jk} + w^-_{jk}(1 - x_{jk}) = w_{jk}x_{jk} + (1 - w_{jk})(1 - x_{jk}) \\ c_{ik} &= w^+_{ik}x_{ik} + w^-_{ik}(1 - x_{ik}) = (1 - w_{ik})x_{ik} + w_{ik}(1 - x_{ik})\,. \end{aligned}$$

Theorem 4.2.2 showed that for this exact situation, the following inequality is satisfied:

$$(c_{ij} + c_{jk} + c_{ik}) \geq \frac{w_{ij} + w_{jk} + w_{ik}}{3}.$$

Expanding and rearranging, we get inequality (4.12):

$$\begin{aligned} & w_{ij}x_{ij} + (1 - w_{ij})(1 - x_{ij}) + w_{jk}x_{jk} + (1 - w_{jk})(1 - x_{jk}) \\ & \qquad + (1 - w_{ik})x_{ik} + w_{ik}(1 - x_{ik}) \geq \frac{w_{ij} + w_{jk} + w_{ik}}{3} \\ \Longrightarrow\ & (1 - w_{ij} - x_{ij} + 2w_{ij}x_{ij}) + (1 - w_{jk} - x_{jk} + 2w_{jk}x_{jk}) \\ & \qquad + (1 - w_{ik})x_{ik} + w_{ik}(1 - x_{ik}) \geq \frac{w_{ij} + w_{jk} + w_{ik}}{3} \\ \Longrightarrow\ & 2 - \frac{4}{3}w_{ij} - x_{ij} + 2w_{ij}x_{ij} - \frac{4}{3}w_{jk} - x_{jk} + 2w_{jk}x_{jk} \\ & \qquad + (1 - w_{ik})x_{ik} + w_{ik}\left(\frac{2}{3} - x_{ik}\right) \geq 0 \end{aligned}$$

Inequality (4.13) is obtained by plugging in $w_{ik} = 1$. ■

4.2.3 Extending the Result

Now consider an instance of heavily negative weighted correlation clustering $G = (V, W^+, W^-)$, in which for every pair of nodes i, j we have

$$w^+_{ij} \leq 1 \tag{4.14}$$

$$w^+_{ij} + w^-_{ij} \geq 1 \implies w^+_{ij} \geq 1 - w^-_{ij} \text{ and } w^-_{ij} \geq 1 - w^+_{ij}\,. \tag{4.15}$$

We prove the following result:

Theorem 4.2.4 *If $w^+_{ij} + w^-_{ij} \geq 1$ and $w^+_{ij} \in [0, 1], w^-_{ij} \geq 0$ for every pair of nodes (i, j) in an instance of correlation clustering, then Theorem 4.2.1 is satisfied with $\alpha = 3$ if we partition the edge set so that for every $(i, j) \in F^+$ we have $x_{ij} \leq 1/3 = \beta$ and for every $(i, j) \in F^-$ we have $x_{ij} \geq 1/3 = \beta$.*

Proof The first condition of Theorem 4.2.1 holds using the same exact proof that applied to probability weights in Theorem 4.2.2. We therefore just focus on proving the second condition. Let

$\{i, j, k\}$ be a bad triangle of nodes in $\tilde{G} = (V, F^+, F^-)$, where $(i, k) \in F^-$, so distance variables satisfy $x_{ij} \leq 1/3$, $x_{jk} \leq 1/3$, and $x_{ik} \geq 1/3$. We again let $w_{ij} = w_{ij}^+ \in [0, 1]$, and $\bar{w}_{ij} = w_{ij}^- \geq (1 - w_{ij})$. Similarly, define $w_{jk} = w_{jk}^+ \in [0, 1]$, and $\bar{w}_{jk} = w_{jk}^- \geq 1 - w_{jk}$. Finally, $w_{ik} = w_{ik}^-$, and $\bar{w}_{ik} = w_{ik}^+ \geq 1 - w_{ik}$. We also again define

$$\begin{aligned} y_{ij} &= 1 - x_{ij} \geq 2/3 \\ y_{jk} &= 1 - x_{jk} \geq 2/3 \\ y_{ik} &= x_{ik} \geq 1/3. \end{aligned}$$

As before, we must show that

$$(c_{ij} + c_{jk} + c_{ik}) \geq \frac{w_{ij} + w_{jk} + w_{ik}}{3}\,. \tag{4.16}$$

We have the following LP budgets and bounds for them based on the bounds in (4.15):

$$c_{ij} = w_{ij}^+ x_{ij} + w_{ij}^-(1 - x_{ij}) \geq w_{ij}x_{ij} + (1 - w_{ij})(1 - x_{ij}) \tag{4.17}$$

$$c_{jk} = w_{jk}^+ x_{jk} + w_{jk}^-(1 - x_{jk}) \geq w_{jk}x_{jk} + (1 - w_{jk})(1 - x_{jk}) \tag{4.18}$$

$$c_{ik} = w_{ik}^+ x_{ik} + w_{ik}^-(1 - x_{ik}) \geq (1 - w_{ik})x_{ik} + w_{ik}(1 - x_{ik})\,. \tag{4.19}$$

Expanding (4.16), we see that our goal is to show

$$w_{ij}^+ x_{ij} + w_{ij}^-(1 - x_{ij}) + w_{jk}^+ x_{jk} + w_{jk}^-(1 - x_{jk}) + w_{ik}^+ x_{ik} + w_{ik}^-(1 - x_{ik}) \geq \frac{w_{ij}^+ + w_{jk}^+ + w_{ik}^-}{3}\,. \tag{4.20}$$

We consider two different cases to show the result.

Case 1: $w_{ik} \leq 1$**.** In this case, note that even though weights do not satisfy probability constraints, we have

$$\begin{aligned} w_{ij}^- &\geq 1 - w_{ij}^+ = 1 - w_{ij} \in [0, 1] \\ w_{jk}^- &\geq 1 - w_{jk}^+ = 1 - w_{jk} \in [0, 1] \\ w_{ik}^+ &\geq 1 - w_{ik}^- = 1 - w_{ik} \in [0, 1]. \end{aligned}$$

By using the bounds given in (4.17)-(4.19) to lower bound the left hand side of (4.20), we see that it is enough to show that

$$w_{ij}x_{ij} + (1 - w_{ij})(1 - x_{ij}) + w_{jk}x_{jk} + (1 - w_{jk})(1 - x_{jk}) + (1 - w_{ik})x_{ik} + w_{ik}(1 - x_{ik}) \geq \frac{w_{ij} + w_{jk} + w_{ik}}{3}$$

which, when rearranged, is exactly inequality (4.12) shown in Lemma 4.2.3. For the case we are currently considering, all the variables $w_{ij}, w_{jk}, w_{ik}, x_{ij}, x_{jk}$ and x_{ik} satisfy the assumptions of Lemma 4.2.3, so the desired inequality holds.

Case 2: $w_{ik} > 1$. In this case it is not useful to employ the bound $w_{ik}^+ \geq 1 - w_{ik}^-$ because w_{ik}^- is larger than 1, and we already know every weight is nonnegative. For this case we observe that inequality (4.20), which we are trying to show, holds if we can prove that

$$w_{ij}x_{ij} + (1-w_{ij})(1-x_{ij}) + w_{jk}x_{jk} + (1-w_{jk})(1-x_{jk}) + w_{ik}(1-x_{ik}) \geq \frac{w_{ij}+w_{jk}+w_{ik}}{3}. \quad (4.21)$$

The left hand side of inequality (4.21) is a lower bound on the left side of (4.20), obtained by applying bounds $w_{ij}^- \geq 1 - w_{ij}^+ = 1 - w_{ij}$ and $w_{jk}^- \geq 1 - w_{jk}^+ = 1 - w_{jk}$, and dropping the non-negative term $w_{ik}^+ x_{ik}$. By simply rearranging terms we see that this new inequality (4.21) is true if an only if

$$2 - \frac{4}{3}w_{ij} - x_{ij} + 2w_{ij}x_{ij} - \frac{4}{3}w_{jk} - x_{jk} + 2w_{jk}x_{jk} + w_{ik}\left(\frac{2}{3} - x_{ik}\right) \geq 0 \qquad (4.22)$$

Since $x_{ik} \leq x_{ij} + x_{jk} \leq 2/3$, we see that $(2/3 - x_{ik}) \geq 0 \implies w_{ik}(2/3 - x_{ik}) \geq (2/3 - x_{ik})$, so (4.22) is true as long as

$$2 - \frac{4}{3}w_{ij} - x_{ij} + 2w_{ij}x_{ij} - \frac{4}{3}w_{jk} - x_{jk} + 2w_{jk}x_{jk} + \left(\frac{2}{3} - x_{ik}\right) \geq 0 \qquad (4.23)$$

We conclude the proof by noting that this is exactly inequality (4.13) given in Lemma 4.2.3, which holds since the variables $x_{ij}, x_{jk}, x_{ik}, w_{ij}, w_{jk}$ which we consider here all satisfy the assumptions of Lemma 4.2.3.

■

4.3 LambdaCC Approximation Algorithms

An approximation algorithm for LambdaCC can be obtained as a corollary of Theorem 4.2.4.

Corollary 4.3.1 *Applying Algorithm 4 when $\beta = 1/3$ will return a 3-approximation for the standard LambdaCC objective when $\lambda \geq 1/2$.*

Proof For any $\lambda \geq 1/2$, a scaled version of LambdaCC (obtained by dividing all edge weights by $(1-\lambda)$) is a special case of the heavily negative weighted correlation clustering problem. ■

For completeness we also include a self-contained proof of the 3-approximation for LambdaCC by directly applying Theorem 4.2.1. One benefit of this proof is that we will be able to directly extend it to obtain parameter-dependent approximation results for $\lambda < 1/2$. We give explicit pseudocode for the 3-approximation for LambdaCC in Algorithm 5. We will refer to this as ThreeLP. The main difference between the ThreeLP rounding approach and the algorithm of van Zuylen and Williamson [2009] for unweighted correlation clustering is that the threshold for determining whether an edge is positive or negative in $\tilde{G}$ is 1/3 instead of 1/2. In order to prove this is a three-approximation for LambdaCC when $\lambda \geq 1/2$, we need to check that Theorem 4.2.1 holds when $\alpha = 3$. In order to prove this we'll frequently make use of the following facts:

Algorithm 5 ThreeLP

```
Input: Complete signed graph $G = (V, E^+, E^-)$, with positive edges weighted $(1-\lambda)$, negative edges weighted $\lambda$, for $\lambda \in (0,1)$.
Output: Clustering $\mathcal{C}$ of $G$
1. Solve the LP-relaxation of LambdaCC:
       minimize     $\sum_{(i,j)\in E}(1-\lambda)x_{ij} + \sum_{(i,j)\in E}\lambda(1-x_{ij})$
       subject to   $x_{ij} \leq x_{ik} + x_{jk}$      for all $i, j, k$
                    $0 \leq x_{ij} \leq 1$             for all $i < j$.
2. Define $\tilde{G} = (V, F^+, F^-)$ where
       $F^+ = \{(i,j) : x_{ij} < 1/3\}, \quad F^- = \{(i,j) : x_{ij} \geq 1/3\}$
3. Return CC-Pivot($\tilde{G}$).
```

Lemma 4.3.2 *Consider three nodes* $\{i,j,k\}$ *which form a bad triangle in* $\tilde{G}$ *with* $(i,k) \in F^-$, $(i,j) \in F^+$, *and* $(j,k) \in F^+$. *Let* (a,b) *be an arbitrary edge in* G, *and let* c_{ab} *be the LP budget for this edge. Then*

(a) $x_{ij} < \frac{1}{3}$ *and* $x_{jk} < \frac{1}{3}$

(b) $\frac{1}{3} \leq x_{ik} \leq x_{ij} + x_{jk} < \frac{2}{3}$

(c) If $(a,b) \in E^+$, *then* $c_{ab} = (1-\lambda)x_{ab}$, $(w^+_{ab}, w^-_{ab}) = (1-\lambda, 0)$

(d) If $(a,b) \in E^-$, *then* $c_{ab} = \lambda(1-x_{ab})$, $(w^+_{ab}, w^-_{ab}) = (0, \lambda)$.

Proof Statement (a) is just the definition of F^+. Statement (b) is the definition of F^- combined with triangle inequality. Statements (c) and (d) are just reminders of what the "budget" c_{ab} and edge weights (w^+_{ab}, w^-_{ab}) are for positive and negative edges in G. ■

Theorem 4.3.3 *Algorithm* ThreeLP *satisfies Theorem 4.2.1 with* $\alpha = 3$ *when* $\lambda > 1/2$, *so this gives a 3-approximation for LambdaCC for any resolution parameter above 1/2.*

Proof The first two inequalities we need to check for Theorem 4.2.1 are

$$w^-_{ij} \leq \alpha c_{ij} \text{ for all } (i,j) \in F^+$$

$$w^+_{ij} \leq \alpha c_{ij} \text{ for all } (i,j) \in F^-$$

If $(i,j) \in F^+$ but $(i,j) \in E^+$, then $w_{ij}^- = 0$ and the proof is trivial. The result is also trivial when $(i,j) \in F^- \cap E^-$. So first assume that $(i,j) \in F^+ \cap E^-$. Then $w_{ij}^- = \lambda$ and $c_{ij} = \lambda(1-x_{ij})$, and we know $x_{ij} < 1/3 \implies (1-x_{ij}) > 2/3$. Therefore:

$$w_{ij}^- = \lambda < 3\lambda\left(\frac{2}{3}\right) < 3\lambda(1-x_{ij}) = \alpha c_{ij}.$$

On the other hand, if $(i,j) \in F^- \cap E^+$, then $w_{ij}^+ = (1-\lambda)$, $c_{ij} = (1-\lambda)x_{ij}$, and $x_{ij} \geq 1/3$, so we see:

$$w_{ij}^+ = (1-\lambda) = 3(1-\lambda)\left(\frac{1}{3}\right) \leq 3(1-\lambda)x_{ij} = \alpha c_{ij}.$$

The next thing we need to do is check that the inequality

$$w_{ij}^+ + w_{jk}^+ + w_{ik}^- \leq \alpha\left(c_{ij} + c_{jk} + c_{ik}\right) \tag{4.24}$$

holds when $\alpha = 3$ for every bad triangle $\{i,j,k\}$ in $\tilde{G}$ where $(i,j),(j,k) \in F^+$, $(i,k) \in F^-$. Checking this is somewhat tedious but simple. We can prove it case by case, by considering any possible combination of edge types linking nodes $\{i,j,k\}$ in G. There are 8 possibilities for types of triangles in G that are mapped to a bad triangle in $\tilde{G}$, illustrated in Figure 4.1. We then provide a different figure for each case, (Figures 4.2, 4.3, 4.4, 4.5, 4.6, 4.7) with a reminder for what the weights and budgets are for the case and a short proof that the desired inequality holds when $\alpha = 3$. ■

4.3.1 Parameter Dependent Guarantees for LambdaCC

Most of the cases shown in the proof of Theorem 4.3.3 hold independent of the value of λ. However, the proofs for cases 1 and 4 make explicit use of the fact that $\lambda \geq 1/2$. For $\lambda < 1/2$, it is much more challenging to obtain approximation guarantees. In fact, we will later show that in the worst case there is an $\Omega(\log n)$ integrality gap, thus a constant factor approximation for all λ based on LP rounding is impossible. Nevertheless, we can slightly adapt Theorem 4.3.3 to obtain parameter-dependent approximations for standard LambdaCC, which in certain parameter regimes are still better than applying the $O(\log n)$ approximation that holds for any weighted version of correlation clustering [Charikar et al., 2005, Demaine et al., 2006].

Theorem 4.3.4 *Applying Algorithm 5 when $\lambda < 1/2$ will yield an α-approximation for the problem, where $\alpha = \max\{\frac{1}{\lambda}, \frac{(6-3\lambda)}{(1+\lambda)}\} > 3$.*

Proof In Theorem 4.3.3, the inequalities $w_{ij}^- \leq \alpha c_{ij}$ for all $(i,j) \in F^+$, and $w_{i,j}^+ \leq \alpha c_{ij}$ for all $(i,j) \in F^-$, both hold independent of λ. The other inequality we must ensure is in order to apply Theorem 4.2.1 is

$$w_{ij}^+ + w_{jk}^+ + w_{ik}^- \leq \alpha(c_{ij} + c_{jk} + c_{ik}) \tag{4.25}$$

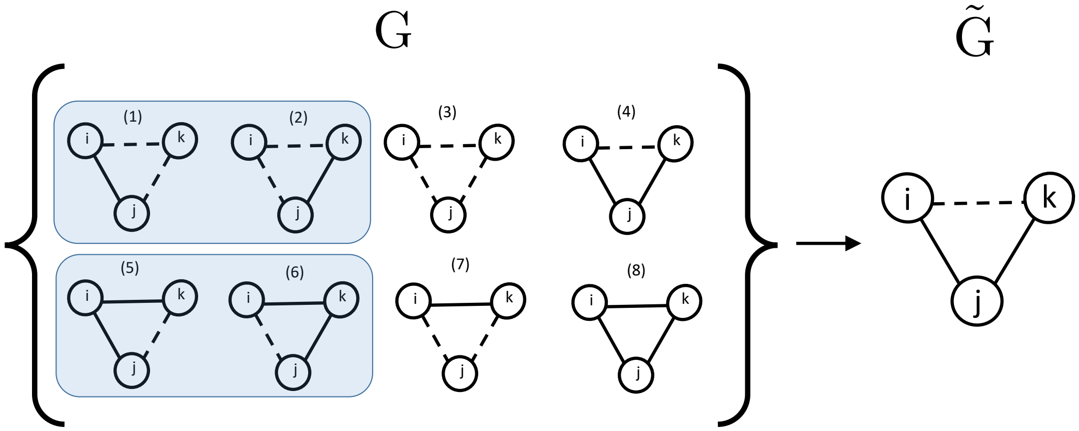
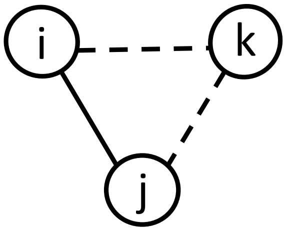


Figure 4.1.: We illustrate the 8 different types of triangles in the original graph G that could be mapped to a bad triangle $\{i, j, k\}$ in $\tilde{G}$ with two positive edges $\{(i, j), (j, k)\} \subset F^+$ and a negative edge $(i, k) \in F^-$. Negative edges are shown with dashed lines. We must check in each case that the inequality (4.24) holds. Cases 1-2 can be handled together, as can cases 5-6, due to the symmetric relationship between (i, j) and (j, k), the two positive edges in the bad triangle in $\tilde{G}$.



Case 1

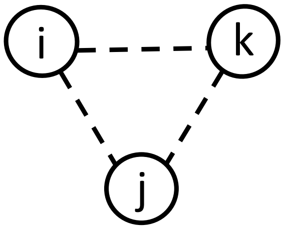
$$
\begin{aligned}
c_{ij} &= (1-\lambda)x_{ij} \\
c_{jk} &= \lambda(1-x_{jk}) \\
c_{ik} &= \lambda(1-x_{ik}) \\
w^+_{ij} &= (1-\lambda) \\
w^+_{jk} &= 0 \\
w^-_{ik} &= \lambda
\end{aligned}
$$

Weights and budgets

$$
\begin{aligned}
&\alpha(c_{ij} + c_{jk} + c_{ik}) \\
&\quad = 3\left((1-\lambda)x_{ij} + \lambda(1-x_{jk}) + \lambda(1-x_{ik})\right) \\
&\quad \geq 3\left(\lambda - \lambda x_{jk} + \lambda - \lambda x_{ik}\right) \\
&\quad \geq 3\left(\lambda - \lambda/3 + \lambda - 2\lambda/3\right) \\
&\quad = 3\lambda \geq 1 \\
&\quad = w^+_{ij} + w^+_{jk} + w^-_{ik}
\end{aligned}
$$

Case 1 inequality proof

Figure 4.2.: Case 1: We explicitly use the fact that $\lambda > 1/2$ here. The same set of steps works for Case 2, if we switch the roles of edges (i, j) and (j, k). This is the case that will make the proof fail if we deal with arbitrarily small λ, indicating as before that LambdaCC for small λ is somehow more challenging than large λ.



Case 3

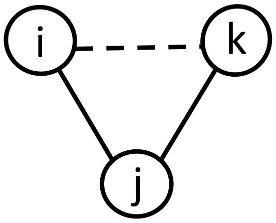
$$
\begin{aligned}
c_{ij} &= \lambda(1 - x_{ij})\\
c_{jk} &= \lambda(1 - x_{jk})\\
c_{ik} &= \lambda(1 - x_{ik})\\
w_{ij}^{+} &= 0\\
w_{jk}^{+} &= 0\\
w_{ik}^{-} &= \lambda
\end{aligned}
$$

Weights and budgets

$$
\begin{aligned}
&\alpha(c_{ij} + c_{jk} + c_{ik})\\
&\qquad = 3(\lambda(1 - x_{ij}) + \lambda(1 - x_{jk}) + \lambda(1 - x_{ik}))\\
&\qquad = 3\lambda(3 - x_{ij} - x_{jk} - x_{ik})\\
&\qquad > 3\lambda(3 - 1/3 - 1/3 - 2/3)\\
&\qquad = 3\lambda(5/3)\\
&\qquad = 5\lambda\\
&\qquad > \lambda = 0 + 0 + \lambda = w_{ij}^{+} + w_{jk}^{+} + w_{ik}^{-}
\end{aligned}
$$

Inequality proof

Figure 4.3.: Case 3: When all edges of the triangle in G are negative, the bound is very loose since the left hand side of inequality (4.24) is just λ and we can easily bound the right hand side below.



Case 4

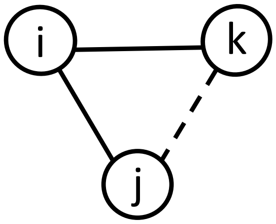
$$
\begin{aligned}
c_{ij} &= (1 - \lambda)x_{ij}\\
c_{jk} &= (1 - \lambda)x_{jk}\\
c_{ik} &= \lambda(1 - x_{ik})\\
w_{ij}^{+} &= (1 - \lambda)\\
w_{jk}^{+} &= (1 - \lambda)\\
w_{ik}^{-} &= \lambda
\end{aligned}
$$

Weights and budgets

$$
\begin{aligned}
&\alpha(c_{ij} + c_{jk} + c_{ik})\\
&\qquad = 3\left((1 - \lambda)(x_{ij} + x_{jk}) + \lambda(1 - x_{ik})\right)\\
&\qquad \geq 3\left((1 - \lambda)x_{ik} + \lambda(1 - x_{ik})\right)\\
&\qquad = 3\left((1 - 2\lambda)x_{ik} + \lambda\right)\\
&\qquad > 3\left((1 - 2\lambda)\frac{2}{3} + \lambda\right)\\
&\qquad = 2 - \lambda = w_{ij}^{+} + w_{jk}^{+} + w_{ik}^{-}
\end{aligned}
$$

Inequality Proof

Figure 4.4.: Case 4: If the same type edges in G map to the same type edges in $\tilde{G}$, the inequality we are trying to prove is almost tight (if $x_{ik} = x_{jk} + x_{ij}$ and the variables are near their upper bound), and we must make explicit use of the fact that $\lambda > 1/2$.



Case 5

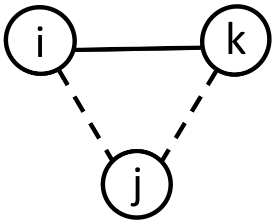
$$c_{ij} = (1-\lambda)x_{ij}$$
$$c_{jk} = \lambda(1-x_{jk})$$
$$c_{ik} = (1-\lambda)x_{ik}$$
$$w^+_{ij} = (1-\lambda)$$
$$w^+_{jk} = 0$$
$$w^-_{ik} = 0$$

Weights and budgets

$$\begin{aligned}&\alpha(c_{ij}+c_{jk}+c_{ik})\\ &\quad\geq 3(c_{ik}) \geq 3(1-\lambda)x_{ik}\\ &\quad\geq 3(1-\lambda)\frac{1}{3}\\ &\quad= (1-\lambda)\\ &\quad= w^+_{ij}+w^+_{jk}+w^-_{ik}\end{aligned}$$

Inequality Proof

Figure 4.5.: Case 5: This case is easy to show, independent of λ. Exactly the same proof holds for case 6.



Case 7

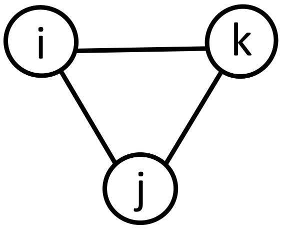
$$c_{ij} = \lambda(1-x_{ij})$$
$$c_{jk} = \lambda(1-x_{jk})$$
$$c_{ik} = (1-\lambda)x_{ik}$$
$$w^+_{ij} = 0$$
$$w^+_{jk} = 0$$
$$w^-_{ik} = 0$$

Weights and budgets

$$\begin{aligned}&\alpha(c_{ij}+c_{jk}+c_{ik})\\ &\quad\geq 0\\ &\quad= w^+_{ij}+w^+_{jk}+w^-_{ik}\end{aligned}$$

Inequality Proof

Figure 4.6.: Case 7: The left hand side is zero, so the proof is trivial



Case 8

$$c_{ij} = (1-\lambda)x_{ij}$$
$$c_{jk} = (1-\lambda)x_{jk}$$
$$c_{ik} = (1-\lambda)x_{ik}$$
$$w_{ij}^{+} = (1-\lambda)$$
$$w_{jk}^{+} = (1-\lambda)$$
$$w_{ik}^{-} = 0$$

Weights and budgets

$$\begin{aligned}
&\alpha(c_{ij}+c_{jk}+c_{ik}) \\
&\qquad = 3(1-\lambda)(x_{ij}+x_{jk}+x_{ik}) \\
&\qquad \geq 3(1-\lambda)(2/3) \\
&\qquad = 2(1-\lambda) \\
&\qquad = w_{ij}^{+} + w_{jk}^{+} + w_{ik}^{-}
\end{aligned}$$

Inequality Proof

Figure 4.7.: Case 8: The key for this case is to use the fact that $1/3 \leq x_{ik} \leq x_{ij} + x_{jk}$.

for every triplet $\{i,j,k\}$ such that $(i,j) \in F^{+}, (j,k) \in F^{+}$ and $(i,k) \in F^{-}$. As mentioned, most of the proof cases from Theorem 4.3.3 for this inequality still directly apply here with $\alpha = 3$, independent of the value of λ. However, when $\lambda < 1/2$, the proof of Case 1 (Figure 4.2) and the proof for Case 4 (Figure 4.4) no longer hold. We re-address these cases here.

Case 1: $(i,j) \in E^{+}, (j,k) \in E^{-}$ and $(i,k) \in E^{-}$. In this case, the LP costs are $(c_{ij}, c_{jk}, c_{ik}) = ((1-\lambda)x_{ij}, \lambda(1-x_{jk}), \lambda(1-x_{ik}))$ and the weights are $(w_{ij}^{+}, w_{jk}^{+}, w_{ik}^{-}) = (1-\lambda, 0, \lambda)$. Therefore,

$$\begin{aligned}
\alpha(c_{ij}+c_{jk}+c_{ik}) &= \alpha\left((1-\lambda)x_{ij} + \lambda(1-x_{jk}) + \lambda(1-x_{ik})\right) \\
&\geq \alpha\left(\lambda - \lambda x_{jk} + \lambda - \lambda x_{ik}\right) \geq \alpha\left(\lambda - \lambda/3 + \lambda - 2\lambda/3\right) \\
&= \alpha\lambda \geq 1 = w_{ij}^{+} + w_{jk}^{+} + w_{ik}^{-}\,,
\end{aligned}$$

which holds as long as $\alpha \geq 1/\lambda$. Note that this also accounts for Case 3 ($(i,j) \in E^{-}, (j,k) \in E^{+}$ and $(i,k) \in E^{-}$), which is symmetric.

Case 4: $(i,j) \in E^{+}, (j,k) \in E^{+}$ and $(i,k) \in E^{-}$. Given these types of edges in the original graph, we know that $c_{ij} = (1-\lambda)x_{ij}$, $c_{jk} = (1-\lambda)x_{ik}$, and $c_{ik} = \lambda(1-x_{ik})$. Therefore,

$$\begin{aligned}
\alpha(c_{ij}+c_{jk}+c_{ik}) &= \alpha\left((1-\lambda)(x_{ij}+x_{jk}) + \lambda(1-x_{ik})\right) \\
&\geq \alpha\left((1-\lambda)x_{ik} + \lambda(1-x_{ik})\right) = \alpha\left((1-2\lambda)x_{ik} + \lambda\right) \\
&> \alpha\left((1-2\lambda)1/3 + \lambda\right) = \alpha(1+\lambda)/3\,.
\end{aligned}$$

The weights satisfy $w_{ij}^{+} = w_{jk}^{+} = 1-\lambda$ and $w_{ik}^{-} = \lambda$. With a few steps of algebra we can see that the above expression is an upper bound for $2-\lambda = w_{ij}^{+} + w_{jk}^{+} + w_{ik}^{-}$ (the right hand side of inequality (4.25)) as long as $\alpha \geq \frac{6-3\lambda}{1+\lambda}$.

Given these two updated cases, and the fact that all other cases hold with $\alpha = 3$, we see that if $\alpha = \max\left\{\frac{1}{\lambda}, \frac{6-3\lambda}{1+\lambda}\right\}$, the full result holds. ■

By solving $1/\lambda = (6 - 3\lambda)/(1 + \lambda)$ for λ, we find that the behavior of the approximation factor changes when $\lambda = (5 - \sqrt{13})/6 \approx 0.2324$. For λ greater than this threshold, the approximation factor is always between 3 and 4.303. Thus, we can obtain an approximation better than 4.5 for all $\lambda \in (0.2324, 0.5)$, but the algorithm performs worse and worse as λ decreases. This approximation is especially bad for values of λ near (but just larger than) the minimum scaled sparsity score of a network. Even for special classes of expander graphs where the sparsest cut score is a constant (e.g., the expander graphs in the work of Reingold et al. [2001]), the minimum *scaled* sparsity (2.3) is inversely proportional to the number of nodes in the graph, and the approximation factor would be $1/\lambda = O(n)$. In fact, we will use these expander graphs to show that the integrality gap of the LP relaxation is $\Omega(\log n)$. Nevertheless, for all λ in $\omega(1/\log n)$, our algorithm outputs a better result than the standard, $O(\log n)$, rounding scheme for general weighted correlation clustering [Demaine et al., 2006].

4.3.2 Two-Approximation for Cluster Deletion

For cluster deletion we can obtain an even better approximation algorithm by changing the algorithm slightly and and showing that Theorem 4.2.1 holds with $\alpha = 2$. Since in this chapter we are focused on special weighted versions of correlation clustering, for consistency we treat cluster deletion as an objective on signed graphs. Thus, the goal of cluster deletion is to minimize the number of positive mistakes made, while strictly forbidding negative edge mistakes. Pseudocode for the new procedure, which we call TwoCD, is given in Algorithm 6.

Note that because $x_{ij} = 1$ for $(i, j) \in E^-$, step 2 in Algorithm 6 is essentially just choosing a subset of positive edges to turn into negative edges. Therefore, $F^+ \subseteq E^+$ and $E^- \subseteq F^-$. We observe that this returns a feasible clustering for cluster deletion:

Lemma 4.3.5 TwoCD *will not cluster nodes i and j together if $(i, j) \in E^-$.*

Proof If $(i, j) \in E^-$, then $x_{ij} = 1$ and $(i, j) \in F^-$, so we would never cluster them together using CC-Pivot on $\tilde{G}$ if i or j were chosen as a pivot. Assume then that some k is chosen as pivot such that $(i, k) \in F^+$ and $(j, k) \in F^+$. This would have to mean that both x_{ik} and x_{jk} are strictly less than 1/2, which leads to a contradiction because we would have

$$(i, j) \in E^- \implies x_{ij} = 1 \leq x_{ik} + x_{jk} < 1$$

where we have applied the triangle inequality constraint, which must hold. ■

```
Algorithm 6 TwoCD
    Input: Complete signed graph $G = (V, E^+, E^-)$
    Output: Clustering $\mathcal{C}$ of $G$ with no negative edge mistakes.
    1. Solve the LP-relaxation of cluster deletion:
           minimize    $\sum_{(i,j) \in E^+} x_{ij}$
           subject to  $x_{ij} \leq x_{ik} + x_{jk}$ for all $i, j, k$
                       $x_{ij} \in [0, 1]$ for all (i,j) $\in E^+$
                       $x_{ij} = 1$ for all (i,j) $\in E^-$
    2. Define $\tilde{G} = (V, F^+, F^-)$ where
           $F^+ = \{(i,j) : x_{ij} < 1/2\}, \quad F^- = \{(i,j) : x_{ij} \geq 1/2\}$
    3. Return CC-Pivot($\tilde{G}$)
```

The lemma above proves that TwoCD gives a valid clustering for cluster deletion. The next theorem shows this clustering is within a factor 2 of optimal:

Theorem 4.3.6 *Theorem 4.2.1 holds for the input graph $G = (V, E^+, E^-)$ and constructed graph $\tilde{G} = (V, F^+, F^-)$ with $\alpha = 2$, so* TwoCD *returns a 2-approximation for cluster deletion.*

Proof Recall that for cluster deletion the edge weights and LP budget for pair (i, j) are given by

$$(w_{ij}^+, w_{ij}^-) = (1, 0) \text{ for } (i,j) \in E^+$$

$$(w_{ij}^+, w_{ij}^-) = (0, 1) \text{ for } (i,j) \in E^-$$

$$c_{ij} = \begin{cases} x_{ij} & \text{if } (i,j) \in E^+ \\ 0 & \text{otherwise.} \end{cases}$$

We just need to confirm that the assumptions of Theorem 4.2.1 hold:

(1) $w_{ij}^- \leq \alpha c_{ij}$ for all $(i,j) \in F^+$ and

$w_{ij}^+ \leq \alpha c_{ij}$ for all $(i,j) \in F^-$,

(2) $w_{ij}^+ + w_{jk}^+ + w_{ik}^- \leq \alpha \left(c_{ij} + c_{jk} + c_{ik} \right)$

for every bad triangle in $\tilde{G}$: $(i,j), (j,k) \in F^+$, $(i,k) \in F^-$.

For cluster deletion, first note that

$$(i,j) \in F^+ \implies (i,j) \in E^+ \implies w_{ij}^- = 0 \leq 2c_{ij}.$$

Similarly, if $(i,j) \in F^-$ and $(i,j) \in E^-$, we know that $w_{ij}^+ = 0$ so $w_{ij}^+ \leq 2c_{ij}$ is again trivial. To finish the proof of part (1), note that if $(i,j) \in F^-$ but $(i,j) \in E^+$, this means that $x_{ij} \geq 1/2$, $w_{ij}^+ = 1$, and $c_{ij} = x_{ij}$, so

$$w_{ij}^+ = 1 = 2(1/2) \leq 2x_{ij} = 2c_{ij}.$$

Now we must check that (2) holds. The key is to realize that for every bad triangle $\{i,j,k\}$ in $\tilde{G} = (V, F^+, F^-)$ (where $(i,k) \in F^-$ and the other edges are positive), $\{i,j,k\}$ forms a triangle of nodes that all have positive edges in G. This is because in order for (i,j) and (j,k) to be in F^+, it must be the case that $x_{ij} < 1/2$ and $x_{jk} < 1/2$, so by the triangle inequality we see

$$x_{ik} \leq x_{ij} + x_{jk} < 1.$$

Since all these distances are strictly less than one, all the edges are positive in the original graph G. Therefore, for this bad triangle we know $c_{ij} = x_{ij}$, $c_{jk} = x_{jk}$, $c_{ik} = x_{ik}$, and $w_{ij}^+ = w_{jk}^+ = w_{ik}^+ = 1$. Also, $w_{ik}^- = 0$. By the construction of F^-, F^+ we know

$$x_{ij} < \frac{1}{2}, x_{jk} < \frac{1}{2}, x_{ik} \geq \frac{1}{2},$$

and then applying triangle inequality we have

$$\frac{1}{2} \leq x_{ik} \leq x_{ij} + x_{jk} \implies x_{ij} + x_{jk} + x_{ik} \geq 1.$$

The desired inequality is satisfied because

$$w_{ij}^+ + w_{jk}^+ + w_{ik}^- = 2 = 2(1) \leq 2\left(x_{ij} + x_{jk} + x_{ik}\right) = \alpha\left(c_{ij} + c_{jk} + c_{ik}\right).$$

Therefore, Theorem 4.2.1 guarantees that CC-Pivot will output a clustering that costs at most

$$2\sum_{i<j} c_{ij} = 2 \sum_{(i,j)\in E^+} x_{ij}$$

so this is a 2-approximation for cluster deletion. ■

Although cluster deletion was not explicitly mentioned in their work, van Zuylen and Williamson [2009] showed a 3-approximation for *constrained* correlation clustering which can be directly applied to get a 3-approximation for cluster deletion. Prior to TwoCD, this 3-approximation can be viewed as the previous best approximation factor for the problem. Until now, the best approximation factors for unweighted correlation clustering, culminating in the 2.06 approximation given by Charikar et al. [2005], have been better than the known approximation factors for cluster deletion. Our result indicates for the first time that although far fewer approximation algorithms for cluster deletion have been developed, it is in a sense an easier problem to approximate.

Integrality Gap for Cluster Deletion The LP relaxation of cluster deletion, shown in Algorithm 6, has an integrality gap of 2. This is the same as the gap for unweighted correlation clustering, and is shown using the same graph example used by Charikar et al. [2005]. Consider an $(n+1)$-node star graph, where the first node shares a (positive) edge with all other nodes, but all other edges are negative (or if we define the problem on an unsigned graph, all other nodes share no edge). The optimal clustering will delete $n-1$ (positive) edges, by placing the first node with one neighbor and making all other nodes singletons. The LP relaxation is constrained so that the distance is $x_{ij} = 1$ whenever $i \neq 1$ and $j \neq 1$. The optimal LP solution will assign distance $x_{1i} = 0.5$ from the first node to every other node, leading to an LP lower bound of $n/2$. The ratio between the optimal solution and the LP bound is $2(n-1)/n$, which goes to 2 as n increases. Furthermore, for star graphs, the rounding procedure in Algorithm 6 will be exactly two times the LP lower bound, regardless of n. Thus our approximation factor for cluster deletion is tight.

4.4 Standard LambdaCC Integrality Gap

Demaine et al. [2006] prove that the integrality gap for the general weighted correlation clustering LP relaxation is $\Omega(\log n)$. This does not immediately imply anything for our specially weighted case, but adapting some of their ideas, which includes a few non-trivial steps, does reveal that in the worst case, there is an $\Omega(\log n)$ integrality gap for the standard LambdaCC linear programming relaxation:

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in E^+}(1-\lambda)x_{ij} + & \sum_{(i,j)\in E^-} \lambda(1-x_{ij}) \\ \text{subject to} & x_{ij} \leq x_{ik} + x_{jk} & \text{for all } i,j,k \\ & 0 \leq x_{ij} \leq 1 & \text{for all } i<j. \end{array} \tag{4.26}$$

The proof takes the following steps.

1. Construct an instance of LambdaCC from an expander graph, G.
2. Prove that, because of the expander properties of G, the optimal LambdaCC clustering must make $\Omega(n)$ mistakes.
3. Demonstrate the LP relaxation has a feasible solution with a score of $O(n/\log n)$.

In order to accomplish third step listed above, we do not (necessarily) produce a feasible solution for the standard LP relaxation of LambdaCC: in particular, in our solution triangle constraints are not guaranteed. Instead, we produce a feasible solution for a related linear program considered by Wirth [2004] in his PhD thesis. The fundamental construct of this LP is the NEGATIVE EDGE WITH POSITIVE PATH CYCLE (NEPPC), where, *NEPPC*$(i_1, i_2, \dots, i_m)$ represents a sequence (a *path*) of (positive) edges, $(i_1, i_2), (i_2, i_3), \dots, (i_{m-1}, i_m) \in E$, with a single (negative) non-edge completing the cycle: $(i_1, i_m) \notin E$. For LambdaCC, defined on a graph $G = (V, E)$, with parameter $\lambda \in (0, 1)$, we have the linear program:

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in E}(1-\lambda)x_{ij} + & \sum_{(i,j)\notin E}\lambda(1-x_{ij}) \\ \text{subject to} & x_{i_1,i_m} \leq \sum_{j=1}^{m-1} x_{i_j,i_{j+1}} & \text{for all } NEPPC(i_1, i_2, \ldots, i_m) \\ & x_{ij} \leq 1 & \text{for all } (i,j) \notin E \\ & 0 \leq x_{ij} & \text{for all } (i,j)\,. \end{array} \tag{4.27}$$

Wirth [2004] proved that the set of optimal solutions to the NEPPC linear program (4.27) is exactly the same as the optimal solution set to the correlation clustering LP, the relaxation of ILP (2.12). Although the proof is shown for the unweighted case, we note that all aspects of the proof immediately carry over to the weighted case. A closely related result for weighted graphs was also explicitly proven by Lange et al. [2018]. Since a feasible solution for the LambdaCC NEPPC linear program (4.27) is an upper bound on the optimum for (4.27), which is the same as the optimum for the standard LambdaCC LP, we can bound the optimum of the latter. We now prove our result:

Theorem 4.4.1 *There exists some λ such that the integrality gap of LP (4.26) is $O(\log n)$.*

Proof We separate the proof into the three steps outlined at the beginning of the section: constructing a LambdaCC instance from an expander graph, bounding the LambdaCC solution from below, and then upper bounding the LP relaxation.

Constructing an instance of LambdaCC from an expander Let $G = (V, E)$ be a (d, c)-expander graph, where both d and c are constants (Reingold et al. proved that such expanders exist Reingold et al. [2000]). That is, G is d-regular, and for every $S \subset V$ with $|S| \leq n/2$, we have

$$\frac{\text{cut}(S)}{|S|} \geq c \implies \frac{\text{cut}(S)}{|S|} + \frac{\text{cut}(S)}{|\bar{S}|} \geq c \implies \frac{\text{cut}(S)}{|S||\bar{S}|} \geq \frac{c}{n}$$

where $\text{cut}(S)$ denotes the number of edges between S and $\bar{S} = V \backslash S$. Define the *scaled cut sparsity* of a set S to be $\text{cut}(S)/(|S||\bar{S}|)$ and let λ^* minimize this ratio over all possible sets $S \subset V$. In Theorem 3.4.2, we showed that for any $\lambda \leq \lambda^*$, the optimal LambdaCC clustering places all nodes into one cluster, but there exists a range of λ values slightly larger than λ^* such that the optimum clustering coincides with a partitioning that produces the scaled sparsest cut score. For the expander graph we consider, this λ^* is at most the scaled sparsest cut score obtained by setting S to be a single node, so we have these upper and lower bounds on λ^*: $c/n \leq \lambda^* \leq d/(n-1)$.

Let S^* be a set inducing an optimal scaled sparsest cut partition: $\lambda^* = \text{cut}(S^*)/(|S^*||\bar{S^*}|)$. From Theorem 3.4.2, we know that there exists some λ', slightly larger than λ^* whose optimum LambdaCC solution is the bipartition $\{S^*, \bar{S^*}\}$; let the LambdaCC score of this solution be OPT, and let $\varepsilon = \lambda' - \lambda^*$. We can choose $\varepsilon > 0$ to be arbitrarily small, so it suffices to assume $\lambda' < 2\lambda^*$.

Bounding *OPT* from below With our choice of λ', by definition,

$$\begin{aligned}
OPT &= \text{cut}(S^*) - \lambda'|S^*||\bar{S}^*| + \lambda'\left(\binom{n}{2} - |E|\right) \\
&= \text{cut}(S^*) - \lambda^*|S^*||\bar{S}^*| - \varepsilon|S^*||\bar{S}^*| + \lambda'\left(\binom{n}{2} - |E|\right) \\
&= 0 - \varepsilon|S^*||\bar{S}^*| + \lambda^*\left(\binom{n}{2} - |E|\right) + \varepsilon\left(\binom{n}{2} - |E|\right) \\
&= \lambda^*\left(\binom{n}{2} - |E|\right) + \varepsilon\left(\binom{n}{2} - |E| - |S^*||\bar{S}^*|\right) \\
&= \lambda^*\left(\frac{n(n-1)}{2} - \frac{nd}{2}\right) + \varepsilon\left(\frac{n(n-1)}{2} - \frac{nd}{2} - |S^*||\bar{S}^*|\right) \\
&\geq \lambda^*\left(\frac{n(n-1)}{2} - \frac{nd}{2}\right) + \varepsilon\left(\frac{n(n-1)}{2} - \frac{nd}{2} - \frac{n^2}{4}\right) \\
&\geq \frac{c}{n}\left(\frac{n(n-1)}{2} - \frac{nd}{2}\right) = \Omega(n)\,,
\end{aligned}$$

relying on the definition of λ^*, the fact that $|E| = nd/2$ in this expander graph, and the bound $|S^*||\bar{S}^*| \leq n^2/4$.

Upper bounding the NEPPC LP We now show that a carefully crafted feasible solution for the NEPPC LP (4.27) has score $O(n/\log n)$. Let $dist(i,j)$ denote the minimum path length between nodes i and j in G, based on unit-weight edges E. We are assuming the graph is connected, so each $dist(i,j)$ is a finite integer. (If the graph is not connected, we ought to solve LambdaCC on each connected component separately.) Consider the following setting of values x_{ij}:

$$x_{ij} = \begin{cases} 2/(\log_d n) & \text{if } (i,j) \in E \\ 1 & \text{if } (i,j) \notin E \text{ and } dist(i,j) \geq (\log_d n)/2 \\ 0 & \text{if } (i,j) \notin E \text{ and } dist(i,j) < (\log_d n)/2\,. \end{cases}$$

We show that this is feasible for the NEPPC LP (4.27). Since all (positive) edges are assigned the same LP score, the NEPPC constraints are satisfied at a (negative) non-edge, (i,j), if and only if $x_{ij} \leq dist(i,j) \cdot 2/(\log_d n)$. When $dist(i,j)$ is less than $\log_d(n)/2$, $x_{ij} = 0$, so this inequality is trivially true. When $dist(i,j)$ is at least $\log_d(n)/2$, the NEPPC inequality is true because $dist(i,j) \cdot 2/(\log_d n)$ is at least 1, which is x_{ij}.

For constant d, the contribution from the (positive) edges to LP (4.27) is:

$$(1-\lambda')|E|2/(\log_d n) = (1-\lambda')(nd)/(\log_d n) = O(n/\log n)\,.$$

From the (negative) non-edges, since the factor is $1 - x_{ij}$, we only have a non-zero contribution from the set of $(i,j) \notin E$ such that $dist(i,j) < (\log_d n)/2 = \log_d \sqrt{n}$. For each node $v \in V$, there are at most $d^{\log_d \sqrt{n}} = \sqrt{n}$ nodes within this distance; the total number of non-edges that contribute to the LP cost is therefore in $O(n\sqrt{n})$. Each has a weight $\lambda' < 2\lambda^*$, so

$$\text{LP contribution of non-edges} \leq \lambda' n\sqrt{n} \leq (2d/(n-1))\, n\sqrt{n} = O(\sqrt{n}) \leq O(n/\log n).$$

Therefore, the total LP cost corresponding to this feasible solution to NEPPC LP (4.27) is $O(n/\log n)$. Since the optimal LambdaCC solution has cost $\Omega(n)$, we have shown that there exists some $\lambda < 1/2$ such that the LP relaxation (4.26) has an integrality gap of $O(\log n)$. ■

5. METRIC CONSTRAINED OPTIMIZATION FOR GRAPH CLUSTERING

5.1 Chapter Overview

This chapter presents a novel technique for solving a class of linear programs that arise frequently in the design of approximation algorithms for graph clustering. As a first example, consider the linear programming relaxation of the LambdaCC objective:

$$\begin{array}{ll} \text{minimize} & \sum_{(i,j)\in E}(1-\lambda)x_{ij} + \sum_{(i,j)\notin E}\lambda(1-x_{ij}) \\ \text{subject to} & x_{ij} \leq x_{ik} + x_{jk} \text{ for all } i,j,k \\ & 0 \leq x_{ij} \leq 1 \text{ for all } i,j \end{array} \tag{5.1}$$

which contains a triangle inequality constraint of the form $x_{ij} \leq x_{ik} + x_{jk}$ for each triplets of distinct nodes $(i,j,k) \in V \times V \times V$. In the previous chapter we showed that solving and rounding this LP leads to a 3-approximation for standard LambdaCC when $\lambda \geq 1/2$. The x_{ij} variables in the LP represent distances between nodes. The triangle inequality constraints embed the nodes of the graph into a metric space (or technically speaking, a pseudo-metric space, since two distinct nodes can have distance zero from one another). Therefore, we refer to this as a *metric constrained* linear program (LP), or more succinctly as a *metric* LP.

Many other variants of correlation clustering, as well as several other NP-hard graph clustering problems, can be approximated by solving and rounding a metric constrained linear programming relaxation [Agarwal and Kempe, 2008, Leighton and Rao, 1999, Trevisan, 2013, Ailon et al., 2008, Veldt et al., 2018b]. These LPs are attractive from a theoretical perspective as they can be solved in polynomial time and can be rounded to produce good output clusterings. However, they are very challenging to solve in practice due to the extreme memory requirement that comes with having a constraint matrix with $O(n^3)$ rows.

After giving several specific examples of metric LPs, we prove that the linear programming relaxation of correlation clustering is equivalent to a special case of a problem called metric nearness [Brickell et al., 2008]. We then adapt and improve previously developed projection methods for the latter problem, in order to develop a general framework for metric constrained optimization that is particularly well-suited to graph clustering applications. Projection methods operate by iteratively visiting constraints and performing orthogonal projections at violated constraints. For metric constrained optimization, these projection steps only change a few distance variables in the LP at once, so they can be performed quickly and efficiently. This approach also avoids the need to explicitly store the extremely large constraint matrix needed to encode triangle inequality constraints. We prove several novel guarantees for using this framework to produce lower bounds and approximate solutions for correlation clustering, sparsest cut, and modularity clustering. In experimental results, we are able to solve optimization problems involving 700 billion constraints. In recent follow-up work, parallelized versions of this method were used to obtain results for problems involving up to 2.9 trillion constraints Ruggles et al. [2019].

An abbreviated version of this chapter has been accepted for publication in the SIAM Journal on Mathematics of Data Science [Veldt et al., 2019a].

5.2 Metric LPs and Graph Clustering

Formally, we define a metric constrained optimization problem to be an optimization problem involving constraints of the form $x_{ij} \leq x_{ik} + x_{jk}$, where x_{ij} is a nonnegative variable representing a distance score between two objects i and j in a given dataset. Optimization problems of this form arise very naturally in the study of graph clustering objectives, since any non-overlapping clustering $\mathcal{C}$ for a graph $G = (V, E)$ is in one-to-one correspondence with a set of binary variables $\mathbf{x} = (x_{ij})$ satisfying triangle inequality constraints:

$$\begin{cases} x_{ij} \in \{0,1\} & \text{for all } i,j \text{ and} \\ x_{ij} \leq x_{ik} + x_{jk} & \text{for all } i,j,k \end{cases} \iff \exists\, \mathcal{C} \text{ s.t. } x_{ij} = \begin{cases} 0 & \text{if } i,j \text{ are together in } \mathcal{C} \\ 1 & \text{otherwise.} \end{cases}$$

In this section, we specifically consider a number of metric constrained *linear* programs, most of which arise as a relaxation of an NP-hard graph clustering task. We also prove an equivalence between the metric nearness problem and the correlation clustering LP (Theorem 5.2.1).

Correlation Clustering In previous chapters we introduced and considered the following metric constrained relaxation of correlation clustering:

$$\begin{array}{lll} \text{minimize} & \sum_{i<j} w_{ij}^{+} x_{ij} + w_{ij}^{-}(1 - x_{ij}) & \\ \text{subject to} & x_{ij} \leq x_{ik} + x_{jk} & \text{for all } i,j,k \\ & 0 \leq x_{ij} \leq 1 & \text{for all } i,j. \end{array} \tag{5.2}$$

With very few exceptions, the best approximation results specifically for minimizing disagreements for special variants of correlation clustering rely on solving and rounding this LP. This includes the $O(\log n)$ approximation for the general weighted case [Charikar et al., 2005, Demaine et al., 2006], the $2.06 - \varepsilon$ approximation for the unweighted, complete case [Chawla et al., 2015], and the best approximation factors for several other special weighted variants [Veldt et al., 2018b, van

Zuylen and Williamson, 2009, Ailon et al., 2008, Puleo and Milenkovic, 2015]. Additionally, the best approximations for k-partite graphs [Chawla et al., 2015], locally bounded errors [Puleo and Milenkovic, 2018, Charikar et al., 2017], and higher-order correlation clustering [Gleich et al., 2018, Fukunaga, 2018, Li et al., 2018], all also rely on solving and rounding such an LP relaxation.

Despite the wealth of techniques that have been developed for *rounding* the the solution to this LP, very little work has addressed how to actually solve the LP efficiently in practice. Wirth [2004] noted that using a multicommodity flow formulation leads to a slightly more efficient approach, but is still rather impractical. Van Gael and Zhu [2007] employed an *LP chunking* technique which allowed them to solve the correlation clustering relaxation on graphs with up to nearly 500 nodes. A number of researchers have developed efficient techniques for solving LP relaxations that are related to, but not exactly the same as (5.2) [Lange et al., 2018, Nowozin and Jegelka, 2009, Swoboda and Andres, 2017]. For example, Lange et al. [2018] and Swoboda and Andres [2017] developed approaches for obtaining lower bounds based on dual LP formulations, which can be rounded using heuristic techniques. However, these approaches cannot be used to implement known algorithms with the best a priori approximation factors. Swoboda and Andres [2017] used their techniques to obtain good lower bounds for sparse correlation clustering problems with up to 4.1 million edges. At the end of this chapter, we will demonstrate numerical results for using our metric constrained optimization framework to obtain results for problems involving up to 63 million edges.

Sparsest Cut Recall that the sparsity of a set $S \subset V$ in an n-node graph $G = (V, E)$ is defined to be

$$\mathbf{scut}(S) = \frac{\text{cut}(S)}{|S|} + \frac{\text{cut}(S)}{|\bar{S}|} = \frac{n\,\text{cut}(S)}{|S||\bar{S}|},$$

where $\bar{S} = V \backslash S$ is the complement of S and cut(S) indicates the number of edges crossing between S and $\bar{S}$. Leighton and Rao [1999] developed an $O(\log n)$-approximation for finding the sparsest cut set for any graph by solving a maximum multicommodity flow problem. This result is equivalent to solving the LP relaxation for the following metric constrained linear program:

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in E} x_{ij} & \\ \text{subject to} & \sum_{i<j} x_{ij} = n & \\ & x_{ij} \le x_{ik} + x_{jk} & \text{for all } i,j,k \\ & x_{ij} \ge 0 & \text{for all } i,j \end{array} \tag{5.3}$$

and rounding the solution into a cut. Trevisan [2013] provides a detailed explanation for how to relax the sparsest cut objective into this metric constrained LP, and how to round it to obtain the $O(\log n)$ approximation guarantee.

Lang and Rao [1993] developed an algorithm closely related to the original Leighton-Rao algorithm, which was later evaluated empirically by Lang et al. [2009]. However, the algorithm only heuristically solves the underlying multicommodity flow problem, and therefore does not satisfy the same theoretical guarantees.

Maximum Modularity Clustering Inspired by the LP rounding procedure of Charikar et al. [2005] for correlation clustering, Agarwal and Kempe [2008] considered the following metric constrained linear programming relaxation of maximum modularity clustering:

$$\begin{array}{lll} \text{maximize} & \frac{1}{2|E|}\sum_{i,j}\left(A_{ij}-\frac{d_i d_j}{2|E|}\right)(1-x_{ij}) & \\ \text{subject to} & x_{ij}\leq x_{ik}+x_{jk} & \text{for all } i,j,k \\ & 0\leq x_{ij}\leq 1 & \text{for all } i,j. \end{array} \tag{5.4}$$

Solving this LP relaxation provides a useful upper bound on the maximum modularity. Since it is NP-hard to approximate modularity to within any constant factor [Dinh et al., 2015], solving and rounding this LP will not lead to any a priori approximation guarantees. However, solving the LP provides a useful bound on the objective and opens up the possibility of obtaining a posteriori approximation guarantees for using heuristic methods for modularity clustering. Agarwal and Kempe [2008] showed experimental results for solving this LP on problems involving up to 235 nodes. Later Aloise et al. [2010] developed an approach for exactly solving modularity that succeeded on graphs with up to 512 nodes.

Cluster Deletion The LP relaxation of cluster deletion can be obtained by starting with the relaxation for correlation clustering and fixing $x_{ij} = 1$ for $(i, j) \notin E$. In the previous chapter we showed a rounding technique that will produce a 2-approximation for the problem. In practice, rather than defining a variable x_{ij} for $(i, j) \notin E$ and constraining it to be one, it is better to eliminate these variables and update the constraint set. The LP relaxation can be expressed in a simplified form as follows:

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in E} x_{ij} & \\ \text{subject to} & x_{ij}\leq x_{ik}+x_{jk} & \text{if } (i,j,k)\in T \\ & 1\leq x_{ik}+x_{jk} & \text{if } (i,j,k)\in \tilde{T}_k \\ & 0\leq x_{ij}\leq 1 & \text{for all } (i,j)\in E. \end{array} \tag{5.5}$$

In the above, T represents the set of triangles, i.e. triplets of nodes i, j, k that form a clique in G. The set $\tilde{T}_k$ represents *bad* triangles “centered” at k, i.e. k shares an edge with i and j, but $(i, j) \notin E$.

Maximum Cut Given a graph $G = (V, E)$, the maximum cut problem seeks to partition G into two clusters in a way that maximizes the number of edges crossing between the clusters. If A_{ij} is the adjacency indicator for an edge between i and j, the linear programming relaxation for Max Cut is

$$\begin{array}{lll} \text{maximize} & \sum_{i,j} A_{ij} x_{ij} & \\ \text{subject to} & x_{ij} \leq x_{ik} + x_{jk} & \text{for all } i, j, k \\ & x_{ij} + x_{ik} + x_{jk} \leq 2 & \text{for all } i, j, k \\ & 0 \leq x_{ij} \leq 1 & \text{for all } i, j. \end{array} \tag{5.6}$$

Note that if the linear constraints $x_{ij} \in [0, 1]$ were replaced with binary constraints $x_{ij} \in \{0, 1\}$, then this would correspond to an integer linear program for the exact Max Cut objective. The constraints $x_{ij} + x_{ik} + x_{jk} \leq 2$ are included to ensure in the binary case that nodes are assigned to at most two different clusters. The integrality gap of this linear program is $2 - \varepsilon$ [Poljak and Tuza, 1994]. In the case of dense graphs, this can be improved to a $1 + \varepsilon$ integrality gap when additional constraints are added [de la Vega and Kenyon-Mathieu, 2007].

Metric Nearness The Metric Nearness Problem [Brickell et al., 2008] is another key example of metric constrained optimization. This problem seeks the nearest *metric* matrix $\mathbf{X}^* = (x^*_{ij})$ to a *dissimilarity* matrix $\mathbf{D} = (d_{ij})$. Here, a dissimilarity matrix is a nonnegative, symmetric, zero-diagonal matrix, and a *metric* matrix is a dissimilarity matrix whose entries satisfy metric constraints. If M_n represents the set of metric matrices of size $n \times n$, then the problem can be formalized as follows:

$$\mathbf{X}^* = \operatorname{argmin}_{\mathbf{X} \in M_n} \left(\sum_{i \neq j} w_{ij} \left| (x_{ij} - d_{ij}) \right|^p \right)^{1/p} \tag{5.7}$$

where $w_{ij} \geq 0$ is a weight indicating how strongly we wish $\mathbf{X}^*$ and $\boldsymbol{D}$ to coincide at entry ij. When $p = 1$, the problem can be cast as a linear program by introducing variables $\boldsymbol{M} = (m_{ij})$:

$$\begin{array}{lll} \text{minimize} & \sum_{i<j} w_{ij} m_{ij} & \\ \text{subject to} & x_{ij} \leq x_{ik} + x_{jk} & \text{for all } i, j, k \\ & x_{ij} - d_{ij} \leq m_{ij} & \text{for all } i, j \\ & d_{ij} - x_{ij} \leq m_{ij} & \text{for all } i, j \end{array} \tag{5.8}$$

where the last two constraints ensure that at optimality, $m_{ij} = |x_{ij} - d_{ij}|$.

Inspired by the metric nearness problem and the results of Brickell et al. [2008], Gilbert and Jain [2017] and Fan et al. [2018] later introduced the closely related problems of *sparse metric repair* and *metric violation distance.* These in turn led to further follow-up work on *metric repair* [Gilbert and Sonthalia, 2018a,b,c].

Our first theorem in this chapter shows that LP (5.2) and LP (5.8) are in fact equivalent.

Theorem 5.2.1 *Consider an instance of correlation clustering* $G = (V, W^+, W^-)$ *and set* $w_{ij} = |w_{ij}^+ - w_{ij}^-|$*. Define an* $n \times n$ *matrix* $\boldsymbol{D} = (d_{ij})$ *where* $d_{ij} = 1$ *if* $w_{ij}^- > w_{ij}^+$ *and* $d_{ij} = 0$ *otherwise. Then* $\boldsymbol{X} = (x_{ij})$ *is an optimal solution to the LP relaxation of* (2.12) *if and only if* $(\boldsymbol{X}, \boldsymbol{M})$ *is an optimal solution for* (5.8)*, where* $\boldsymbol{M} = (m_{ij}) = (|x_{ij} - d_{ij}|)$*.*

Proof We will assume that at most one of w_{ij}^+, w_{ij}^- is positive, so every pair of nodes is either labeled similar or dissimilar. If this were not the case, we could introduce new edge weights $(\tilde{w}_{ij}^+, \tilde{w}_{ij}^-)$ for each pair ij defined to be

$$\tilde{w}_{ij}^+ = \begin{cases} w_{ij}^+ - w_{ij}^- & \text{if } w_{ij}^+ > w_{ij}^- \\ 0 & \text{otherwise,} \end{cases} \qquad \tilde{w}_{ij}^- = \begin{cases} w_{ij}^- - w_{ij}^+ & \text{if } w_{ij}^- > w_{ij}^+ \\ 0 & \text{otherwise}. \end{cases} \tag{5.9}$$

This change of variables would alter the LP objective by only an additive constant, so the optimal LP solution would remain the same. In the remainder of the proof we will simply assume that at most one of w_{ij}^+, w_{ij}^- is positive.

We equivalently consider an unsigned graph $G' = (V, E)$ with the same node set V and an adjacency matrix $\boldsymbol{A} = (A_{ij})$ where $A_{ij} = 1$ if $w_{ij}^- = 0$ and $A_{ij} = 0$ otherwise. If $w_{ij} = \max\{w_{ij}^+, w_{ij}^-\}$, the correlation clustering LP can then be written

$$\begin{array}{lll} \text{minimize} & \sum_{i<j} w_{ij}\left(A_{ij}x_{ij} + (1 - A_{ij})(1 - x_{ij})\right) & \\ \text{subject to} & x_{ij} \le x_{ik} + x_{jk} & \text{for all } i, j, k \\ & 0 \le x_{ij} \le 1 & \text{for all } i, j. \end{array} \tag{5.10}$$

In order to see the equivalence between the correlation clustering LP relaxation and the metric nearness problem, we define a dissimilarity matrix $\mathbf{D} = (d_{ij})$ by setting $d_{ij} = 1 - A_{ij}$. Notice that because $d_{ij} \in \{0, 1\}$ and $x_{ij} \in [0, 1]$, the key factor in the objective can be simplified thus:

$$(1 - d_{ij})x_{ij} + d_{ij}(1 - x_{ij}) = |x_{ij} - d_{ij}|\,,$$

and the LP relaxation of correlation clustering shown in (5.10) is equivalent to

$$\begin{array}{lll} \text{minimize} & \sum_{i<j} w_{ij}|x_{ij} - d_{ij}| & \\ \text{subject to} & x_{ij} \le x_{ik} + x_{jk} & \text{for all } i, j, k \\ & 0 \le x_{ij} \le 1 & \text{for all } i, j. \end{array} \tag{5.11}$$

The only difference between this objective and ℓ_1 metric nearness is that we have included explicit bounds on the variables x_{ij}. To finish the proof we note that even without the constraint family "$0 \le x_{ij} \le 1$ for all pairs (i, j)", every optimal solution to problem (5.11) in fact satisfies those constraints. We can prove this fact by contradiction: assume that $\boldsymbol{X} = (x_{ij})$ is an optimal solution obtained by solving objective (5.11), *without* including constraints $x_{ij} \in [0, 1]$. Assume also that at least one variable is greater than one or less than zero. Next, define a new set of variables $\boldsymbol{Z} = (z_{ij})$ as follows:

$$z_{ij} = \begin{cases} 0 & \text{if } x_{ij} < 0 \\ x_{ij} & \text{if } x_{ij} \in [0,1] \\ 1 & \text{if } x_{ij} > 1. \end{cases}$$

Notice that this $\boldsymbol{Z}$ would have a strictly lower (i.e. better) objective score than $\boldsymbol{X}$ for problem (5.11), because $|z_{ij} - d_{ij}| \leq |x_{ij} - d_{ij}|$ for all (i,j), and this inequality is strict for at least one pair of nodes, since we assumed that there is at least one variable x_{ij} that is not in $[0,1]$. It just remains to show that $\boldsymbol{Z}$ also satisfies triangle inequality constraints and is thus feasible, which would lead to the desired contradiction to the optimality of $\boldsymbol{X}$.

Proving $\boldsymbol{Z}$ is feasible is tedious, as it requires checking a long list of cases. We consider a triplet (i,j,k), and we wish to check that

$$z_{ij} \leq z_{jk} + z_{ik}$$
$$z_{ik} \leq z_{jk} + z_{ij}$$
$$z_{jk} \leq z_{ij} + z_{ik}$$

for all possible values of (x_{ij}, x_{ik}, x_{jk}). To illustrate the technique we will provide proofs for the cases where the $\boldsymbol{X}$ variables are all nonnegative, but may be larger than 1. The same approach works if we also considered the case where some of the $\boldsymbol{X}$ are negative, though this involves checking 27 cases, which we do not list exhaustively here. We just focus on the eight subcases cases in which all variables are assumed nonnegative. Let v_{ab} be a binary variable indicating whether $x_{ab} > 1$.

- Case 1: $(v_{ij}, v_{ik}, v_{jk}) = (0,0,0)$. In this case the z variables are identical to the x variables and the constraints hold.

- Case 2: $(v_{ij}, v_{ik}, v_{jk}) = (1,1,1)$. In this case $z_{ij} = z_{ik} = z_{jk} = 1$ and the constraints hold.

- Case 3: $(v_{ij}, v_{ik}, v_{jk}) = (1,0,0)$. Since $x_{ij} > 1$, by construction $z_{ij} = 1$, and $z_{jk} = x_{jk} \leq 1$, $z_{ik} = x_{ik} \leq 1$, so we can confirm that:

$$z_{ij} < x_{ij} \leq x_{ik} + x_{jk} = z_{ik} + z_{jk}$$
$$z_{ik} = x_{ik} \leq 1 \leq x_{jk} + 1 = z_{jk} + z_{ij}$$
$$z_{jk} = x_{jk} \leq 1 \leq x_{ik} + 1 = z_{ik} + z_{ij}\,.$$

- Case 4: $(v_{ij}, v_{ik}, v_{jk}) = (0,1,0)$. This is symmetric to case 3.

- Case 5: $(v_{ij}, v_{ik}, v_{jk}) = (1, 1, 0)$. In this case we see that $z_{ij} = 1 < x_{ij}$, $z_{ik} = 1 < x_{ik}$, and $z_{jk} = x_{jk}$, so

$$\begin{aligned} z_{ij} &= 1 < 1 + z_{jk} = z_{ik} + z_{jk} \\ z_{ik} &= 1 < 1 + z_{jk} = z_{ij} + z_{jk} \\ z_{jk} &< 1 \leq 1 + 1 = z_{ij} + z_{ik}. \end{aligned}$$

- Case 6: $(v_{ij}, v_{ik}, v_{jk}) = (0, 0, 1)$. Symmetric to cases 3 and 4.
- Case 7: $(v_{ij}, v_{ik}, v_{jk}) = (1, 0, 1)$. Symmetric to case 5.
- Case 8: $(v_{ij}, v_{ik}, v_{jk}) = (0, 1, 1)$. Symmetric to cases 5 and 7.

■

This equivalence result is significant for our purposes given that Sra et al. [2005] developed an efficient technique for metric nearness than enabled them to solve problems with up to 5000 data points. In contrast, most results for solving metric LPs in practice only involve graphs with up to a few hundred nodes. Miyauchi et al. [2018] were able to solve correlation clustering exactly (i.e., solving the *integer* linear program, and not just the LP relaxation) on one problem with 2500 nodes, though this relied crucially on the problem being very sparse.

The success of Sra et al. [2005] is due to a carefully implemented projection method for solving metric nearness. While this enabled the authors to obtain results for problems with thousands of nodes, the output of their algorithm satisfies no rigorous guarantees with respect to optimality or constraint satisfaction. Nevertheless, this approach shows great promise for the development of memory efficient techniques for metric optimization. Inspired by their work, this chapter presents a rigorous optimization framework for efficiently solving metric LPs using projection methods. We begin with a broad overview of projection methods and their application to *quadratic* programming. After this, we prove several results for approximately solving metric constrained *linear* programs for certain graph clustering problems, by defining and solving a quadratic regularization of the original LP.

5.3 Background: Projection Methods and Quadratic Programming

Each of the graph clustering relaxations given in the previous section can be cast as a linear program in the following form:

$$\min_{\mathbf{x}} \; \mathbf{c}^T\mathbf{x} \quad \text{s.t. } \boldsymbol{A}\mathbf{x} \leq \mathbf{b}, \tag{5.12}$$

where $\boldsymbol{A}$ is a large and very sparse matrix with $O(n^3)$ rows and $O(n^2)$ columns. Standard optimization software will be unable to solve these LPs for large values of n, due to memory constraints, so we instead turn our attention to applying a simple projection method for solving a closely related quadratic program:

$$\min_{\mathbf{x}} \; Q(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \frac{1}{2\gamma}\mathbf{x}^T\boldsymbol{W}\mathbf{x} \quad \text{s.t. } \boldsymbol{A}\mathbf{x} \leq \mathbf{b}, \tag{5.13}$$

In (5.13), above, $\boldsymbol{W}$ is a diagonal matrix with positive diagonal entries and $\gamma > 0$. When $\boldsymbol{W}$ is the identity matrix, Mangasarian [1984] showed that there exists some $\gamma_0 > 0$ such that for all $\gamma > \gamma_0$, the optimal solution to the quadratic program corresponds to the minimum 2-norm solution of the original LP. In later sections we will consider how to set γ in order to obtain good approximation for graph clustering problems such as correlation clustering, cluster deletion, and sparsest cut. In this section we providing a broad overview of Dykstra's projection method for quadratic programming, which will set the stage for developing efficient projection methods for solving graph clustering relaxations.

5.3.1 Strictly Convex Quadratic Programming

For positive integers M and N, let $\boldsymbol{A} \in \mathbb{R}^{N\times M}$, $\mathbf{b} \in \mathbb{R}^M$, $\mathbf{c} \in \mathbb{R}^N$, and let $\boldsymbol{W} \in \mathbb{R}^{N\times N}$ be a symmetric positive definite matrix. Consider the following quadratic program:

$$\begin{aligned} \min_{\mathbf{x}} \; & Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\boldsymbol{W}\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{s.t. } & \boldsymbol{A}\mathbf{x} \leq \mathbf{b}. \end{aligned} \tag{5.14}$$

Since $\boldsymbol{W}$ is positive definite, this objective function is strictly convex and has a unique solution. The dual of this quadratic program is

$$\begin{aligned} \max_{\mathbf{y}} \; D(\mathbf{y}) = -\mathbf{b}^T\mathbf{y} - \frac{1}{2}\left\|\mathbf{A}^T\mathbf{y} + \mathbf{c}\right\|^2_{\boldsymbol{W}^{-1}} \\ \text{s.t. } \mathbf{y} \geq 0. \end{aligned} \tag{5.15}$$

The set of primal and dual variables $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ are optimal for (5.13) and (5.15) when the Karush-Kuhn-Tucker (KKT) conditions are satisfied:

$$\begin{aligned} &\boldsymbol{W}\hat{\mathbf{x}} = -\mathbf{A}^T\hat{\mathbf{y}} - \mathbf{c} \\ &\boldsymbol{A}\hat{\mathbf{x}} \leq \mathbf{b} \\ &\hat{\mathbf{y}}^T(\boldsymbol{A}\hat{\mathbf{x}} - \mathbf{b}) = 0 \\ &\hat{\mathbf{y}} \geq 0. \end{aligned}$$

In this case, $\hat{\mathbf{y}} = \operatorname{argmin} D(\mathbf{y})$ and $\hat{\mathbf{x}} = \operatorname{argmin} Q(\mathbf{x})$ and $D(\hat{\mathbf{y}}) = Q(\hat{\mathbf{x}})$. In general, if $\mathbf{x}$ satisfies $\boldsymbol{A}\mathbf{x} \leq \mathbf{b}$ and $\mathbf{y} \geq 0$, then $D(\mathbf{y}) \leq D(\hat{\mathbf{y}}) = Q(\hat{\mathbf{x}}) \leq Q(\mathbf{x})$.

Algorithm 7 Dykstra's Method for BAP

```
**Input:** Convex sets $C_i \subset \mathbb{R}^N$, vector $\mathbf{z} \in \mathbb{R}^N$
**Output:** $\mathbf{x}^* = P_C(\mathbf{z}) = \arg\min_{\mathbf{x}\in C} ||\mathbf{x} - \mathbf{z}||^2$ where $C = \cap_{i=1}^{M} C_i$
(Initialize *increment* vectors) $I_i = 0 \in \mathbb{R}^N$ for $i = 1, 2, \ldots, M$
(Initialize iterate) $\mathbf{x} := \mathbf{z}$
for $k = 1, 2, \ldots$ do
    (Visit constraints cyclically): $i := (k-1) \bmod M + 1$
    (Perform correction step): $\mathbf{x}_c := \mathbf{x} - I_i$
    (Perform projection step): $\mathbf{x} := P_{C_i}(\mathbf{x}_c)$
    (Update increment vector): $I_i := \mathbf{x} - \mathbf{x}_c$
```

5.3.2 The Best Approximation Problem

Let $C_i \subset \mathbb{R}^N$ for $i = 1, 2, \ldots, M$ be convex sets and $C = \cap_{i=1}^{M} C_i$ be their intersection (which is also convex). Given $\mathbf{z} \in \mathbb{R}^N$, the best approximation problem (BAP) is to find

$$\mathbf{x}^* = P_C(\mathbf{z}) = \arg\min_{\mathbf{x}\in C} ||\mathbf{x} - \mathbf{z}||^2 \tag{5.16}$$

for some norm $|| \cdot ||$. $P_C(\mathbf{z})$ is the *projection of* $\mathbf{z}$ *onto* C. The BAP is often solved using *projection methods*, which operate by visiting the constraints ($\mathbf{x} \in C_i$) cyclically and repeatedly performing easier projections of the form

$$\mathbf{x}_i = P_{C_i}(\mathbf{z}_i) = \arg\min_{\mathbf{x}\in C_i} ||\mathbf{x} - \mathbf{z}_i||^2.$$

5.3.3 Dykstra's Method for BAP

Dykstra's method [Dykstra, 1983] is one common approach to solving the BAP. Pseudocode for this method is given in Algorithm 7. This general approach simplifies when we apply it specifically to a quadratic program like (5.14). To do this, we will not use the standard norm and standard inner product, but, given arbitrary vectors $\mathbf{f}, \mathbf{g} \in \mathbb{R}^N$, we will apply a weighted norm:

$$\langle \mathbf{f}, \mathbf{g} \rangle_w = \mathbf{f}^T \boldsymbol{W} \mathbf{g}$$

$$||\mathbf{f}||_w^2 = \langle \mathbf{f}, \mathbf{f} \rangle_w = \mathbf{f}^T \boldsymbol{W} \mathbf{f},$$

where $\boldsymbol{W}$ is assumed to be positive definite.

Observe the the following equivalence:

$$\begin{aligned}\frac{1}{2}\left\|\mathbf{x}+\boldsymbol{W}^{-1}\mathbf{c}\right\|_w^2 &= \frac{1}{2}\mathbf{x}^T\boldsymbol{W}\mathbf{x}+\mathbf{x}^T\boldsymbol{W}\boldsymbol{W}^{-1}\mathbf{c}+\frac{1}{2}\mathbf{c}^T\boldsymbol{W}^{-1}\mathbf{c}\\ &= \frac{1}{2}\mathbf{x}^T\boldsymbol{W}\mathbf{x}+\mathbf{c}^T\mathbf{x}+\textit{constant.}\end{aligned}$$

In other words, the quadratic program (5.14) is equivalent to the following best approximation problem:

$$\mathbf{x}^* = P_C(\mathbf{z}) = \arg\min_{\mathbf{x}\in C} ||\mathbf{x}-\mathbf{z}||_w^2, \text{ where } C = \cap_{i=1}^M C_i\,. \tag{5.17}$$

Here, $\mathbf{z} = -\boldsymbol{W}^{-1}\mathbf{c}$ and we are projecting onto half-space constraints:

$$C_i = \{\mathbf{x}\in\mathbb{R}^N : \mathbf{a}_i^T\mathbf{x}\le b_i\} = \{\mathbf{x}\in\mathbb{R}^N : \langle\tilde{\mathbf{a}}_i,\mathbf{x}\rangle_w \le b_i\}\,, \tag{5.18}$$

where $\mathbf{a}_i^T$ is the ith row of constraint matrix $\boldsymbol{A}$ and $\tilde{\mathbf{a}}_i = \boldsymbol{W}^{-1}\mathbf{a}_i$ is scaled so that

$$\langle\tilde{\mathbf{a}}_i,\mathbf{x}\rangle_w = \tilde{\mathbf{a}_i}^T\boldsymbol{W}\mathbf{x} = \mathbf{a}_i^T\boldsymbol{W}^{-1}\boldsymbol{W}\mathbf{x} = \mathbf{a}_i^T\mathbf{x}.$$

Dykstra's method relies on being able to project quickly onto each constraint C_i. For the half-space constraints in (5.18), the projection can be computed as follows:

$$P_{C_i}(\mathbf{x}) = \arg\min_{\mathbf{x}'\in C_i} ||\mathbf{x}'-\mathbf{x}||^2 = \mathbf{x} - \frac{[\langle\tilde{\mathbf{a}}_i,\mathbf{x}\rangle - b_i]^+}{\|\tilde{\mathbf{a}}_i\|^2}\tilde{\mathbf{a}}_i$$

where $[a]^+$ is a if $a > 0$, and is zero otherwise (a standard textbook result, see section 4.1.3 of Cegielski [2012]). Since we are using the $\boldsymbol{W}$-weighted norm and inner product, for our problem this becomes:

$$P_{C_i}(\mathbf{x}) = \mathbf{x} - \frac{[\mathbf{a}_i^T\mathbf{x}-b_i]^+}{\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i}(\boldsymbol{W}^{-1}\mathbf{a}_i).$$

Observe that this type of projection always takes the original vector $\mathbf{x}$ and just adds a constant times a vector $\boldsymbol{W}^{-1}\mathbf{a}_i$ when visiting constraint i. Therefore, when implementing the algorithm, we do not need to store an entire increment vector I_i for each constraint. As long as we have the weight matrix $\boldsymbol{W}$ and the constraint matrix $\mathbf{A}$ stored up front, it will suffice to store a single extra constant $[\mathbf{a}_i^T\mathbf{x}-b_i]^+/\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i$ in order to perform the correction step.

We re-write Dykstra's algorithm for quadratic programming in Algorithm 8.

5.3.4 Hildreth's Projection Method

It is well known that for half-space constraints, Dykstra's method is equivalent to Hildreth's method [Hildreth, 1957]. Pseudocode for Hildreth's method applied to the same strictly convex quadratic program (5.13) is shown in Algorithm 9. At first glance there appear to be slight differences, but these are easily accounted for. For completeness we include a full proof.

Algorithm 8 Dykstra's Method for QP (5.13)

```
Input: $\boldsymbol{A} \in \mathbb{R}^{N\times M}, \mathbf{b} \in \mathbb{R}^M, \mathbf{c} \in \mathbb{R}^N, \boldsymbol{W} \in \mathbb{R}^{N\times N}$(positive definite)
Output: $\hat{\mathbf{x}} = \operatorname{argmin}_{\mathbf{x}\in\mathcal{A}} Q(\mathbf{x})$ where $\mathcal{A} = \{\mathbf{x} \in \mathbb{R}^N : \boldsymbol{A}\mathbf{x} \le \mathbf{b}\}$
$\mathbf{y} := \mathbf{0} \in \mathbb{R}^M$, $\mathbf{x} := -\boldsymbol{W}^{-1}\mathbf{c}$, $k := 0$
while not converged do
    $k := k + 1$
    (Visit constraints cyclically): $i := (k-1) \bmod M + 1$
    (Perform correction step): $\mathbf{x} := \mathbf{x} + y_i(\boldsymbol{W}^{-1}\mathbf{a}_i)$ where $\mathbf{a}_i$ is the $i$th row of $\boldsymbol{A}$
    (Perform projection step): $\mathbf{x} := \mathbf{x} - \theta_i^+(\boldsymbol{W}^{-1}\mathbf{a}_i)$ where $\theta_i^+ = \frac{\max\{\mathbf{a}_i^T\mathbf{x}-b_i,0\}}{\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i}$
    (Update dual variables): $y_i := \theta_i^+ \ge 0$
```

Algorithm 9 Hildreth's Method for QP (5.13)

```
Input: $\boldsymbol{A} \in \mathbb{R}^{N\times M}, \mathbf{b} \in \mathbb{R}^M, \mathbf{c} \in \mathbb{R}^N, \boldsymbol{W} \in \mathbb{R}^{N\times N}$(positive definite)
Output: $\hat{\mathbf{x}} = \operatorname{argmin}_{\mathbf{x}\in\mathcal{A}} Q(\mathbf{x})$ where $\mathcal{A} = \{\mathbf{x} \in \mathbb{R}^N : \boldsymbol{A}\mathbf{x} \le \mathbf{b}\}$
$\mathbf{y} := \mathbf{0} \in \mathbb{R}^M$, $\mathbf{x} := -\boldsymbol{W}^{-1}\mathbf{c}$, $k := 0$
while not converged do
    $k := k + 1$
    $i := (k-1) \bmod M + 1$
    $\theta_i := \frac{\mathbf{a}_i^T\mathbf{x}-b_i}{\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i}$
    $\delta := \min\{-\theta_i, y_i\}$
    $\mathbf{x} := \mathbf{x} + \delta\boldsymbol{W}^{-1}\mathbf{a}_i$
    $y_i := y_i - \delta$.
```

Theorem 5.3.1 *The iterates computed by Algorithm 8 and Algorithm 9 are identical. Furthermore,* $y_i \ge 0$ *is maintained throughout the algorithm.*

Proof The equivalence can be shown by combining the correction and projection steps in Dykstra's method:

$$\begin{aligned}
&\text{(Dykstra's correction step) } \mathbf{x}_c := \mathbf{x} + y_i\boldsymbol{W}^{-1}\mathbf{a}_i \\
&\text{(Dykstra's projection step) } \mathbf{x}' := \mathbf{x}_c - \frac{[\mathbf{a}_i^T\mathbf{x}_c - b_i]^+}{\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i}\boldsymbol{W}^{-1}\mathbf{a}_i.
\end{aligned}$$

Combining both of these steps in one line amounts to the following update

$$\mathbf{x}' = \mathbf{x} + y_i\boldsymbol{W}^{-1}\mathbf{a}_i - \frac{[\mathbf{a}_i^T(\mathbf{x} + y_i\boldsymbol{W}^{-1}\mathbf{a}_i) - b_i]^+}{\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i}\boldsymbol{W}^{-1}\mathbf{a}_i.$$

The outcome of this update depends on the value of

$$\mathbf{a}_i^T(\mathbf{x} + y_i\boldsymbol{W}^{-1}\mathbf{a}_i) - b_i = (\mathbf{a}_i^T\mathbf{x} + y_i\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i) - b_i$$

which is greater than or equal to zero if and only if:

$$-\theta_i = -\frac{\mathbf{a}_i^T\mathbf{x} - b_i}{\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i} \leq y_i.$$

Case 1: $-\theta_i \leq y_i$. In this case, for Hildreth's method $\delta = -\theta_i$ and the update is:

$$\begin{aligned}\mathbf{x}' &= \mathbf{x} + y_i\boldsymbol{W}^{-1}\mathbf{a}_i - \frac{\mathbf{a}_i^T\mathbf{x} - b_i + y_i\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i}{\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i}\boldsymbol{W}^{-1}\mathbf{a}_i \\ &= \mathbf{x} + y_i\boldsymbol{W}^{-1}\mathbf{a}_i - (\theta_i + y_i)\boldsymbol{W}^{-1}\mathbf{a}_i \\ &= \mathbf{x} - \theta_i\boldsymbol{W}^{-1}\mathbf{a}_i \\ &= \mathbf{x} + \delta\boldsymbol{W}^{-1}\mathbf{a}_i.\end{aligned}$$

Case 2: $y_i < -\theta_i$. In this case, $[\mathbf{a}_i^T(\mathbf{x} + y_i\boldsymbol{W}^{-1}\mathbf{a}_i) - b_i]^+ = 0$ and the update is $\mathbf{x}' = \mathbf{x} + y_i\boldsymbol{W}^{-1}\mathbf{a}_i$.

In both cases, we see that the updates performed by Dykstra's method are identical to the steps in Hildreth's method. Note that $y_i \geq 0$ is maintained in either case: if $\delta = y_i$ then we update $y_i := y_i - \delta = 0$, and if $\delta = -\theta_i$ that means $-\theta_i \leq y_i \implies 0 \leq y_i + \theta_i = y_i - \delta$. Note that these variables y_i correspond to nonnegative dual variables for the dual quadratic program shown in (5.15).

■

Let $\mathbf{x}_k$ and $\mathbf{y}_k$ represent the kth iterates for primal and dual variables in Algorithm 8. We showed in the above theorem that the constraint $\mathbf{y}_k \geq 0$, which is one of the four KKT conditions for optimality, is satisfied at all iterations. Next we show that there is one more KKT condition that is also satisfied at every step of the algorithm.

Theorem 5.3.2 *Vectors* $(\mathbf{x}_k, \mathbf{y}_k)$ *satisfy* $\boldsymbol{W}\mathbf{x}_k = -\boldsymbol{A}^T\mathbf{y}_k - \mathbf{c}$ *at every step of Algorithm 8 (and the equivalent Algorithm 9).*

Proof When we visit the ith constraint in Algorithm 9, we perform the following steps:

1. Compute $\theta_i = \frac{\mathbf{a}_i^T\mathbf{x} - b_i}{\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i}$
2. Set $\delta = \min\{-\theta_i, y_i\}$
3. Update $\mathbf{x} := \mathbf{x} + \delta\boldsymbol{W}^{-1}\mathbf{a}_i$, and $\mathbf{y} := \mathbf{y} - \mathbf{e}_i\delta$,

where $\mathbf{e}_i$ is the vector with zeros everywhere except for a 1 in the ith position. We will prove the result by induction. The primal and dual variables are initialized to $\mathbf{y}_0 = 0$ and $\mathbf{x}_0 = -\boldsymbol{W}^{-1}\mathbf{c}$, so the base case holds since $\boldsymbol{W}\mathbf{x}_0 = \boldsymbol{W}(-\boldsymbol{W}^{-1}\mathbf{c}) = 0 - \mathbf{c} = -\mathbf{A}^T\mathbf{y}_0 - \mathbf{c}$.

For the induction step, assume that $\boldsymbol{W}\mathbf{x}_k = -\boldsymbol{A}^T\mathbf{y}_k - \mathbf{c}$ holds for some k and perform a single update to get new primal and dual vectors $\mathbf{x}_{k+1}$ and $\mathbf{y}_{k+1}$:

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k + \delta\boldsymbol{W}^{-1}\mathbf{a}_i\\ \mathbf{y}_{k+1} &= \mathbf{y}_k - \mathbf{e}_i\delta.\end{aligned}$$

Then note:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta\boldsymbol{W}^{-1}\mathbf{a}_i = \boldsymbol{W}^{-1}(-\boldsymbol{A}^T\mathbf{y}_k - \mathbf{c}) + \delta\boldsymbol{W}^{-1}\mathbf{a}_i = \boldsymbol{W}^{-1}(-\boldsymbol{A}^T\mathbf{y}_k + \delta\mathbf{a}_i - \mathbf{c})$$

and

$$-\boldsymbol{A}^T\mathbf{y}_{k+1} - \mathbf{c} = -\boldsymbol{A}^T(\mathbf{y}_k - \mathbf{e}_i\delta) - \mathbf{c} = -\boldsymbol{A}^T\mathbf{y}_k - \mathbf{c} + \delta\mathbf{a}_i$$

so these combine to yield $\boldsymbol{W}\mathbf{x}_{k+1} = -\boldsymbol{A}^T\mathbf{y}_{k+1} - \mathbf{c}$. ■

5.3.5 Equivalence with Coordinate Descent

Hildreth's method is also equivalent to performing a coordinate descent procedure on the negative of the dual objective function. The details shown here are a slight generalization of the result shown by Dax [2001, 2003], updated to explicitly include a non-identity weight matrix $\boldsymbol{W}$.

We first replace the maximization objective (5.15) with an equivalent quadratic program that is minimized:

$$\begin{aligned}\min_{\mathbf{y}}\ F(\mathbf{y}) = \mathbf{b}^T\mathbf{y} + \frac{1}{2}\left\|\mathbf{A}^T\mathbf{y} + \mathbf{c}\right\|^2_{\boldsymbol{W}^{-1}}\\ \text{s.t. } \mathbf{y} \geq 0.\end{aligned} \tag{5.19}$$

The connection to coordinate descent is seen by realizing that the value θ_i computed in Algorithm 9 uniquely minimizes the following one-variable function:

$$\begin{aligned}f(\theta) &= F(\mathbf{y} + \mathbf{e}_i\theta)\\ &= \mathbf{b}^T\mathbf{y} + \theta b_i + \frac{1}{2}\left\|\boldsymbol{A}^T(\mathbf{y} + \mathbf{e}_i\theta) + \mathbf{c}\right\|^2_{\boldsymbol{W}^{-1}}\\ &= \mathbf{b}^T\mathbf{y} + \theta b_i + \frac{1}{2}\left\|-\boldsymbol{W}\mathbf{x} + \theta\mathbf{a}_i\right\|^2_{\boldsymbol{W}^{-1}}\\ &= \mathbf{b}^T\mathbf{y} + \theta b_i + \frac{1}{2}\left((-\boldsymbol{W}\mathbf{x} + \theta\mathbf{a}_i)^T\boldsymbol{W}^{-1}(-\boldsymbol{W}\mathbf{x} + \theta\mathbf{a}_i)\right)\\ &= \mathbf{b}^T\mathbf{y} + \theta b_i + \frac{1}{2}\left(\mathbf{x}^T\boldsymbol{W}\mathbf{x} + \theta^2\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i - 2\theta\mathbf{a}_i^T\mathbf{x}\right)\\ &= \theta b_i + \frac{1}{2}\theta^2\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i - \theta\mathbf{a}_i^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\boldsymbol{W}\mathbf{x} + \mathbf{b}^T\mathbf{y}.\end{aligned}$$

This is minimized when

$$f'(\theta) = b_i - \mathbf{a}_i^T\mathbf{x} + \theta\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i = 0 \implies \theta = \frac{\mathbf{a}_i^T\mathbf{x} - b_i}{\mathbf{a}_i^T\boldsymbol{W}^{-1}\mathbf{a}_i}.$$

During one step of Hildreth's method, if the algorithm can perform this update without violating constraint $y_i \geq 0$, then it does so, i.e. $\delta = \theta_i$ and $y_i = y_i + \theta \geq 0$. Otherwise if $\theta < -y_i \leq 0$, the algorithm decreases y_i as much as possible without violating the constraint (i.e. it sets $y_i = 0$). The function strictly decreases, since

$$f(0) - f(\delta) \geq \frac{1}{2}\delta^2 \mathbf{a}_i^T \boldsymbol{W}^{-1}\mathbf{a}_i > 0.$$

To see this, realize that $\delta = \nu\theta_i$ for some $\nu \in [0,1]$ and $\theta_i(\mathbf{a}_i^T \boldsymbol{W}^{-1}\mathbf{a}_i) = (\mathbf{a}_i^T\mathbf{x} - b_i)$, so

$$\begin{aligned} f(0) - f(\delta) &= \delta(\mathbf{a}_i^T\mathbf{x} - b_i) - \delta^2/2\mathbf{a}_i^T \boldsymbol{W}^{-1}\mathbf{a}_i \\ &= \nu\theta_i(\theta_i \mathbf{a}_i^T \boldsymbol{W}^{-1}\mathbf{a}_i) - \frac{1}{2}\nu^2\theta_i^2 \mathbf{a}_i^T \boldsymbol{W}^{-1}\mathbf{a}_i \\ &= \frac{1}{2}\theta_i^2\nu^2 \mathbf{a}_i^T \boldsymbol{W}^{-1}\mathbf{a}_i(2-\nu)/\nu \\ &\geq \frac{1}{2}\delta^2 \mathbf{a}_i^T \boldsymbol{W}^{-1}\mathbf{a}_i \qquad \text{(since } \nu \in [0,1]\text{)}. \end{aligned}$$

5.4 Dykstra's Method for Metric Constrained Optimization

We now address in detail how to apply Dykstra's method in order to obtain a memory-efficient optimization framework for solving metric constrained linear programs. Recall that the metric LPs we wish to solve take on the form

$$\min_{\mathbf{x}} \ \mathbf{c}^T\mathbf{x} \quad \text{s.t. } \boldsymbol{A}\mathbf{x} \leq \mathbf{b}, \tag{5.20}$$

where $\boldsymbol{A}$ has $O(n^3)$ rows, $O(n^2)$ columns, and is very sparse and well-structured. We do not solve this LP directly, but instead introduce a positive definite (diagonal) matrix $\boldsymbol{W}$ and a parameter $\gamma > 0$, and focus on a related quadratic program, a *regularization* of the original LP:

$$\min_{\mathbf{x}} \ Q(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \frac{1}{2\gamma}\mathbf{x}^T\boldsymbol{W}\mathbf{x} \quad \text{s.t. } \boldsymbol{A}\mathbf{x} \leq \mathbf{b}. \tag{5.21}$$

This presentation differs slightly from the previous background section in that we have made the role of the parameter γ explicit. The dual of (5.21) is another quadratic program:

$$\max_{\mathbf{y}} \ D(\mathbf{y}) = -\mathbf{b}^T\mathbf{y} - \frac{\gamma}{2}(\mathbf{A}^T\mathbf{y} + \mathbf{c})^T\boldsymbol{W}^{-1}(\mathbf{A}^T\mathbf{y} + \mathbf{c}) \quad \text{s.t. } \mathbf{y} \geq 0. \tag{5.22}$$

Algorithm 10 gives pseudocode for applying Dykstra's method specifically to solve (5.21). Our full algorithmic approach takes Dykstra's method (Algorithm 10) and includes a number of key features that allow us to efficiently obtain high-quality solutions to metric constrained problems in practice. The first feature, a procedure for locally performing projections at metric constraints, is the key insight which led Sra et al. [2005] to develop efficient algorithms for metric nearness. In addition, we detail a sparse storage scheme for dual vectors, and include a more robust convergence check

Algorithm 10 Dykstra's Method for QP (5.21)

```
**Input:** $\boldsymbol{A} \in \mathbb{R}^{N\times M}, \mathbf{b} \in \mathbb{R}^{M}, \mathbf{c} \in \mathbb{R}^{N}, \gamma > 0, \boldsymbol{W} \in \mathbb{R}^{N\times N}$(diagonal, positive definite)
**Output:** $\hat{\mathbf{x}} = \operatorname{argmin}_{\mathbf{x}\in\mathcal{A}} Q(\mathbf{x})$ where $\mathcal{A} = \{\mathbf{x} \in \mathbb{R}^N : \boldsymbol{A}\mathbf{x} \leq \mathbf{b}\}$
$\mathbf{y} := \mathbf{0} \in \mathbb{R}^M$
$\mathbf{x} := -\gamma \boldsymbol{W}^{-1}\mathbf{c}$, $k := 0$
**while** *not converged* **do**
    $k := k + 1$
    (Visit constraints cyclically): $i := (k-1) \bmod M + 1$
    (Perform correction step): $\mathbf{x} := \mathbf{x} + y_i(\gamma \boldsymbol{W}^{-1}\mathbf{a}_i)$ where $\mathbf{a}_i$ is the $i$th row of $\boldsymbol{A}$
    (Perform projection step): $\mathbf{x} := \mathbf{x} - \theta_i^+(\gamma \boldsymbol{W}^{-1}\mathbf{a}_i)$ where $\theta_i^+ = \frac{\max\{\mathbf{a}_i^T\mathbf{x} - b_i, 0\}}{\gamma \mathbf{a}_i^T \boldsymbol{W}^{-1}\mathbf{a}_i}$
    (Update dual variables): $y_i := \theta_i^+ \geq 0$
```

that leads to better constraint satisfaction and stronger optimality guarantees for a variety of metric constrained problems.

In order to demonstrate our application of Dykstra's method to metric constrained linear programs, we will specifically consider the quadratic program related to the Leighton-Rao sparsest cut relaxation:

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in E} x_{ij} + \frac{1}{2\gamma}\sum_{i<j} w_{ij}x_{ij}^2 & \\ \text{subject to} & \sum_{i<j} x_{ij} = n & \\ & x_{ij} \leq x_{ik} + x_{jk} & \text{for all } i,j,k \\ & x_{ij} \geq 0 & \text{for all } i,j \end{array} \tag{5.23}$$

where $w_{ij} = 1$ if $(i,j) \in E$ and $w_{ij} = \lambda$ for some $\lambda \in (0,1)$ otherwise. We give justification for this choice of weights matrix in Section 5.5. We incorporate the parameter γ directly into a new weight matrix $\boldsymbol{W}_\gamma = \boldsymbol{W}/\gamma$. Initially the vector of dual variables $\mathbf{y}$ is set to zero, and $\mathbf{x} = -\boldsymbol{W}_\gamma^{-1}\mathbf{c}$. For the sparsest cut relaxation in particular, this means we set $\mathbf{x} = (x_{ij})$ as follows:

$$x_{ij} = \begin{cases} -\gamma & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

5.4.1 Efficient local updates

Projections of the form $\mathbf{x} := \mathbf{x} + \alpha \boldsymbol{W}^{-1}\mathbf{a}_i$ for $\boldsymbol{W}$ diagonal and a constant α will change $\mathbf{x}$ by at most the number of nonzero entries of $\mathbf{a}_i$, the ith row of constraint matrix $\boldsymbol{A}$. In the case of triangle inequality constraints, which dominate our constraint set, there are exactly three non-zero entries per constraint, so we perform each projection in a constant number of operations.

Consider the triangle inequality constraint $x_{ij} - x_{ik} - x_{jk} \leq 0$ associated with an ordered triplet (i,j,k). Let $t = t_{ijk}$ represent a unique ID corresponding to this constraint. For now we ignore the correction step of Dykstra's method, which is skipped over in the first round since the vector of dual variables is initialized to zero (i.e. $y_t = 0$). The projection step we must perform is

$$\mathbf{x} \leftarrow \mathbf{x} - \frac{[\mathbf{a}_t^T \mathbf{x} - b_t]^+}{\mathbf{a}_t^T \boldsymbol{W}_\gamma^{-1} \mathbf{a}_t} \boldsymbol{W}_\gamma^{-1} \mathbf{a}_t$$

where $\mathbf{a}_t$ contains exactly three entries: 1, -1, and -1, at the locations corresponding to variables x_{ij}, x_{ik}, and x_{jk}. This projection will only change $\mathbf{x}$ if constraint t is violated, so we first check if

$$\Delta = \mathbf{a}_t^T \mathbf{x} - b_t = x_{ij} - x_{ik} - x_{jk} > 0.$$

If so, we compute

$$\theta_t^+ = \frac{[\mathbf{a}_t^T \mathbf{x} - b_t]^+}{\mathbf{a}_t^T \boldsymbol{W}_\gamma^{-1} \mathbf{a}_t} = \frac{\Delta}{1/w_{ij} + 1/w_{ik} + 1/w_{jk}} = \frac{\Delta w_{ij} w_{ik} w_{jk}}{w_{ij} w_{ik} + w_{ij} w_{jk} + w_{ik} w_{ij}}.$$

The projection step then updates exactly three entries of $\mathbf{x}$:

$$x_{ij} \leftarrow x_{ij} - \theta_t^+ \frac{x_{ij}}{w_{ij}}, \qquad x_{ik} \leftarrow x_{ik} + \theta_t^+ \frac{x_{ik}}{w_{ik}}, \qquad x_{jk} \leftarrow x_{jk} + \theta_t^+ \frac{x_{jk}}{w_{jk}}.$$

All of these steps can be performed in a constant number of operations for each triangle inequality constraint.

5.4.2 Sparse storage of dual variables

For constraint sets that include triangle inequalities for every triplet of nodes (i,j,k), the dual vector $\mathbf{y}$ will be of length $O(n^3)$. Observe that the correction step in Algorithm 10 at constraint t will be nontrivial if and only if there was a nontrivial projection last time the constraint was visited. In other words, θ_t^+ was nonzero in the previous round and therefore $y_t > 0$.

Sparsity in the metric constraint dual variables Note that each triplet (i,j,k) corresponds to three different metric constraints: $x_{ij} - x_{ik} - x_{jk} \leq 0$, $x_{jk} - x_{ik} - x_{ij} \leq 0$, and $x_{ik} - x_{ij} - x_{jk} \leq 0$, and in each round at most one of these constraints will be violated, indicating that at least two dual variables will be zero. Dhillon et al. [2003] concluded that $\binom{n}{3}$ floating point numbers must be stored in implementing Dykstra's algorithm for the metric nearness problem. We further observe, especially for the correlation clustering LP, that often in practice for a large percentage of triplets (i,j,k), none of the three metric constraints is violated. Thus we can typically avoid the worst case $O(n^3)$ memory requirement by storing $\mathbf{y}$ sparsely.

Algorithm 11 MetricProjection(i, j, k)

```
$t :=$ unique ID for $(i, j, k)$ (constraint $x_{ij} - x_{ik} - x_{jk}$)
Obtain $(x_{ij}, x_{ik}, x_{jk})$ and weights $(w_{ij}, w_{ik}, w_{jk})$ from $\boldsymbol{X}$ and $\boldsymbol{W}_\gamma$
if $y_t > 0$ then
    $x_{ij} \leftarrow x_{ij} + y_t \frac{x_{ij}}{w_{ij}},\ x_{ik} \leftarrow x_{ik} - y_t \frac{x_{ik}}{w_{ik}},\ x_{jk} \leftarrow x_{jk} - y_t \frac{x_{jk}}{w_{jk}}.$
$\delta := x_{ij} - x_{ik} - x_{jk}$
if $\delta > 0$ then
    $\theta_t = \frac{\delta w_{ij} w_{ik} w_{jk}}{w_{ij} w_{ik} + w_{ij} w_{jk} + w_{ik} w_{ij}}$
    $x_{ij} \leftarrow x_{ij} - \theta_t \frac{x_{ij}}{w_{ij}},\ x_{ik} \leftarrow x_{ik} + \theta_t \frac{x_{ik}}{w_{ik}},\ x_{jk} \leftarrow x_{jk} + \theta_t \frac{x_{jk}}{w_{jk}}.$
    Store $y_t = \theta_t$
```

Storing y in dictionaries or arrays. Conceptually the easiest approach to storing nonzero entries in $\mathbf{y}$ is to maintain a dictionary of key-value pairs (t, y_t). In this case, when visiting constraint t, we check if the dictionary contains a nonzero dual variable $y_t > 0$ for this constraint, and if so we perform the corresponding non-trivial correction step. However, as long as the constraints are always visited in the same order, it is faster in practice to store two arrays with pairs (t, y_t) rather than a dictionary. The first array stores entries y_t that were set to a nonzero value in the previous pass through the constraints. We maintain a pointer to the entry in the array which gives us the next such constraint t that will require a nonzero correction in the current pass through the constraint set. The second array allocates space for the new dual variables that become nonzero after a projection step in the current pass through the constraints. These will be needed for corrections in the next round. Dykstra's method does not require we remember history beyond the last pass through constraints, so we never need more than two arrays storing pairs (t, y_t).

Pseudocode Algorithm 11 displays pseudocode for one step of our implementation of Dykstra's method when visiting a metric constraint. We assume the nonzero dual variables y_t are stored sparsely and can be efficiently queried and updated. The same basic outline applies also to non-metric constraints.

5.4.3 Robust Stopping Criteria

Many implementations of Dykstra's method stop when the change in vector $\mathbf{x}$ drops below a certain tolerance after one or more passes through the entire constraint set. This is the approach applied by Sra et al. [2005] for the metric nearness problem. However, Birgin and Raydan [2005] noted that in some cases this may occur even when the iterates are far from convergence. Because we are applying Dykstra's method specifically to quadratic programming, we can obtain a much more robust stopping criterion by carefully monitoring the dual objective function and dual variables, in a manner similar to the approach of Dax [2003].

Optimality Conditions Let $(\mathbf{x}_k, \mathbf{y}_k)$ denote the pair of primal and dual vectors computed by Dykstra's method after k iterations. We know that these vectors will converge to an optimal pair $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ such that $D(\hat{\mathbf{y}}) = \hat{Q} = Q(\hat{\mathbf{x}})$ where $\hat{Q}$ is the optimal objective for both the primal (5.21) and dual (5.22) quadratic programs. The KKT optimality conditions for quadratic programming state that the pair $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is optimal for the primal (5.21) and dual (5.22) quadratic programs if and only if the following conditions hold:

$$\textbf{1. } \boldsymbol{A}\hat{\mathbf{x}} \leq \mathbf{b} \qquad \textbf{2. } \hat{\mathbf{y}}^T(\boldsymbol{A}\hat{\mathbf{x}} - \mathbf{b}) = 0 \qquad \textbf{3. } \boldsymbol{W}_\gamma \hat{\mathbf{x}} = -\mathbf{A}^T\hat{\mathbf{y}} - \mathbf{c} \qquad \textbf{4. } \hat{\mathbf{y}} \geq 0.$$

We showed in Section 5.3 that the dual update step $y_i := \theta_i^+$ in Algorithm 10 will guarantee that the last two KKT conditions are always satisfied. In other words, the primal and dual variables at iteration k, $(\mathbf{x}_k, \mathbf{y}_k)$, satisfy $\mathbf{y}_k \geq 0$ and $\boldsymbol{W}_\gamma \mathbf{x}_k = -\mathbf{A}^T\mathbf{y}_k - \mathbf{c}$. This means that $\mathbf{y}_k$ is always feasible for the dual objective (5.22), and by weak duality we have the following lower bound on the optimal solution to objective (5.21)

$$D(\mathbf{y}_k) = -\mathbf{b}^T\mathbf{y}_k - \frac{1}{2}(\mathbf{A}^T\mathbf{y}_k + \mathbf{c})^T\boldsymbol{W}_\gamma^{-1}(\mathbf{A}^T\mathbf{y}_k + \mathbf{c}) = -\mathbf{b}^T\mathbf{y}_k - \frac{1}{2}\mathbf{x}_k^T\boldsymbol{W}_\gamma\mathbf{x}_k. \tag{5.24}$$

We also proved in Section 5.3 that performing Dykstra's method is equivalent to applying a coordinate ascent procedure on the dual quadratic program (5.22). This means that $D(\mathbf{y}_k)$ is a strictly increasing lower bound that converges to $\hat{Q}$. Meanwhile, $Q(\mathbf{x}_k)$ does not necessarily upper bound $\hat{Q}$ since $\mathbf{x}_k$ is not necessarily feasible. However, $\mathbf{x}_k$ converges to the optimal primal solution, so as the algorithm progresses, the maximum constraint violation of $\mathbf{x}_k$ decreases to zero. In practice, once $\mathbf{x}_k$ has satisfied constraints to within a small enough tolerance we treat $Q(\mathbf{x}_k)$ as an upper bound.

After each pass through the constraints we check the primal-dual gap ω_k and maximum constraint violation ρ_k, given by

$$\omega_k = \frac{D(\mathbf{y}_k) - Q(\mathbf{x}_k)}{D(\mathbf{y}_k)}$$
$$\rho_k = \max_t(b_t - \mathbf{a}_t^T\mathbf{x}_k).$$

Together these two scores provide an indication for how close $(\mathbf{x}_k, \mathbf{y}_k)$ are to convergence.

Computing ω_k and ρ_k To compute ω_k in practice, we note that $\frac{1}{2}\mathbf{x}_k^T\boldsymbol{W}_\gamma\mathbf{x}_k$ appears in both $Q(\mathbf{x}_k)$ and $D(\mathbf{y}_k)$ (see (5.24)). This term, as well as the term $\mathbf{c}^T\mathbf{x}$ can be easily computed by iterating through the $O(n^2)$ entries of $\mathbf{x}$. Finding $\mathbf{b}^T\mathbf{y}_k$ could theoretically involve $O(n^3)$ computations, but this can be done during the main loop of Dykstra's algorithm by continually updating a variable that adds up terms of the form $y_t b_t$ whenever y_t and b_t are both nonzero for a particular constraint t. Note that in most of the problems we have considered here, $b_t = 0$ for the majority of the constraints. For example, in the sparsest cut relaxation (5.23), b_t is only nonzero for the constraint $\sum_{i<j} x_{ij} = n$.

Computing ρ_k requires we iterate though the entire constraint set and simply record the worst constraint violation. Since this requires visiting $O(n^3)$ constraints, it may take nearly as long as a full pass through constraints using Dykstra's method. In practice we therefore just check each constraint until we come across one that violates the desired constraint tolerance, if such a constraint exists. At this point we know the algorithm did not converge, and there is no need to continue checking constraint violations. Every 10-20 passes through the algorithm we perform a full constraint check to report on the progress of the algorithm.

5.4.4 Entrywise Rounding Procedure

In practice we could simply run Dykstra's iteration until both ω_k and ρ_k fall below user-defined tolerances. We additionally incorporate another step in out convergence check that significantly improves the algorithm's performance in practice. Because $\mathbf{x}_k \to \hat{\mathbf{x}}$, we know that after a certain point, the maximum entrywise difference between $\hat{\mathbf{x}}$ and $\mathbf{x}_k$ will be arbitrarily small. Therefore, once both ρ_k and $|\omega_k|$ have dropped below a given tolerance, we will test for convergence by rounding every entry of $\mathbf{x}_k$ to r significant figures for a range of values of r: $\mathbf{x}_r = round(\mathbf{x}_k, r)$. As long as $\mathbf{x}_k$ is close enough to optimality and we have chosen the proper r, $\mathbf{x}_r$ will satisfy constraints to within the desired tolerance and will have an objective exactly or nearly equal to the best lower bound we have for $\hat{Q}$: $[D(\mathbf{y}_k) - Q(\mathbf{x}_r)]/D(\mathbf{y}_k) \leq \epsilon$. If $\mathbf{x}_r$ does not satisfy constraints or has a poor objective score, we simply discard $\mathbf{x}_r$ and continue with $\mathbf{x}_k$ and the original Dykstra iteration. Even if this rounding procedure is always unsuccessful, we simply fall back on the iterates $(\mathbf{x}_k, \mathbf{y}_k)$ until ω_k and ρ_k eventually fall below the given tolerance. In practice however, we do find that specifically for the sparsest cut relaxation, the rounding procedure dramatically improves both the runtime of the method as well as constraint satisfaction.

We highlight the fact that when checking whether we are close enough to convergence to apply the entrywise rounding step, we consider the absolute value of ω_k and not ω_k itself. Recall that this value may be negative if $Q(\mathbf{x}_k)$ is not an upper bound on the optimal objective. Often in practice we find that by the time we are close to convergence, $Q(\mathbf{x}_k)$ is indeed an upper bound and ω_k is a small positive number. However, we also observe cases where $\mathbf{x}_k$ is infeasible and ω_k is negative, but $|\omega_k|$ is small and our entrywise rounding procedure succeeds in producing a feasible point $\mathbf{x}_r$. When this happens, the duality gap between $D(\mathbf{y}_k)$ and $Q(\mathbf{x}_r)$ is guaranteed to be non-negative, and tells us how close the feasible vector $\mathbf{x}_r$ is to the optimal solution.

5.5 Approximation Guarantees for Clustering Objectives

The results of Mangasarian [1984] confirm that for all γ greater than some $\gamma_0 > 0$, the original linear program (5.20) and the quadratic regularization (5.21) will have the same optimal solution. However, it is challenging to compute γ_0 in practice, and if we set γ to be too high then this may lead to very slow convergence for solving QP (5.21) using projection methods. Sra et al. [2005] suggest ways to set γ for variants of the metric nearness problem based on empirical observations, but no approximation guarantees are provided. Here we outline a set of results which show how to set $\boldsymbol{W}$ and γ in order to obtain specific guarantees for approximating specific graph clustering objectives. These results hold for all $\gamma > 0$, whether larger or smaller than the unknown value γ_0. We begin with a general theorem that provides a useful strategy for obtaining approximation guarantees for a large class of linear programs.

Theorem 5.5.1 *Let* $\boldsymbol{A} \in \mathbb{R}^{M \times N}$, $\mathbf{b} \in \mathbb{R}^N$, $\mathbf{c} \in \mathbb{R}^N_{>0}$, *and* $\mathcal{A} = \{\mathbf{x} \in \mathbb{R}^N : \boldsymbol{A}\mathbf{x} \leq \mathbf{b}\}$. *Denote* $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x} \in \mathcal{A}} \mathbf{c}^T\mathbf{x}$ *and assume that all entries of* $\mathbf{x}^*$ *are between* 0 *and* $B > 0$. *Let* $\boldsymbol{W}$ *be a diagonal matrix with entries* $\boldsymbol{W}_{ii} = c_i > 0$ *and let* $\hat{\mathbf{x}} = \operatorname{argmin}_{\mathbf{x} \in \mathcal{A}}[\mathbf{c}^T\mathbf{x} + 1/(2\gamma)\mathbf{x}^T\boldsymbol{W}\mathbf{x}]$. *Then*

$$\mathbf{c}^T\mathbf{x}^* \leq \mathbf{c}^T\hat{\mathbf{x}} \leq \mathbf{c}^T\mathbf{x}^*(1 + B/(2\gamma)).$$

Proof Vectors $\hat{\mathbf{x}}$ and $\mathbf{x}^*$ are optimal for their respective problems, meaning that

$$\mathbf{c}^T\mathbf{x}^* \leq \mathbf{c}^T\hat{\mathbf{x}} \leq \mathbf{c}^T\hat{\mathbf{x}} + \frac{1}{2\gamma}\hat{\mathbf{x}}^T\boldsymbol{W}\hat{\mathbf{x}} \leq \mathbf{c}^T\mathbf{x}^* + \frac{1}{2\gamma}(\mathbf{x}^*)^T\boldsymbol{W}\mathbf{x}^*.$$

The proof follows from combining these inequalities with a bound on the second term on the far right. By our construction of $\boldsymbol{W}$ and the bounds we assume hold for $\mathbf{x}^*$, we have:

$$(\mathbf{x}^*)^T\boldsymbol{W}\mathbf{x}^* = \sum_{i=1}^n c_i(\mathbf{x}_i^*)^2 = B^2\sum_{i=1}^n c_i(\mathbf{x}_i^*/B)^2 \leq B\sum_{i=1}^n c_i\mathbf{x}_i^* = B\mathbf{c}^T\mathbf{x}^*$$

where the second to last step holds because $0 \leq x_i^* \leq B \implies (x_i^*/B)^2 < (x_i^*/B)$. ■

5.5.1 Cluster Deletion Approximation

Theorem 5.5.1 directly implies a result for the cluster deletion LP relaxation (5.5). Cluster deletion has a variable x_{ij} for each edge $(i,j) \in E$. The objective can be written $\mathbf{e}^T\mathbf{x} = \sum_{(i,j)\in E} x_{ij}$, where $\mathbf{e}$ is the all ones vector. Since the LP also includes constraints $x_{ij} \in [0,1]$, the assumptions of

Theorem 5.5.1 hold with $\boldsymbol{W}$ equal to the identity matrix and $B = 1$. This means that a Dykstra-based projection method will produce a solution within a factor $(1 + 1/(2\gamma))$ of the optimal cluster deletion LP relaxation. Coupling this result with the LP rounding procedure we outlined in Chapter 4, we can obtain a $(2 + 1/\gamma)$ approximation for cluster deletion in practice.

5.5.2 Correlation Clustering

Consider a correlation clustering problem on n nodes where each pair of nodes (i, j) is either strictly similar or strictly dissimilar, with a nonzero weight $w_{ij} > 0$. That is, exactly one of the weights (w_{ij}^-, w_{ij}^+) is positive and the other is zero. We focus on the LP relaxation for this problem given in the form of the ℓ_1 metric nearness LP (5.8). We slightly alter this formulation by performing a change of variables $y_{ij} = x_{ij} - d_{ij}$. The LP can then be written equivalently as:

$$\begin{array}{lll} \text{minimize} & \sum_{i<j} w_{ij} m_{ij} & \\ \text{subject to} & y_{ij} - y_{ik} - y_{jk} \leq b_{ijk} & \text{for all } i, j, k \\ & y_{ij} \leq m_{ij} & \text{for all } i, j \\ & -y_{ij} \leq m_{ij} & \text{for all } i, j \end{array} \tag{5.25}$$

where $b_{ijk} = -d_{ij} + d_{ik} + d_{jk}$ is defined so that the implicit variables $x_{ij} = y_{ij} + d_{ij}$ satisfy triangle inequalities. Recall that $d_{ij} \in \{0, 1\}$ is one if and only if there is a negative edge between nodes i and j. To write this LP in the format of (5.20), we as usual use $\mathbf{x}$ to represent the set of variables of the linear program. However, for this problem we must take care to note that $\mathbf{x}$ does not represent a linearization of the x_{ij} distance variables, but instead stores both y_{ij} and m_{ij} variables. More precisely, to relate (5.25) to the format of LP (5.20), we set $\mathbf{x} = \begin{bmatrix} \mathbf{y} & \mathbf{m} \end{bmatrix}^T$ and $\mathbf{c} = \begin{bmatrix} \mathbf{0} & \mathbf{w} \end{bmatrix}^T$, where $\mathbf{y}, \mathbf{m}$ represent linearizations of the doubly-indexed (y_{ij}) and (m_{ij}) variables, and $\mathbf{w} = (w_{ij})$ is the vector of positive weights for the node pairs. Rather than minimizing $\mathbf{c}^T\mathbf{x} = \sum_{i<j} w_{ij} m_{ij}$ we have a method that can minimize the quadratic objective $\mathbf{c}^T\mathbf{x} + \frac{1}{2\gamma}\mathbf{x}^T\boldsymbol{W}\mathbf{x}$ over the same constraint set. We construct a weight matrix that contains two copies of the weight vector $\mathbf{w}$, one to match up with the $\mathbf{y}$ vector and one corresponding to the $\mathbf{m}$ vector:

$$\boldsymbol{W} = \begin{bmatrix} diag(\mathbf{w}) & \mathbf{0} \\ \mathbf{0} & diag(\mathbf{w}) \end{bmatrix}. \tag{5.26}$$

The quadratic regularization of the original LP objective is then

$$\min_{\mathbf{x}} \mathbf{c}^T\mathbf{x} + \frac{1}{2\gamma}\mathbf{x}^T\boldsymbol{W}\mathbf{x} = \min_{(m_{ij}),(y_{ij})} \sum_{i<j} w_{ij} m_{ij} + \frac{1}{2\gamma}\sum_{i<j} w_{ij} m_{ij}^2 + \frac{1}{2\gamma}\sum_{i<j} w_{ij} y_{ij}^2. \tag{5.27}$$

For both (5.25) and (5.27), the constraints enforce $|y_{ij}| \leq m_{ij}$. Since the objectives are being minimized, this guarantees $m_{ij} = |y_{ij}|$ at optimality. This implies that $m_{ij}^2 = y_{ij}^2$, which is the reason we choose to introduce variables $y_{ij} = x_{ij} - d_{ij}$ rather than working directly with x_{ij}. Introducing y_{ij} variables allows us to replace y_{ij}^2 with m_{ij}^2 in (5.27), and re-write the objective using terms only involving m_{ij} variables:

$$\min_{(m_{ij}),(y_{ij})} \sum_{i<j} w_{ij} m_{ij} + \frac{1}{\gamma} \sum_{i<j} w_{ij} m_{ij}^2. \tag{5.28}$$

Let (m_{ij}^*) and (y_{ij}^*) be optimal for (5.25) and $(\hat{m}_{ij}), (\hat{y}_{ij})$ be optimal for (5.28). Then

$$\sum_{i<j} w_{ij} \hat{m}_{ij} + \frac{1}{\gamma} \sum_{i<j} w_{ij} \hat{m}_{ij}^2 \leq \sum_{i<j} w_{ij} m_{ij}^* + \frac{1}{\gamma} \sum_{i<j} w_{ij} (m_{ij}^*)^2 \leq \left(1 + \frac{1}{\gamma}\right) \sum_{i<j} w_{ij} m_{ij}^*. \tag{5.29}$$

In the last step above we have used the fact that $m_{ij}^* = |y_{ij}^*| \leq 1 \implies m_{ij}^* \leq (m_{ij}^*)^2$ (see the proof of Theorem 5.2.1 for why $|y_{ij}^*| \leq 1$). This proves an approximation result for correlation clustering:

Theorem 5.5.2 *Let* (m_{ij}^*) *and* (y_{ij}^*) *be the optimal solution vectors for the correlation clustering LP relaxation given in* (5.25) *and* $(\hat{m}_{ij}), (\hat{y}_{ij})$ *be the optimal solution to the related QP* (5.27). *Then*

$$\sum_{i<j} w_{ij} m_{ij}^* \leq \sum_{i<j} w_{ij} \hat{m}_{ij} \leq \left(1 + \frac{1}{\gamma}\right) \sum_{i<j} w_{ij} m_{ij}^*.$$

Therefore, given any rounding procedure for the original LP that gives a factor p approximation for a correlation clustering problem, we can instead solve the related QP using projection methods to obtain a factor $p(1 + 1/\gamma)$ approximation. For weighted correlation clustering, the best rounding procedures guarantee an $O(\log n)$ approximation [Charikar et al., 2005, Demaine et al., 2006], so this can still be achieved even if we use a small value for γ.

5.5.3 Sparsest Cut

The Leighton-Rao linear programming relaxation for sparsest cut is presented in (5.3). This LP has a variable x_{ij} for every pair of distinct nodes $i < j$ in some unweighted graph $G = (V, E)$. Let $\mathbf{x} = (x_{ij})$ be a linearization of these distance variables, and define $\mathbf{c} = (c_{ij})$ to be the adjacency indicators, i.e.

$$c_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Then the objective can be written in the familiar format $\min_{\mathbf{x}} \mathbf{c}^T \mathbf{x}$. If we assume we have chosen a weight matrix $\boldsymbol{W}$ and a parameter $\gamma > 0$, the regularized version of (5.3) has objective

$$\min_{\mathbf{x}} \sum_{i<j} c_{ij} x_{ij} + \frac{1}{2\gamma} \sum_{i<j} w_{ij} x_{ij}^2.$$

As we have seen in Theorem 5.5.1, it seems fitting for c_{ij}, the coefficients of x_{ij}, to match up with w_{ij}, the coefficients of x_{ij}^2. However, many of the c_{ij} variables are zero, so it will not work to choose

$w_{ij} = c_{ij}$, since $\boldsymbol{W}$ needs to be positive definite in order for us to apply our projection method. Instead we introduce another parameter $\lambda \in (0,1)$ and define a set of weights $\mathbf{w} = (w_{ij})$ by

$$w_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ \lambda & \text{otherwise.} \end{cases}$$

In this way, the weight w_{ij} is still positive but can be *near* zero (i.e. near c_{ij}) when $(i,j) \notin E$. We can then prove the following approximation result:

Theorem 5.5.3 *Let $G = (V,E)$ be a connected graph with $n = |V| > 4$. Let ϕ^* be the sparsest cut score in G and assume that each side of the sparsest cut partition has at least 2 nodes. Let $\gamma > 0$ and $\boldsymbol{W} = diag(\mathbf{w})$ be defined as above for a given $\lambda \in (0,1)$, and let $\mathcal{A}$ denote the set of constraints from the Leighton-Rao LP relaxation for sparsest cut. Then*

$$\min_{\mathbf{x} \in \mathcal{A}} \mathbf{c}^T\mathbf{x} + \frac{1}{2\gamma}\mathbf{x}^T\boldsymbol{W}\mathbf{x} \leq \left(1 + \frac{1+\lambda n}{2\gamma}\right)\phi^*.$$

Proof The quadratic regularization of the sparsest cut LP relaxation is

$$\begin{array}{lll} \text{minimize} & \sum_{i<j} c_{ij}x_{ij} + (1/2\gamma)\sum_{i<j} w_{ij}x_{ij}^2 & \\ \text{subject to} & \sum_{i<j} x_{ij} = n & \\ & x_{ij} \leq x_{ik} + x_{jk} & \text{for all } i,j,k \\ & x_{ij} \geq 0 & \text{for all } i,j. \end{array} \tag{5.30}$$

The result we prove here relates the optimal solution of (5.30) directly back to the minimum cut sparsity ϕ^*, rather than back to the LP relaxation of sparsest cut (5.3). This makes sense given that our purpose in solving these convex relaxations is to develop approximation results for the original NP-hard sparsest cut objective.

Let $S^* \subset V$ be the set of nodes inducing the sparsest cut partition of G, so that

$$\phi^* = \frac{\text{cut}(S^*)}{|S^*|} + \frac{\text{cut}(S^*)}{|\bar{S}^*|} = \frac{n\,\text{cut}(S^*)}{|S^*||\bar{S}^*|}.$$

Without loss of generality, assume $|S^*| \leq |\bar{S}^*|$. In the statement of the theorem we assume that G is connected, $n > 4$, and $|S^*| > 1$. The connectivity of G ensures the problem can't be trivially solved by finding a single connected component, and guarantees that $\text{cut}(S^*) \geq 1$. Together the remaining two assumptions guarantee that $\frac{n}{|S^*||\bar{S}^*|} \leq \frac{n}{2(n-2)} \leq 1$, which will be useful later in the proof. We will also use the fact that $\frac{n}{|S^*||\bar{S}^*|} \leq \frac{n\,\text{cut}(S^*)}{|S^*||\bar{S}^*|} = \phi^*$. Note that if $n \leq 4$, the problem is trivial to solve by checking all possible partitions, and if $|S^*| = 1$ then the sparsest cut problem is easy to solve by checking all n partitions that put a single node by itself.

In order to encode the optimal partition as a vector, define $\mathbf{s}^* = (s_{ij}^*)$ by

$$s_{ij}^* = \begin{cases} \frac{n}{|S^*||\bar{S}^*|} & \text{if nodes } i \text{ and } j \text{ are on opposite side of the partition } \{S^*, \bar{S}^*\} \\ 0 & \text{otherwise.} \end{cases}$$

Observe that this vector $\mathbf{s}^*$ satisfies the constraints of (5.30) and that

$$\sum_{i<j} c_{ij} s^*_{ij} = \sum_{(i,j)\in E} s^*_{ij} = \frac{\text{cut}(S^*)n}{|S^*||\bar{S}^*|} = \phi^*.$$

We can also prove a useful bound on the quadratic term in the objective:

$$\begin{aligned}(\mathbf{s}^*)^T \boldsymbol{W} \mathbf{s}^* &= \sum_{i<j} w_{ij} (s^*_{ij})^2 \\ &= \sum_{(i,j)\in E} (s^*_{ij})^2 + \sum_{(i,j)\notin E} \lambda (s^*_{ij})^2 \\ &< \sum_{(i,j)\in E} (s^*_{ij})^2 + \sum_{i<j} \lambda (s^*_{ij})^2 \\ &= \text{cut}(S^*) \frac{n^2}{|S^*|^2 |\bar{S}^*|^2} + \lambda |S^*||\bar{S}^*| \frac{n^2}{|S^*|^2|\bar{S}^*|^2} \\ &= \phi * \frac{n}{|S^*||\bar{S}^*|} + \lambda n \frac{n}{|S^*||\bar{S}^*|} \\ &\leq \phi^*(1+\lambda n),\end{aligned}$$

where we have used the fact that $n/(|S^*||\bar{S}^*|) \leq \min\{1, \phi *\}$ because of our simple assumptions on G. With more restrictive assumptions and careful analysis we could obtain even better approximation guarantees, but our aim is simply to show for now that we can eventually obtain an $O(\log n)$ approximation for sparsest cut by minimizing a quadratic program (5.30) instead of the original Leighton-Rao LP (5.3).

Let $\hat{\mathbf{x}}$ be the optimal solution for the QP (5.30), and recall that $\mathbf{s}^*$ is another feasible point. We combine the bounds shown above to prove the final result:

$$\sum_{i<j} c_{ij}\hat{x}_{ij} < \sum_{i<j} c_{ij}\hat{x}_{ij} + \frac{1}{2\gamma}\sum_{i<j} w_{ij}\hat{x}_{ij}^2 \leq \sum_{i<j} c_{ij} s^*_{ij} + \frac{1}{2\gamma}\sum_{i<j} w_{ij}(s^*_{ij})^2 \leq \phi^* + \frac{1}{2\gamma}(1+\lambda n)\phi^*.$$

■

5.6 Improved a Posteriori Approximations

The approximation bounds in the previous section provide helpful suggestions for how to set parameters γ and $\boldsymbol{W}$ before running Dykstra's projection algorithm on a quadratic regularization of a metric constrained LP. Once we have chosen these parameters and solved the quadratic program, we would like to see if we can improve these guarantees using the output solution for the QP.

5.6.1 A First Strategy for Improved Bounds

Consider again the optimal solutions to the LP and QP given by

$$\begin{aligned}\mathbf{x}^* &= \text{argmin}_{\mathcal{A}} \;\; \mathbf{c}^T\mathbf{x}\\ \hat{\mathbf{x}} &= \text{argmin}_{\mathcal{A}} \;\; \mathbf{c}^T\mathbf{x} + \frac{1}{2\gamma}\mathbf{x}^T\boldsymbol{W}\mathbf{x}.\end{aligned}$$

where $\mathcal{A} = \{\mathbf{x} \in \mathbb{R}^N : \boldsymbol{A}\mathbf{x} \leq \mathbf{b}\}$ is the set of feasible solutions. For each of the NP-hard graph clustering objectives we have considered, we have proven a sequence of inequalities of the form

$$\mathbf{c}^T\mathbf{x}^* \leq \mathbf{c}^T\hat{\mathbf{x}} \leq \mathbf{c}^T\hat{\mathbf{x}} + \frac{1}{2\gamma}\hat{\mathbf{x}}^T\boldsymbol{W}\hat{\mathbf{x}} \leq \mathbf{c}^T\mathbf{x}^* + \frac{1}{2\gamma}(\mathbf{x}^*)^T\boldsymbol{W}(\mathbf{x}^*) \leq (1+A)OPT\,,$$

where A is a term in the approximation factor (e.g. $1/\gamma$, $1/(2\gamma)$, $(1+\lambda n)/\gamma$) and OPT is the optimal score for the NP-hard objective. If we have already computed $\hat{\mathbf{x}}$, we can improve this approximation result by computing

$$R = \frac{\hat{\mathbf{x}}^T\boldsymbol{W}\hat{\mathbf{x}}}{2\gamma\mathbf{c}^T\hat{\mathbf{x}}}\,.$$

We then get an improved approximation guarantee:

$$\mathbf{c}^T\hat{\mathbf{x}} + \frac{1}{2\gamma}\hat{\mathbf{x}}^T\boldsymbol{W}\hat{\mathbf{x}} = (1+R)\mathbf{c}^T\hat{\mathbf{x}} \implies \mathbf{c}^T\hat{\mathbf{x}} \leq \frac{(1+A)}{(1+R)}OPT\,.$$

In some cases R will be small and this improvement will be minimal. However, intuitively we can see that in some special cases R may be large enough to significantly improve the approximation factor. For example, it may be the case that for some correlation clustering relaxation, we choose γ large enough so that the optimal solution to the QP, $\hat{\mathbf{m}}$, and the optimal solution to the LP, $\mathbf{m}^*$, are actually identical. Even after computing $\hat{\mathbf{m}}$ we may not realize that $\mathbf{c}^T\hat{\mathbf{x}} = \mathbf{c}^T\mathbf{x}^*$. However, in some cases, a significant proportion of the $m^*_{ij} = \hat{m}_{ij}$ variables will close to zero or close to one. Thus, $\hat{m}_{ij} \approx \hat{m}_{ij}^2$ for many pairs i, j. For the correlation clustering relaxation this will mean that $\hat{\mathbf{x}}\boldsymbol{W}\hat{\mathbf{x}} \approx 2\mathbf{c}^T\hat{\mathbf{x}} \implies R \approx A$. Even in cases where $\mathbf{x}^*$ and $\hat{\mathbf{x}}$ are not identical but very close, similar reasoning shows that the above a posteriori approximation result may be much better than the a priori $(1+A)$ approximation. We note that our approximation results for correlation clustering in the experiments section are greatly aided by this a posteriori guarantee.

5.6.2 Improved Guarantees by Solving a Small LP

We outline one more approach for getting improved approximation guarantees, this time based on a careful consideration of dual variables $\hat{\mathbf{y}}$ computed by Dykstra's method. This result requires a more sophisticated approach than the guarantee given in the last section. This will be extremely helpful for providing strong a posteriori guarantees when solving the quadratic relaxation of sparsest cut.

Once more we consider our initial linear program, which we assume is too challenging to solve using standard optimization software because of memory requirements:

$$\begin{aligned} \min_{\mathbf{x}}\ & \mathbf{c}^T\mathbf{x} \\ \text{s.t. }\ & \boldsymbol{A}\mathbf{x} \leq \mathbf{b}. \end{aligned} \tag{5.31}$$

We again let $\mathbf{x}^*$ denote the (unknown) optimizer for (5.31). In practice, we solve a quadratic regularization:

$$\begin{aligned} \min_{\mathbf{x}}\ & \mathbf{c}^T\mathbf{x} + \frac{1}{2\gamma}\mathbf{x}^T\boldsymbol{W}\mathbf{x} \\ \text{s.t. }\ & \boldsymbol{A}\mathbf{x} \leq \mathbf{b}. \end{aligned} \tag{5.32}$$

We solve (5.32) by finding a primal-dual pair of vectors $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ satisfying KKT conditions. In particular, as noted in previous sections, these vectors satisfy

$$\frac{1}{\gamma}\boldsymbol{W}\hat{\mathbf{x}} = -\mathbf{A}^T\hat{\mathbf{y}} - \mathbf{c} \tag{5.33}$$

$$-\mathbf{b}^T\hat{\mathbf{y}} - \frac{1}{2\gamma}\hat{\mathbf{x}}^T\boldsymbol{W}\hat{\mathbf{x}} = \mathbf{c}^T\hat{\mathbf{x}} + \frac{1}{2\gamma}\hat{\mathbf{x}}^T\boldsymbol{W}\hat{\mathbf{x}}. \tag{5.34}$$

Given this setup, we prove a new theorem for obtaining a lower bound on $\mathbf{c}^T\mathbf{x}^*$ by considering $\hat{\mathbf{y}}$ and solving another small, less expensive LP.

Theorem 5.6.1 *Given* $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$*, set* $\hat{\mathbf{p}} = 1/\gamma\boldsymbol{W}\hat{\mathbf{x}}$ *and let* $\tilde{\mathbf{x}}$ *be the optimal solution to the following new linear program:*

$$\begin{aligned} \max_{\mathbf{x}}\ & \hat{\mathbf{p}}^T\mathbf{x} \\ \textit{s.t. }\ & \mathbf{c}^T\mathbf{x} \leq \mathbf{c}^T\hat{\mathbf{x}} \\ & \mathbf{x} \in \mathcal{B} \end{aligned} \tag{5.35}$$

where $\mathcal{B}$ *is any set which is guaranteed to contain* $\mathbf{x}^*$ *(i.e.* $\mathcal{B}$ *encodes a subset of constraints that are known to be satisfied by* $\mathbf{x}^*$*). Then we have the following lower bound on the optimal solution to* (5.31)*:*

$$-\mathbf{b}^T\hat{\mathbf{y}} - \hat{\mathbf{p}}^T\tilde{\mathbf{x}} \leq \mathbf{c}^T\mathbf{x}^*. \tag{5.36}$$

Furthermore, if $\mathbf{x}^* = \hat{\mathbf{x}} = \tilde{\mathbf{x}}$*, then this bound is tight.*

Proof The dual of the original linear program (5.31) is

$$\begin{aligned} \max\ & -\mathbf{b}^T\mathbf{y} \\ \text{s.t. }\ & -\boldsymbol{A}^T\mathbf{y} - \mathbf{c} = 0 \\ & \mathbf{y} \geq 0. \end{aligned} \tag{5.37}$$

One way to obtain a lower bound on $\mathbf{c}^T\mathbf{x}^*$ would be to find some feasible point $\mathbf{y}$ for (5.37), in which case $-\mathbf{b}^T\mathbf{y} \leq \mathbf{c}^T\mathbf{x}^*$. Note that we have access to a vector $\hat{\mathbf{y}}$ satisfying $\hat{\mathbf{y}} \geq 0$ and $\mathbf{A}^T\hat{\mathbf{y}} - \mathbf{c} = \hat{\mathbf{p}} = (1/\gamma)\boldsymbol{W}\hat{\mathbf{x}}$. This $\hat{\mathbf{y}}$ is not feasible for (5.37), but we note that if the entries of $\hat{\mathbf{p}}$ are very small (which they will be for large γ), then the constraint $\mathbf{A}^T\mathbf{y} - \mathbf{c} = 0$ is *nearly* satisfied by $\hat{\mathbf{y}}$. If we define a new vector $\hat{\mathbf{c}} = \mathbf{c} + \hat{\mathbf{p}}$, then we can observe that $\hat{\mathbf{y}}$ is feasible for a slightly perturbed linear program:

$$\begin{aligned} \max \ & -\mathbf{b}^T\mathbf{y} \\ \text{s.t.} \ & -\boldsymbol{A}^T\mathbf{y} - \hat{\mathbf{c}} = 0 \\ & \mathbf{y} \geq 0. \end{aligned} \tag{5.38}$$

We realize that this is the dual of a slight perturbation of the original LP (5.31):

$$\begin{aligned} \min_{\mathbf{x}} \ & \hat{\mathbf{c}}^T\mathbf{x} \\ \text{s.t.} \ & \boldsymbol{A}\mathbf{x} \leq \mathbf{b}. \end{aligned} \tag{5.39}$$

Since $\hat{\mathbf{y}}$ is feasible for (5.38) and $\mathbf{x}^*$ is feasible for (5.39), we have the following inequality:

$$-\mathbf{b}^T\hat{\mathbf{y}} \leq \hat{\mathbf{c}}^T\mathbf{x}^* = \mathbf{c}^T\mathbf{x}^* + \hat{\mathbf{p}}\mathbf{x}^*. \tag{5.40}$$

Finally, observe that $\mathbf{x}^*$ is feasible for the LP (5.35) defined in the statement of the theorem, and therefore $\hat{\mathbf{p}}\mathbf{x}^* \leq \hat{\mathbf{p}}\tilde{\mathbf{x}}$. Combining this fact with (5.40) we get our final result:

$$-\mathbf{b}^T\hat{\mathbf{y}} \leq \mathbf{c}^T\mathbf{x}^* + \hat{\mathbf{p}}\mathbf{x}^* \leq \mathbf{c}^T\mathbf{x}^* + \hat{\mathbf{p}}\tilde{\mathbf{x}} \implies -\mathbf{b}^T\hat{\mathbf{y}} - \hat{\mathbf{p}}^T\tilde{\mathbf{x}} \leq \mathbf{c}^T\mathbf{x}^*.$$

If we happen to choose $\gamma > 0$ and $\boldsymbol{W}$ in such a way that $\mathbf{x}^* = \hat{\mathbf{x}}$, and then pick a set $\mathcal{B}$ so that $\tilde{\mathbf{x}} = \mathbf{x}^*$, then property (5.34) ensures that this bound will be tight. ■

Typically it will be difficult to choose parameters in such a way that $\mathbf{x}^* = \tilde{\mathbf{x}} = \hat{\mathbf{x}}$. However, the fact that this bound is tight for a certain choice of parameters is a good indication that the bound will not be too loose to be useful in practice, as long as we choose parameters carefully.

A Bound for Sparsest Cut Consider the quadratic regularization of the sparsest cut relaxation shown in (5.30), with diagonal weight matrix defined as in Section 5.5.3. Assume $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is the set of primal and dual variables obtained by solving the objective with Dykstra's method. We give a corollary to Theorem 5.6.1 that shows how to obtain good a posteriori approximations for how close $\mathbf{c}^T\hat{\mathbf{x}}$ is to the original LP relaxation of sparsest cut (5.3).

Corollary 5.6.2 *Let $\tilde{\mathbf{x}} = (\tilde{x}_{ij})$ be the optimizer for the following LP:*

$$\begin{array}{ll} \textit{maximize} & (1/\gamma)\sum_{i<j}(w_{ij}\hat{x}_{ij})x_{ij} \\ \textit{subject to} & \sum_{i<j} x_{ij} = n \\ & \sum_{(i,j)\in E} x_{ij} \leq \sum_{(i,j)\in E} \hat{x}_{ij} \\ & 0 \leq x_{ij} \leq \frac{n}{n-1} \qquad\qquad \textit{for all } i,j. \end{array} \tag{5.41}$$

Then

$$n\hat{y}_1 - n\hat{y}_2 - \frac{1}{\gamma}\sum_{i<j} w_{ij}\hat{x}_{ij}\tilde{x}_{ij} \leq \sum_{(i,j)\in E} x^*_{ij}$$

where $\hat{y}_1$ *and* $\hat{y}_2$ *are correction variables within the dual vector* $\hat{\mathbf{y}}$*, corresponding to the constraints* $\sum_{i<j} x_{ij} \leq n$ *and* $-\sum_{i<j} x_{ij} \leq -n$ *respectively. These two constraints combine to form the equality constraint* $\sum_{i<j} x_{ij} = n$*.*

Proof We just need to show that the assumptions of Theorem 5.6.1 are satisfied. Let x^*_{ij} be the optimal solution vector for the sparsest cut LP relaxation (5.3). Note that $x^*_{ij} \leq n/(n-1)$ for all i, j. If this were not the case and $x^*_{uv} > n/(n-1)$ for some pair (u, v), then there would exist $(n-2)$ nodes k distinct from u and v such that

$$\frac{n}{n-1} < x^*_{uv} \leq x^*_{uk} + x^*_{vk}.$$

Then

$$\sum_{i<j} x^*_{ij} \geq x^*_{uv} + \sum_{u\neq k\neq v} x^*_{uk} + x^*_{vk} > \frac{n}{n-1} + (n-2)\frac{n}{n-1} = n,$$

which contradicts the fact that the entries of $\mathbf{x}^*$ sum to n. We see then that all the constraints included in LP (5.41) are satisfied by x^*_{ij}, so the result holds. ■

5.7 Experiments

We implement Dykstra-based solvers for relaxations of sparsest cut (DykstraSC) and correlation clustering (DykstraCC) in the Julia programming language. We additionally show how to apply DykstraCC to obtain good empirical results for the modularity objective. Code for our algorithms and experiments are available online at `https://github.com/nveldt/MetricOptimization`. In practice we solve sparsest cut and modularity relaxations on graphs with up to 3086 nodes, and dense correlation clustering problems with up to 11,204 nodes. Thus, we solve optimization problems with up to 63 million edges and 7.0×10^{11} constraints. For correlation clustering, the previous best approaches managed to optimally solve instances with 13 million constraints [Miyauchi et al., 2018], or solve a different, but related, LP relaxation on problems with just over 4 million edges [Swoboda and Andres, 2017]. For the metric nearness problem, Sra et al. apply their triangle-fixing algorithm to solve metric nearness problems on random $n \times n$ dissimilarity matrices with n up to 5000 [Sra et al., 2005]. However, their method simply runs Dykstra's method until the change in the solution vector falls below a certain threshold. Since this approach does not take constraint satisfaction or duality gap into consideration, it comes with no output guarantees.

5.7.1 Implementation Details for Convergence Check

Both DykstraSC and DykstraCC use the detailed convergence check outlined in Section 5.4. To check whether $\tau > \rho_k$ for a constraint tolerance τ, after every pass through the constraints using Dykstra's method, our algorithm iterates again through the constraints and returns false as soon as it encounters a violated constraint, if one exists. Thus in early stages, the method can confirm that $\rho_k \geq \tau$ without visiting all $\Theta(n^3)$ constraints each time. Every C iterations, the algorithm performs a full pass through constraints to check the maximum violation ρ_k, and to apply the entrywise rounding procedure. For the rounding procedure, we set a preliminary threshold τ_0. After every C iterations, if $\rho_k < \tau_0$, the algorithm computes the rounded vector $\mathbf{x}_r$ for a range of r values. For sparsest cut, $\tau_0 = 0.1$ and $C = 10$, and we test each value of r from 2 to 6. In practice the rounding procedure significantly increases the method's performance in finding solutions with constraints satisfied to within machine precision. For correlation clustering we focus only on solving relaxations to within an overall constraint tolerance of 0.01 (or 0.001 for related modularity experiments), and we do not see any performance gains using the rounding procedure. Regardless, by design, the rounding procedure neither dominates the runtime nor affects the method's ability to converge using the standard Dykstra iterate. In our weighted correlation clustering experiments, we perform the rounding procedure every $C = 20$ steps, leading to an overall runtime increase of around 5-7%. For modularity clustering, we check every $C = 10$ steps, leading to an increase between 10-15%.

5.7.2 Using Gurobi Software

In our experiments our aim is obtain high-quality solutions to metric constrained relaxations of graph clustering problems. Thus we compare primarily against solving the same correlation clustering and sparsest cut relaxations using commercial Gurobi software. Gurobi possesses a number of solvers for LPs. In practice we separately run Gurobi's barrier method (i.e., the interior-point solver), the primal simplex method, and the dual simplex method. For the interior-point method, Gurobi's default setting is to convert any solution it finds to a basic feasible solution, but we turn this setting off since we do not require this of our own solver and we are simply interested in finding any solution to the LP. In practice we find that the interior-point solver is the fastest. The runtimes we report do not include the time spent forming the constraint matrix. This in and of itself is an expensive task that must be taken into account when using off-the-shelf software to solve problems of this form.

Lazy-Constraint Method Both for sparsest cut and correlation clustering we also test out an additional *lazy-constraint* method when employing Gurobi software. This procedure works as follows:

1. Given a metric constrained LP, solve the objective on a subproblem that includes all the same constraints *except* metric constraints.
2. Given the solution to the subproblem, check for violations in the metric constraints. Update the constraint set to include all such violated constraints. Re-solve the LP using black-box software on the updated set of constraints.
3. Continually re-solve the problem, check for violations, and update the constraint set. If we reach a point when all original metric constraints are satisfied before the algorithm fails due to memory issues, the solution is guaranteed to be the solution to the original metric constrained LP.

This procedure in some cases leads to significantly improved runtimes since it may permit us to solve the original LP without ever forming the entire $O(n^3) \times O(n^2)$ constraint matrix. Quite often we find, especially for correlation clustering problems, that many constraints will naturally be satisfied without explicitly including them in the problem setup. However, for the sparsest cut relaxation, we find that a large number of metric constraints are tight at optimality, and therefore must be included explicitly in the constraint set. In practice therefore we observe that for the sparsest cut relaxation, Gurobi continues to add constraints until a very large percentage of the original constraints are included explicitly. It therefore typically does not save time or space to repeatedly solve smaller subproblems.

5.7.3 Real-world Graphs

In our experiments we use real-world networks obtained almost exclusively from the SuiteSparse Matrix Collection [Davis and Hu, 2011] and the SNAP repository [Leskovec and Krevl, 2014], with one graph (Vassar85) from the Facebook100 networks [Traud et al., 2012]. The graphs we experiment on come from numerous domains,

- **Citation networks**: SmallW and SmaGri [Batagelj and Mrvar, 2006]
- **Collaboration networks**:caGrQc, caHepTh, caHepPh [Leskovec et al., 2007]; Netscience [Newman, 2006], and Erdos991 [Batagelj and Mrvar, 2006]
- **Web-based graphs**: Harvard500 (web matrix) [Moler, 2004], Polblogs (links between political blogs) [Adamic and Glance, 2005], Email (email correspondence graph) [Guimerà et al., 2003]

- **Biological networks**: C. El-Neural (neural network for nematode C. Elegans) [White et al., 1986, Watts and Strogatz, 1998], C. El-Meta (metabolic network for C. Elegans) [Duch and Arenas, 2005]
- **Networks based on words and books**: Roget (thesaurus associations) [Batagelj and Mrvar, 2006], Polbooks (co-purchased books on US politics) [Krebs, 2004], Journals (Slovenian magazines and journals) [Batagelj and Mrvar, 2006], Adjnoun (adjacent nouns and adjectives in the novel "David Copperfield") [Newman, 2006], Les Mis (co-appearing characters in the novel "Les Miserables") [Knuth, 1993]
- **Other networks**: Power (power grid network) [Batagelj and Mrvar, 2006], Dolphins (social network of dolphins) [Lusseau et al., 2003], USAir97 (US flights graph) [Batagelj and Mrvar, 2006], Football (network of American football games) [Girvan and Newman, 2002], Jazz (network of jazz musicians) [Gleiser and Danon, 2003].

Before running experiments, we make all edges undirected, remove edge weights, and find the largest connected component to ensure we are always working with connected, unweighted, and undirected networks.

5.7.4 The Sparsest Cut Relaxation

We run DykstraSC on graphs ranging in size from 62 to 3068 nodes. Our machine has two 14-core 2.66 GHz Xeon processors and for ease of reproducibility we limit experiments to 100GB of RAM. For all datasets but the largest we set $\gamma = 5$ and $\lambda = 1/n$. For Vassar85, the largest and hence most expensive problem, we use $\gamma = 2$ and $\lambda = 1/1000$. These parameter settings lead to a faster convergence, at the expense of a slightly worse approximation guarantee. Results are shown in Table 5.1. The last column of this table reports the ratio Δ between the LP objective output by our DykstraSC and the minimum LP score. In cases where we do not know Δ exactly, we report an upper bound computed using our a posteriori approximation guarantees (see Section 5.6.2). Figure 5.1 is a runtime plot for our DykstraSC experiments.

In Table 5.1 we see that Gurobi has an advantage on smaller graphs, but slows down and then runs out of memory once the graphs scale beyond a few hundred nodes. Since DykstraSC is in fact optimizing a quadratic regularization of the sparsest cut LP relaxation, we also report how close our solution is to the optimal LP solution, either by comparing against Gurobi or using our a posteriori approximation guarantee, presented in Corollary 5.6.2. In nearly all cases we are within 1% of the optimal LP solution, and in several cases our solver returns the optimal LP solution.

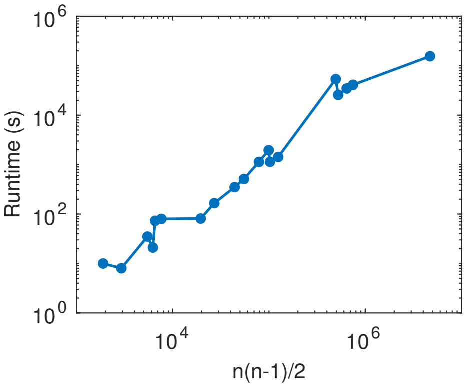


Figure 5.1.: Runtimes for DykstraSC on real-world graphs from Table 5.1. If n is the number of nodes in the graph, then DykstraSC solves for $n(n-1)/2$ distance scores.

Gurobi Performance When running Gurobi, for graphs with fewer than 500 nodes we have run all three solvers (interior-point, dual simplex, and primal simplex). We report times for the interior-point solver, since it proves to be the fastest in all cases. Gurobi runs out of memory when trying to form the entire constraint matrix for larger problems. We also test the lazy-constraint method to find it yields almost not benefit for the sparsest cut relaxation. For graphs smaller than Harvard500, where Gurobi was able to work with the entire constraint matrix, coupling the interior-point solver with the lazy-constraint procedure leads to much longer runtimes. Additionally, we find in all cases that by the time the lazy-constraint solver converged, well over half of the original constraint set had to be explicitly included in order to force all other metric constraints to be satisfied. Therefore, in addition to significantly worse runtimes, we see only a minor decrease in the memory requirement.

On larger graphs, the slight decrease in memory afforded by the lazy-constraint method does allows us to solve the sparsest cut relaxation on Harvard500, which was not possible when forming the entire constraint matrix up front. This is the only positive result we see for using this approach for the sparsest cut relaxation. However, it still requires solving a large number of expensive subproblems, leading to a runtime that is an order of magnitude slower than DykstraSC. We also tried the lazy-constraint approach on Roget, SmaGri, Email, and Polblogs. For all of these graphs, Gurobi spends a considerable amount of time solving subproblems, but still eventually runs out of memory before finding a solution. Due to this repeated failure to produce results on much smaller graphs, we did not attempt to run the lazy-constraint solver on Vassar85.

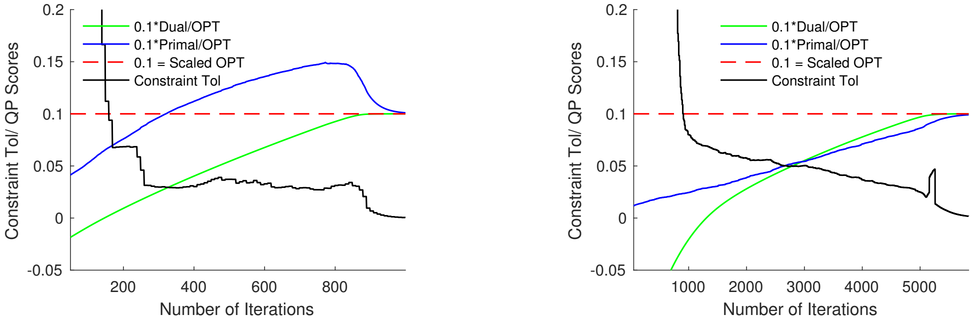
DykstraSC Convergence Plots Figure 5.2 displays convergence plots for DykstraSC on for several of the graphs. In all of the plots, a red dotted line represents the optimal objective score for the underlying quadratic objective. Scores for the dual objective $D(\mathbf{y}_k)$ are shown in green, and primal scores $Q(\mathbf{x}_k)$ are shown in blue. For each plot, we have in fact scaled the primal, dual, and optimal scores so that the optimal score is 0.1 for each graph. This allows us to then overlay the maximum constraint violation with a solid black line. Thus the same plot shows the progress of

Table 5.1.: We solve the LP relaxation for sparsest cut via DykstraSC on 19 graphs. Both DykstraSC and Gurobi (when it doesn't run out of memory) solve the problems to within a relative gap tolerance of 10^{-4}, and satisfy constraints to within machine precision. Time is given in seconds.

Graph	$\|V\|$	$\|E\|$	# constraints	Gurobi Time	Dykstra Time	Δ Approx
Dolphins	62	159	1.1×10^5	1	10	1.000
Les Mis	77	254	2.2×10^5	2	8	1.000
Polbooks	105	441	5.6×10^5	5	35	1.000
Adjnoun	112	425	6.8×10^5	5	21	1.000
Football	115	613	7.4×10^5	6	73	1.001
Journals	124	5972	9.3×10^5	11	80	1.000
Jazz	198	2742	3.8×10^6	60	81	1.003
SmallW	233	994	6.2×10^6	93	166	1.001
C.El-Neural	297	2148	1.2×10^7	274	350	1.000
USAir97	332	2126	1.8×10^7	471	511	1.041
Netscience	379	914	2.7×10^7	887	1134	1.000
Erdos991	446	1413	4.4×10^7	2574	1954	1.011
C.El-Meta	453	2025	4.6×10^7	2497	1138	1.000
Harvard500	500	2043	6.2×10^7	18769	1427	1.000
Roget	994	3640	4.9×10^8	out of memory	53449	≤ 1.008
SmaGri	1024	4916	5.4×10^8	out of memory	25703	≤ 1.002
Email	1133	5451	7.3×10^8	out of memory	34621	≤ 1.005
Polblogs	1222	16714	9.1×10^8	out of memory	41080	≤ 1.013
Vassar85	3068	119161	1.4×10^{10}	out of memory	155333	≤ 1.165

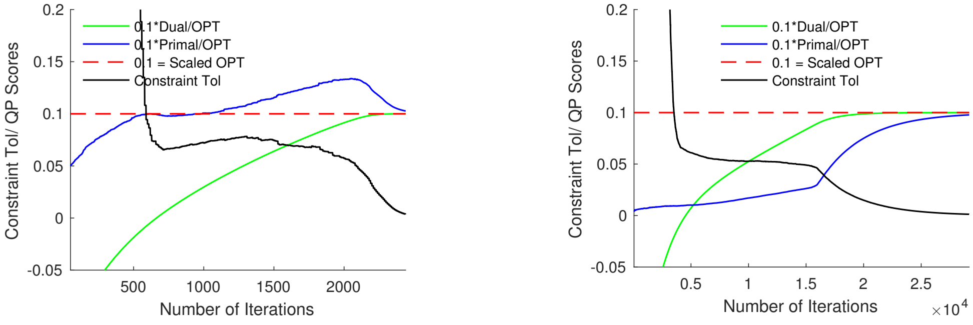
DykstraSC at each iteration both in terms of constraint satisfaction as well as convergence to the optimal objective.

The convergence behavior of the method varies depending on the dataset. As shown in Section 5.3, the dual variables $\mathbf{y}_k$ are always feasible for the dual quadratic function, and thus the green line in the plots always gives a lower bound on the optimal objective. Meanwhile, the primal values $\mathbf{x}_k$ are not necessarily feasible, so we note that $Q(\mathbf{x}_k)$ is not an upper bound for the optimal solution at the start. In some cases (e.g. plots for Jazz, Harvard500, Polblogs, and Vassar85), after the first few iterations the blue line climbs above the red dotted line, indicating that at this point $Q(\mathbf{x}_k)$



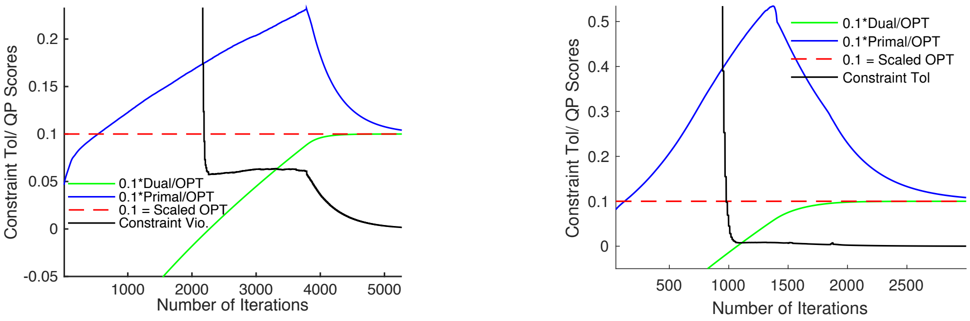
(a) Jazz

(b) Erdos991



(c) Harvard500

(d) Roget



(e) Polblogs

(f) Vassar85

Figure 5.2.: Convergence plots for DykstraSC on several graphs.

becomes an upper bound on the optimal solution. However, in other cases this value in fact stays below the optimal solution until the algorithm converges (e.g. plots for Erdos991 and Roget).

5.7.5 Weighted Correlation Clustering

We convert several real-world graphs into instances of correlation clustering using the approach of Wang et al. [2013]. The procedure is as follows:

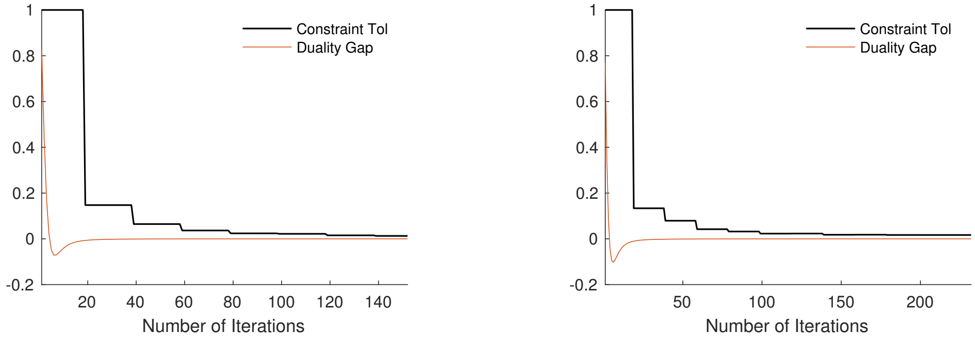
1. Given $G = (V, E)$, compute the Jaccard coefficient between each pair of nodes i, j: $J_{ij} = |N(i) \cap N(j)|/|N(i) \cup N(j)|$ where $N(u)$ is the set of nodes adjacent to node u.
2. Apply a non-linear function on Jaccard coefficients to obtain a score indicating similarity or dissimilarity: $S_{ij} = \log\left((1 + J_{ij} - \delta)/(1 - J_{ij} + \delta)\right)$. Here, δ is set so that $S_{ij} > 0$ if $J_{ij} > \delta$ and $S_{ij} < 0$ when $J_{ij} < \delta$. Following Wang et al. [2013], we fix $\delta = 0.05$.
3. Wang et al. stop after the above step and use S_{ij} scores for their correlation clustering problems. We additionally offset each entry by $\pm\epsilon$ to avoid cases where edge weights are zero:

$$Z_{ij} = \begin{cases} S_{ij} + \epsilon & \text{if } S_{ij} > 0 \\ S_{ij} - \epsilon & \text{if } S_{ij} < 0 \\ \epsilon & \text{if } S_{ij} = 0 \text{ and } (i,j) \in E \\ -\epsilon & \text{if } S_{ij} = 0 \text{ and } (i,j) \notin E. \end{cases}$$

In the above construction, $S_{ij} = 0$ indicates there is no strong similarity or dissimilarity between nodes based on their Jaccard coefficient. If in this case nodes i and j are adjacent, we interpret this as a small indication of similarity and assign them a small positive weight. Otherwise we assign a small negative weight. In all our experiments we fix $\epsilon = 0.01$. The sign of Z_{ij} indicates whether nodes i and j are similar or dissimilar, and $w_{ij} = |Z_{ij}| > 0$ is the non-negative weight for the associated correlation clustering problem. Results for running DykstraCC and Gurobi on the resulting signed graphs are shown in Table 5.2. We show convergence plots in Figure 5.3. For the first three plots in Figure 5.3, notice that for the first several iterations the duality gap $(Q(\mathbf{x}) - D(\mathbf{y}))/D(\mathbf{y})$ is negative. This is because Dykstra's method does not guarantee the primal scores $Q(\mathbf{x})$ will be greater than dual scores $D(\mathbf{y})$ at the beginning of the algorithm. After a few iterations, the maximum constraint violation (black) decreases significantly and the duality gap steadily goes to zero. In Figure 5.3d, the convergence plot for caHepPh, we zoom in on the last couple hundred iterations of the method until convergence falls below 0.01. In practice our solver re-computes the maximum constraint violation every 20 iterations, leading to jumps in the constraint violation curves displayed.

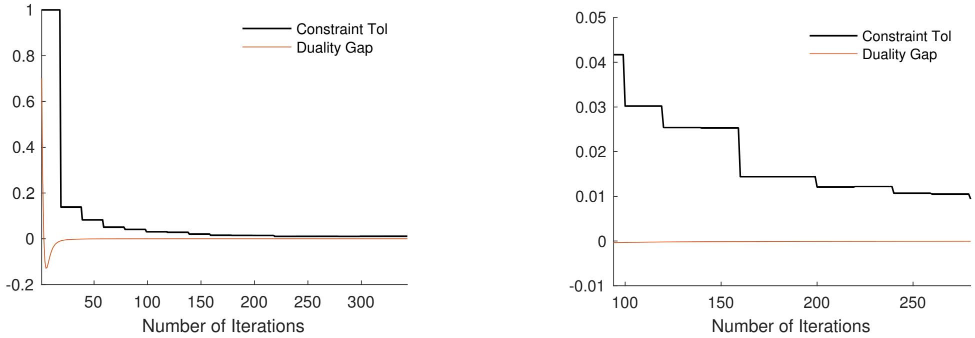
On problems of this size, we restrict to using the lazy-constraint approach, coupled with Gurobi's interior-point solver. In one case, the lazy-constraint method converges very quickly. Effectively, it finds a small subset of constraints that are sufficient to force all other metric constraints to be satisfied at optimality. However, Gurobi runs out of memory on the other large problems considered, indicating that, even if we are extremely careful, standard off-the-shelf solvers are unable to compete with our Dykstra-based approach.

Because the correlation clustering problems we address are so large, we set $\gamma = 1$ and run Dykstra's method until constraints are satisfied to within a tolerance of 0.01. We find that long



(a) Power $n = 4941$

(b) caGrQc $n = 4158$



(c) caHepTh$n = 8638$

(d) caHepPh $n = 11204$

Figure 5.3.: Convergence plots for DykstraCC on four graphs.

before the constraint tolerance reaches this point, the duality gap shrinks below 10^{-4}. We note that although it takes a long time to reach convergence on graphs with thousands of nodes, DykstraCC has no issues with memory. Monitoring the memory usage of our machine, we noted that for the 11,204 node graph, DykstraCC was using only around 12GB of the 100GB of available RAM. Given enough time, therefore, we expect our method to be able to solve metric constrained LPs on a much larger scale. The ability to solve these relaxations on problems of this scale is already an accomplishment, given the fact that standard optimization software often fails on graphs with even a few hundred nodes.

5.7.6 Maximum Modularity Clustering via LP Rounding

For our last experiment we use DykstraCC to obtain approximations to the popular maximum modularity graph clustering objective [Newman and Girvan, 2004]. Although modularity is NP-hard to approximate to within any constant factor [Dinh et al., 2015], solving its LP relaxation allows practitioners to obtain good a posteriori guarantees for fast heuristics such as the celebrated

Table 5.2.: DykstraCC solves convex relaxations of correlation clustering with up to 700 billion constraints. The lazy-constraint Gurobi solver does very well for one very sparse graph, but runs out of memory on all other problems. We set $\gamma = 1$, and constraint tolerance to 0.01. Selecting a small γ leads to poorer approximation guarantees, but dramatically decreases the number of needed iterations until convergence. For problems on which we cannot optimally solve the LP with Gurobi to obtain an approximation ratio Δ, we report an upper bound.

Graph	$\lvert V \rvert$	$\lvert E \rvert$	# constraints	Gurobi Time	Dykstra Time	Δ Approx
power	4941	6594	6.0×10^{10}	549 s	7.6 hrs	1.07
caGrQc	4158	13422	3.6×10^{10}	out of memory	6.6 hrs	≤ 1.33
caHepTh	8638	24806	3.2×10^{11}	out of memory	88.3 hrs	≤ 1.34
caHepPh	11204	117619	7.0×10^{11}	out of memory	173.7 hrs	≤ 1.27

Louvain method [Blondel et al., 2008]. With our approach we solve relaxations on graphs that are roughly an order of magnitude larger than previous approaches [Aloise et al., 2010, Agarwal and Kempe, 2008] and can quickly obtain good bounds on modularity for smaller graphs. Additionally, we demonstrate that rounding the LP and then greedily improving the output with the Louvain algorithm often leads to clusterings with higher modularity than running Louvain by itself.

Modularity Objective Recall that the maximum modularity objective for a graph $G = (V, E)$ (when the Chung-Lu null model is used) is

$$\max_{\mathcal{C}} \ \mathbf{mod}(\mathcal{C}) = \frac{1}{2|E|} \sum_{i,j} \left(A_{ij} - \frac{d_i d_j}{2|E|} \right) \delta_{ij}^{\mathcal{C}} , \tag{5.42}$$

where d_i is the degree of node i, and A_{ij} is the $\{0,1\}$ indicator for whether i, j are adjacent in G. The $\delta_{ij}^{\mathcal{C}}$ variables encode the clustering, i.e., $\delta_{ij}^{\mathcal{C}} = 1$ if i, j are together in $\mathcal{C}$ and $\delta_{ij}^{\mathcal{C}} = 0$ otherwise. As shown in Chapter 3, modularity is a linear function of degree-weighted LambdaCC when we set $\lambda = 1/(2|E|)$. The modularity objective **mod** for a clustering $\mathcal{C}$ and the corresponding correlation clustering objective **DW-LamCC** are then related as follows:

$$\mathbf{DW\text{-}LamCC}(\mathcal{C}) = m(1 - \mathbf{mod}(\mathcal{C})) - \sum_{(i,j) \in E} \frac{d_i d_j}{2m} + \sum_{i=1}^{n} \frac{d_i^2}{4m} , \tag{5.43}$$

where $m = |E|$ and $n = |V|$. We note that this relationship also holds for relaxed clusterings, i.e., we can replace $\delta_{ij}^{\mathcal{C}}$ in (5.42) with $(1 - x_{ij})$ where x_{ij} are relaxed distance variables satisfying metric constraints. One way to upper bound the maximum value of $\mathbf{mod}(\mathcal{C})$ is to first approximately minimize the related instance of correlation clustering using our projection method, and then compute approximation bounds for the correlation clustering objective using guarantees in Sections 5.5 and 5.6. Let $\mathcal{C}^*$ be the optimal clustering for modularity and correlation clustering and $\tilde{\mathcal{C}}$ represent the *relaxed* clustering output by DykstraCC. If we know $\textbf{DW-LamCC}(\tilde{\mathcal{C}}) \leq (1+\delta)\textbf{LamCC}(\mathcal{C}^*)$ for some $\delta > 0$, then using the relationship in (5.43), we can upper bound the maximum modularity:

$$\textbf{mod}(\mathcal{C}^*) \leq \frac{\textbf{mod}(\tilde{\mathcal{C}})}{1+\delta} + \frac{\delta}{1+\delta}\left(1 - \sum_{(i,j)\in E} \frac{d_i d_j}{2m^2} + \sum_{i=1}^{n} \frac{d_i^2}{(2m)^2}\right). \tag{5.44}$$

Modularity scores range between -1 and 1 for any fixed clustering, and for sufficiently small δ the additive approximation above will provide a good upper bound.

We run DykstraCC on a subset of the larger graphs from the first experiment. The weighted correlation clustering problem we are solving here is an instance of LambdaCC with a small resolution parameter $\lambda = 1/(2|E|)$. Recall that for small resolution parameters, LambdaCC is closely related to the sparsest cut and normalized cut objectives. In practice we are not surprised to find therefore that applying our solver to this problem is more similar to our experiments on sparsest cut than the weighted correlation clustering problems in the previous section. We find that many triangle constraints are tight and there is a significant memory requirement even for problems with just a few thousand nodes. Nevertheless, our approach scales to problems an order of magnitude larger than previous results.

LP Rounding and Louvain We compute the LP relaxation to within a constraint tolerance of 10^{-3} and a duality gap of 10^{-4}. We then round the (x_{ij}) variables into clusterings using the pivoting technique ThreeLP given in Algorithm 5 in Chapter 4. Recall that this method repeatedly selects a uniform random node, clusters it with all its neighbors within LP-distance 1/3, then removes the cluster and recurses on the rest of the graph. The method provides no a priori guarantees for $\textbf{mod}(\mathcal{C}^*)$ or $\textbf{DW-LamCC}(\mathcal{C}^*)$, but is a natural approach to test given the provable approximation guarantees we showed for certain parameter regimes. The rounding scheme is very fast, so we take the best of 50 instantiations each time we run it.

The Louvain method is a very popular heuristic developed by Blondel et al. [2008] for maximizing modularity. The method takes an input clustering and repeatedly performs agglomerative moves that greedily improve the objective. We test Louvain in two ways: we first apply the method on the input clustering in which each node belongs to its own cluster, which is the standard initialization for Louvain.

We then run Louvain method Blondel et al. [2008] as a way to greedily improve and refine the clustering output by the LP rounding procedure. As in experimental sections of other chapters, we use the Louvain software of Jeub et al. [2011-2017], which includes randomized variations that can lead to higher-modularity outputs. In Table 5.3 we report the best modularity and median

Table 5.3.: We approximate the LP relaxation of modularity using our projection methods with $\gamma = 2$. Runtime for solving the relaxation is given in column 3. (UB) is an upper bound on the maximum modularity, computed using (5.44). We obtain clusterings with the Louvain method (Louv), a simple LP rounding technique (3LP), and by using the output of 3LP as input to Louvain. We report maximum and median modularity scores over 15 trials. Shown in bold are median and maximum scores that are higher than those achieved with only Louvain.

Graph	n	Time	UB	Louv		3LP		3LP+Lou	
				Max	Med	Max	Med	Max	Med
Netscience	379	56s	0.8652	0.8484	0.8417	0.8310	0.8276	**0.8486**	**0.8485**
C.El-Meta	453	146s	0.5479	0.4478	0.4421	0.2640	0.2479	**0.4517**	**0.4485**
Erdos991	446	167s	0.6602	0.5374	0.5293	0.3498	0.3441	**0.5391**	**0.5369**
Harvard500	500	167s	0.7630	0.7386	0.7349	0.6868	0.6817	0.7386	**0.7386**
Roget	994	2189s	0.7304	0.5438	0.5383	0.2938	0.2888	**0.5472**	**0.5434**
SmaGri	1024	1897s	0.6124	0.4755	0.4697	0.2137	0.2066	**0.4773**	**0.4757**
Email	1133	3203s	0.6880	0.5788	0.5766	0.3356	0.3240	**0.5811**	**0.5789**
Polblogs	1222	3638s	0.5170	0.4270	0.4268	0.1658	0.1300	0.4270	**0.4270**
Vassar85	3068	48.2hr	0.5641	0.3958	0.3957	0.0950	0.0940	0.3958	0.3957

modularity returned over 15 runs for each approach. We observe that LP rounding on its own is not competitive with Louvain. This is not surprising given that the rounding scheme performs simplistic clustering moves that are easy to analyze, but are less sophisticated and intelligent than the Louvain heuristics. However, we notice that combining LP rounding with the Louvain method leads to a more robust approach with higher median and maximum scores. While the improvement is only slight, it is consistent. This indicates that solving the LP relaxation is useful not only for providing bounds on NP-hard objectives, but can also be used as a guide for making heuristic algorithms more robust. This suggests future work in developing LP rounding techniques that are specifically designed to initialize greedy local heuristics like Louvain using global knowledge about the problem instance.

6. LEARNING GRAPH CLUSTERING RESOLUTION PARAMETERS

6.1 Chapter Overview

The value of a resolution parameter λ controls the size and structure of clusters formed by optimizing the LambdaCC objective. For cases where the "right" choice of λ is known in advance, one can apply LP rounding techniques (such as ThreeLP) or heuristic algorithms (such as the Louvain method) in order to approximately solve the problem in practice. However, it many situations it may not be clear a priori what the best value of λ is to use for a given application. In this chapter we therefore present a novel strategy for learning resolution parameters for graph clustering. This strategy can be viewed as an optimization framework in which we optimize over all possible resolution parameters to find one that fits "best" with a certain application.

We approach the task of learning resolution parameters by addressing an *inverted* form of a previously covered theoretical question. In Chapter 4 we asked: how can we find a clustering that optimizes LambdaCC for a given λ? Here we instead assume that an example clustering is *given*, and we seek the parameter λ for which this clustering best approximates the LambdaCC objective. To accomplish this we introduce the notion of a *parameter fitness function*, which measures how well a fixed example clustering optimizes a generalized objective (such as LambdaCC) in different parameter regimes. We prove that under certain reasonable assumptions, this function can be minimized to within arbitrary precision using a bisection-like algorithm. Minimizing this function tells us how well the clustering optimizes a generalized objective in the best case, and returns a good resolution parameter to use in order to find other clusterings like it. Although the results we develop are strongly motivated by the LambdaCC framework, we develop a general theoretical framework that can be applied to a broader class of clustering objective functions with tunable resolution parameters.

There are many reasons one might wish to learn a good resolution parameter from a fixed example clustering. Our primary motivation is that practitioners often will not known which among many mathematical objective functions to apply when faced with a new problem. However, typically they will have some idea of what clustering structure *should* look like for their application. In many cases they will be able to provide at least one example of a clustering that embodies the right notion of community structure in their application domain. In our experimental section, we consider other applications of learning resolution parameters as well. The techniques we develop here will in some cases allow us to learn good resolution parameters when we know limited information about a clustering, but do not know the clustering itself. In this case, the limited information can be viewed as semi-supervised information, and learning a parameter from it will allow us to better uncover the hidden cluster. Another application we will consider extensively is how to use our parameter fitness function to measure how well (or how poorly) different node sets in a real-world network embody community structure at *any* resolution. In particular we use this to measure the relationship between community structure and metadata attributes in social networks.

The results in this chapter were first presented in a conference paper at the 2019 International World Wide Web conference [Veldt et al., 2019b].

6.2 Generalized Clustering Objectives

Many families of objective functions for graph clustering rely on a tunable resolution parameter. We will broadly refer to these as *generalized clustering objectives*. Although LambdaCC is the primary example considered in this chapter, other examples include the Hamiltonian objective studied by Reichardt and Bornholdt [2006], the stability objective of Delvenne et al. [2010], and a multi-resolution variant of the map equation [Schaub et al., 2012]. In the past, solving generalized objective functions for a range of resolution parameters has been used as a way to detect hierarchical clusterings in a network [Reichardt and Bornholdt, 2006]. For example, Jeub et al. [2018] introduced a technique for sampling values of a resolution parameter and applying hierarchical consensus clustering techniques. Our interest in generalized clustering objectives differs in a number of key ways from these results. Instead of seeking hierarchical clustering structure, in this chapter we are interested in learning how to identify a single parameter regime that matches a specific type of community structure. We assume that to begin with, we have access to a single example clustering of a network, which embodies the right notion of community structure for a given application. The task is then to find the resolution parameter that could have produced this clustering (or something similar to it) if we were to optimize a generalized objective function.

6.2.1 Local Graph Clustering

LambdaCC and the other generalized clustering objectives listed above focus on global clustering, in which the goal is to partition an entire network into multiple disjoint clusters. Before moving on to theoretical results, we will also consider a class of clustering objectives that are designed for finding a single local cluster in a specific region of a large graph. For local clustering applications, we will consider an input graph $G = (V, E)$, and additionally a set of seed or *reference* nodes R

around which we wish to form a good community. In Section 2.4.1, we noted that one good way to measure the clustering structure of a node set S is by computing its conductance:

$$\mathbf{cond}(S) = \frac{\text{cut}(S)}{\min\{\text{vol}(S), \text{vol}(\bar{S})\}}. \tag{6.1}$$

This is small when S is connected very well internally but shares few edges with $\bar{S}$. Several graph clustering algorithms have been designed to minimize local variants of (6.1). These substitute the denominator of (6.1) with a measure of the overlap between an output cluster S and the reference set R. One such objective is the following local conductance measure:

$$\phi_R(S) = \begin{cases} \frac{\text{cut}(S)}{\text{vol}(R\cap S) - \varepsilon\,\text{vol}(\bar{R}\cap S)} & \text{if } \text{vol}(R\cap S) - \varepsilon\,\text{vol}(\bar{R}\cap S) > 0 \\ \infty & \text{otherwise.} \end{cases} \tag{6.2}$$

This objective includes a *locality* parameter ε that controls how much overlap there should be between the seed set and output cluster. Algorithms minimizing variants of (6.2) include FlowImprove [Andersen and Lang, 2008], LocalImprove [Orecchia and Zhu, 2014] and SimpleLocal [Veldt et al., 2016]. FlowImprove always uses parameter $\varepsilon = \text{vol}(R)/\text{vol}(\bar{R})$, whereas the latter two approaches choose larger values of ε in order to keep computations more local. In the extreme case where $\varepsilon = \infty$, the problem reduces to finding the minimum conductance *subset* of a reference set R, which can be accomplished by the Minimum Quotient Improvement (MQI) algorithm of Lang and Rao [2004].

Objective (6.2) can be efficiently minimized by repeatedly solving a minimum s-t cut problem on an auxiliary graph constructed from G, which introduces a sink node s attached to nodes in R, and a source node t attached to nodes in $\bar{R} = V\backslash R$. Edges are weighted with respect to the locality parameter ε and another parameter α. In order to detect whether there exists some set S with $\phi_R(S) \leq \alpha$, one can solve a local clustering objective corresponding to the minimum s-t cut objective on the auxiliary graph. We refer to this simply as the *local flow clustering objective*:

$$\min\ f_\alpha(S) = \text{cut}(S) + \alpha\,\text{vol}(R\cap\bar{S}) + \alpha\varepsilon\,\text{vol}(\bar{R}\cap S). \tag{6.3}$$

If the set S minimizing f_α satisfies $f_\alpha(S) < \alpha\,\text{vol}(R)$, then rearranging terms one can show that $\phi_R(S) < \alpha$. Thus, by performing binary search over α or repeatedly solving (6.3) for smaller and smaller α, one can minimize the local conductance measure (6.2).

Previous research has largely treated α as a temporary parameter used in one step of a larger algorithm seeking to minimize (6.2). Algorithms which minimize (6.2) do so by finding the smallest α such that the minimum of (6.3) is $\alpha\,\text{vol}(R)$. We depart from this approach by instead treating α as a tunable resolution parameter for balancing two conflicting goals: finding clusters with a small cut, and finding clusters that have a large overlap with the seed set R. In the case where ε is treated as infinitely large and we are simply looking for subsets of a seed set R satisfying $\operatorname{vol}(R) \leq \operatorname{vol}(\bar{R})$, then in effect we are trying to solve the optimization problem:

$$\min \operatorname{cut}(S) - \alpha \operatorname{vol}(S) + \alpha \operatorname{vol}(R) \text{ such that } S \subseteq R. \tag{6.4}$$

This goal is related to, but ultimately should be contrasted with, the goal of minimizing the ratio $\operatorname{cut}(S)/\operatorname{vol}(S)$. The objectives are similar in that they both tend to prefer sets with small cut and large volume. We argue that treating α as a tunable parameter is in fact more versatile than simply minimizing the ratio score. In multiple applications it may be useful to find clusters with small cut and large volume, but different applications may put a different weight on each aspect of the objective. We observe that ε also plays an important role in the size and structure of the output community when it is less than ∞. For simplicity, we will treat this as a fixed constant, and in our experimental section we simply focus on objective (6.4).

6.2.2 Parametric Linear Programs

Before moving on we provide key background on parametric linear programming which will be important in our theoretical results. A standard linear program is a problem of the form

$$\min_{\mathbf{x}} \ \mathbf{c}^T\mathbf{x} \text{ such that } \boldsymbol{A}\mathbf{x} \leq \mathbf{b} \tag{6.5}$$

where $\mathbf{c}, \mathbf{b}$ are vectors and $\mathbf{A}$ is a constraint matrix. A *parametric* linear program is a related problem of the form

$$\min_{\mathbf{x}} \ \mathbf{c}^T\mathbf{x} + \beta(\Delta\mathbf{c})^T\mathbf{x} \text{ such that } \boldsymbol{A}\mathbf{x} \leq \mathbf{b} \tag{6.6}$$

where $\Delta\mathbf{c}$ is another vector of the same length as $\mathbf{c}$ and β is a parameter controlling the difference between (6.5) and (6.6). We state a well-known result about the solutions of (6.6) for different β. This result is not new; it follows directly from Proposition 2.3b from Adler and Monteiro [1992].

Theorem 6.2.1 *Let $L(\beta)$ be the minimum of (6.6) for a fixed β. If we are given bounds a and b such that $L(\beta) \in \mathbb{R}$ for all $\beta \in [a, b]$, then L is a piecewise linear and concave function in β over this interval.*

Parametric LPs in Graph Clustering Applications In our work it is significant to note that the linear programming relaxation of LambdaCC is a parametric linear program in λ. Furthermore, the local flow clustering objective can be cast as a parametric linear program in α, since this objective corresponds simply to a special case of the minimum s-t cut problem, which can be cast as an LP.

6.3 Theoretical Results

The major theoretical contribution of this chapter is a new framework for learning clustering resolution parameters, based on minimizing a parameter fitness function. We present results for a generic clustering objective and fitness function, and later show how to apply our results to LambdaCC and local flow clustering.

6.3.1 Problem Formulation

Let $\mathscr{C}$ denote a set of valid clusterings for a graph $G = (V, E)$. We consider a generic clustering objective function $f_\beta : \mathscr{C} \rightarrow \mathbb{R}_{\geq 0}$ that depends on a resolution parameter β. The function takes as input a clustering $\mathcal{C} \in \mathscr{C}$, and outputs a nonnegative clustering quality score for $\mathcal{C}$. We assume that smaller values of f_β are better. We intentionally allow f_β to be very general in order to develop broadly applicable theory. For intuition, one can think of f_β as being the LambdaCC function (4.26) with $\beta = \lambda$. Alternatively, one can picture f_β to be the local flow objective (6.3) with $\beta = \alpha$ and with $\mathscr{C}$ representing the set of bipartitions, i.e. for any $\mathcal{C} \in \mathscr{C}$, $\mathcal{C} = \{S, \bar{S}\}$ for some set $S \subset V$.

Given some objective function f_β, a standard clustering paradigm is to assume that an appropriate value of β has already been chosen, and then the goal is to produce some clustering $\mathcal{C}$ that exactly or approximately minimizes f_β. Here, we address an inverse question: given an example clustering $\mathcal{C}_x$, how do we determine a parameter β such that $\mathcal{C}_x$ approximately minimizes f_β? Ideally we would like to solve the following problem:

$$\textbf{Goal 1:} \quad \text{Find } \beta > 0 \text{ such that } f_\beta(\mathcal{C}_x) \leq f_\beta(\mathcal{C}) \text{ for all } \mathcal{C} \in \mathscr{C}. \tag{6.7}$$

In practice, however, $\mathcal{C}_x$ may not exactly minimize a generic clustering objective for any choice of resolution parameter. Thus we relax this to a more general and useful goal:

$$\textbf{Goal 2:} \quad \text{Find the minimum } \Delta \geq 1 \text{ such that for some } \beta > 0$$

$$f_\beta(\mathcal{C}_x) \leq \Delta f_\beta(\mathcal{C}) \text{ for all } \mathcal{C} \in \mathscr{C}. \tag{6.8}$$

This second goal is motivated by the study of approximation algorithms for clustering. In effect this asks: if we are given a certain clustering $\mathcal{C}_x$, is $\mathcal{C}_x$ a good approximation to f_β for any choice of β? Note that this generalizes (6.7): if β can be chosen to satisfy Goal 1, then the same β will satisfy Goal 2 with $\Delta = 1$. Furthermore, it has the added advantage that, if solved, Goal 2 will produce a value Δ which communicates how well clusterings like $\mathcal{C}_x$ can be detected using variants of the objective function f_β. If Δ is near 1, it means that f_β is able to produce similar clusterings for a correct choice of β, whereas if Δ is very large this indicates that $\mathcal{C}_x$ will be difficult to find even for an optimal β, and thus a different approach will be necessary for detecting clusterings of this type.

Clustering Relaxations While Goal 2 is a more reasonable target than Goal 1, it may still be a very challenging problem to solve when objective f_β is hard to optimize, e.g., if it is NP-hard. We thus consider one final relaxation that is slightly weaker than (6.8), but will be more feasible to work with. Let $\hat{\mathscr{C}}$ denote a superset of $\mathscr{C}$ which includes not only clusterings for G, but also some notion of a relaxed clustering, and let $g_\beta : \hat{\mathscr{C}} \rightarrow \mathbb{R}_{\geq 0}$ be an objective that assigns a score for every $\mathcal{C} \in \hat{\mathscr{C}}$. Furthermore, assume g_β represents a lower bound function for f_β: $g_\beta(\mathcal{C}) \leq f_\beta(\mathcal{C})$ for all β and all $\mathcal{C} \in \mathscr{C}$.

Our consideration of g_β is motivated by the fact that many NP-hard clustering objectives permit convex relaxations, which can be optimized in polynomial time over a larger set of relaxed clusterings that contain all valid clusterings of G as a subset. This is exactly the case for the LambdaCC objective, which is NP-hard to optimize in general, but permits a useful linear programming relaxation. Since g_β is indeed easier to optimize than f_β, the following goal will be easier to approach but still provide strong guarantees for learning a good value of β:

$$\textbf{Goal 3:} \text{ Find the minimum } \Delta \geq 1 \text{ such that for some } \beta > 0$$
$$f_\beta(\mathcal{C}_x) \leq \Delta g_\beta(\mathcal{C}) \text{ for all } \mathcal{C} \in \hat{\mathscr{C}}. \tag{6.9}$$

If we can solve (6.9), this still guarantees that $\mathcal{C}_x$ is a Δ-approximation to f_β for an appropriately chosen β. For problems where f_β is very challenging to optimize, but g_β is not, this will be a much more feasible approach. In the next section we will focus on developing theory for addressing Goal 3, though we note that in applying this theory we can still choose $g_\beta = f_\beta$ and therefore instead address the stronger Goal 2 whenever this is feasible. In particular we note that the local flow clustering objective (6.3) can be solved in polynomial time, so we can feasibly address Goal 2, or in other words Goal 3 with $g_\beta = f_\beta$.

6.3.2 Parameter Fitness Function

We now present a parameter fitness function whose minimization is equivalent to solving (6.9). Functions f_β and g_β take a clustering or relaxed clustering as input and output an objective score. However, we wish to view β as an input parameter and we treat an example clustering C_x as a fixed input. Thus for convenience we introduce new related functions:

$$F(\beta) = f_\beta(C_x) \tag{6.10}$$

$$G(\beta) = \min_{C \in \hat{\mathcal{C}}} g_\beta(C) \tag{6.11}$$

The ratio of these two functions defines the *parameter fitness function* that we seek to minimize:

$$\mathcal{P}(\beta) = \frac{F(\beta)}{G(\beta)}. \tag{6.12}$$

Observe that this function is always greater than or equal to 1 since $G(\beta) \leq F(\beta)$ for any β. The minimizer of $\mathcal{P}$ is a resolution parameter β that minimizes the ratio between the clustering score of a fixed C_x and a lower bound on f_β. Thus, by minimizing (6.12) we achieve Goal 3 in (6.9) with $\Delta = \min_\beta \mathcal{P}(\beta)$.

In Section 6.2.2, we noted that the local flow clustering objective can be characterized as a parametric linear program, as can the LP relaxation of LambdaCC. Furthermore, for a fixed clustering, both objective functions can be viewed as a linear function in terms of their resolution parameter. Motivated by these facts, we present a theorem which characterizes the behavior of the parameter fitness function $\mathcal{P}$ under certain reasonable conditions on the functions F and G. In the subsequent section we will use this result to show that $\mathcal{P}$ can be minimized to within arbitrary precision using an efficient bisection-like method.

Theorem 6.3.1 *Assume $F(\beta) = a + b\beta$ for nonzero real numbers a and b. Let G be concave and piecewise linear in β, and assume $F(\beta) \geq G(\beta) \geq 0$ for all $\beta \in [\ell, r]$ where ℓ and r are nonnegative lower and upper (i.e. left and right) bounds for β. Then $\mathcal{P}$ satisfies the following two properties:*

(a) If $\beta^- < \beta < \beta^+$, then $\mathcal{P}(\beta)$ cannot be strictly greater than both $\mathcal{P}(\beta^-)$ and $\mathcal{P}(\beta^+)$.

(b) If $\mathcal{P}(\beta^-) = \mathcal{P}(\beta^+)$, then $\mathcal{P}$ achieves its minimum in $[\beta^-, \beta^+]$.

Proof Note that for some $\gamma \in (0, 1)$, $\beta = (1-\gamma)\beta^+ + \gamma\beta^-$. By concavity of G and linearity of F, we know

$$\begin{aligned}\mathcal{P}(\beta) = \frac{F((1-\gamma)\beta^+ + \gamma\beta^-)}{G((1-\gamma)\beta^+ + \gamma\beta^-)} &\leq \frac{(1-\gamma)F(\beta^+) + \gamma F(\beta^-)}{(1-\gamma)G(\beta^+) + \gamma G(\beta^-)} \\ &\leq \max\left\{\frac{(1-\gamma)F(\beta^+)}{(1-\gamma)G(\beta^+)}, \frac{\gamma F(\beta^-)}{\gamma G(\beta^-)}\right\} = \max\left\{\mathcal{P}(\beta^+), \mathcal{P}(\beta^-)\right\},\end{aligned}$$

which proves the first property. Now assume that $\mathcal{P}(\beta^-) = \mathcal{P}(\beta^+)$. Using property (a), we know as β increases from its lower to upper limit, $\mathcal{P}$ cannot increase and then decrease. Thus, either $\mathcal{P}$ attains its minimum on $[\beta^-, \beta^+]$, else $\mathcal{P}$ is a constant for all $\beta \in [\beta^-, \beta^+]$. If the latter is true, then for some $\beta \in [\beta^-, \beta^+]$ and some sufficiently small $\epsilon > 0$, G must be linear in the range $(\beta - \epsilon, \beta + \epsilon)$, since we know that G is piecewise linear. Therefore, $G(\beta) = c + d\beta$ and

$$\mathcal{P}(\beta) = (a + b\beta)/(c + d\beta) = \textit{constant} \tag{6.13}$$

for $\beta \in (\beta - \epsilon, \beta + \epsilon)$ and for some $c, d \in \mathbb{R}$. This ratio of linear functions can only be a constant if $a = c = 0$, or $b = d = 0$, or if $a = c$ and $b = d$. Since we assumed a and b were nonzero, the last case must hold, and thus $\mathcal{P}(\beta) = 1$ for every $\beta \in [\beta^-, \beta^+]$, so the minimizer is obtained in this case, since $\mathcal{P}(\beta) \geq 1$ for all β. ■

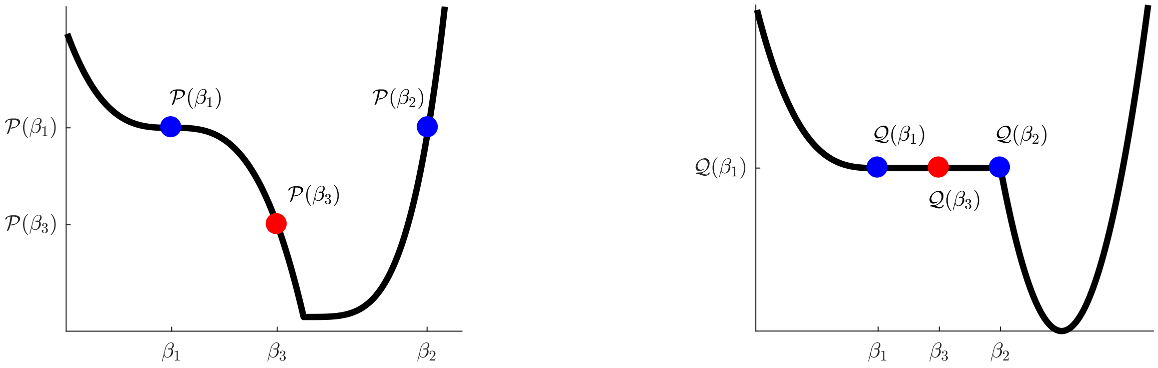


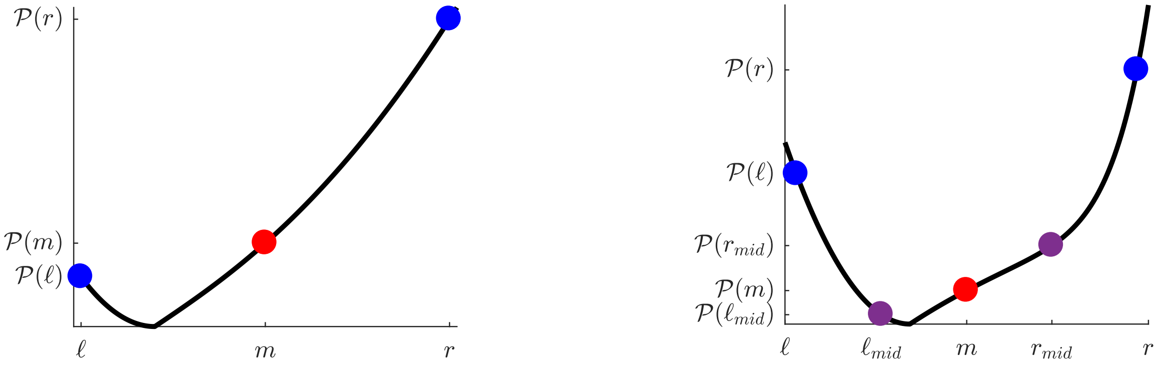
Figure 6.1.: Left: Function $\mathcal{P}$ satisfies both properties (a) and (b) in Theorem 6.3.1. If $\mathcal{P}(\beta_1) = \mathcal{P}(\beta_2)$, querying $\mathcal{P}$ at any point $\beta_3 \in [\beta_1, \beta_2]$ gets us closer to a minimizer. Right: Function $\mathcal{Q}$ satisfies property (a) but not property (b). If $\mathcal{Q}(\beta_1) = \mathcal{Q}(\beta_2)$, we can get stuck evaluating $\mathcal{Q}$ in a flat region not near a minimizer.

In the next section we present a method for finding the minimizer of a function satisfying properties (a) and (b) in Theorem 6.3.1 to within arbitrary precision. Before doing so, we highlight the importance of ensuring that *both* properties hold. In Figure 6.1 we plot two toy functions, $\mathcal{P}$ and $\mathcal{Q}$. Although both satisfy property (a), only $\mathcal{P}$ additionally satisfies (b). Assume we do not have explicit representations of either function, but we can query them at specific points to help find their minimizers.

Consider Figure 6.1. If we query $\mathcal{P}$ at points β_1 and β_2 to find that $\mathcal{P}(\beta_1) = \mathcal{P}(\beta_2)$, then choosing any third point $\beta_3 \in (\beta_1, \beta_2)$ will get us closer to the minimizer. However, if $\mathcal{Q}(\beta_1) = \mathcal{Q}(\beta_2)$ for some β_1, β_2, we cannot guarantee these points are not part of a flat region of $\mathcal{Q}$ somewhere far from the minimizer. It thus becomes unclear how to choose a third point β_3 at which to query $\mathcal{Q}$. If we choose some $\beta_3 \in (\beta_1, \beta_2)$ and find that $\mathcal{Q}(\beta_3) = \mathcal{Q}(\beta_2) = \mathcal{Q}(\beta_1)$, the minimizer may be within $[\beta_1, \beta_2]$, within $[\beta_2, \beta_3]$, or in a completely different region. Thus it is important for the denominator of a parameter fitness function to be piecewise linear in addition to being concave, since this piecewise linear assumption guarantees property (b) will hold.

6.3.3 Minimizing the Parameter Fitness Function

We now outline an approach for finding a minimizer of $\mathcal{P}$ to within arbitrary precision when Theorem 6.3.1 holds. Our approach is closely related to the standard bisection method for finding zeros of a continuous function f. Recall that standard bisection starts with a and b such that $sign(f(a)) \neq sign(f(b))$, and then computes $f(c)$ where $c = (a+b)/2$. Checking the sign of $f(c)$



(a) One-branch phase

(b) Two-branch phase

Figure 6.2.: We evaluate $\mathcal{P}$ at left and right bounds (blue points), and at a midpoint m (red point). Left: If $\mathcal{P}(\ell) < \mathcal{P}(m) < \mathcal{P}(r)$, then we know the minimizer of $\mathcal{P}$ is in $[\ell, m]$, and we recursively call the *one-branch* phase (Algorithm 1) with new bounds ℓ and m. Right: If $\mathcal{P}(m) < \mathcal{P}(\ell) < \mathcal{P}(r)$, we don't know if the minimizer is in the left branch $[\ell, m]$ or right branch $[m, r]$. Evaluating $\mathcal{P}$ at the midpoint of each branch (purple points), we rule out branch $[m, r]$ and recursively call Algorithm 2 with new endpoints ℓ and m and midpoint ℓ_{mid}.

allows one to determine whether the zero of f is located within the interval $[a, c]$ or $[b, c]$. Thus each new query of the function f halves the interval in which a zero must be located.

Two-Branch Bisection for Minimizing $\mathcal{P}$ Assume $\mathcal{P}$ satisfies properties (a) and (b) in Theorem 6.3.1 over an interval $[\ell, r]$. To satisfy Goal 3, given in (6.9) in Section 6.3.1, it suffices to find any minimizer of $\mathcal{P}$, which we do by repeatedly halving the interval in which the minimizers of $\mathcal{P}$ must lie. Our approach differs from standard bisection in that we are trying to find a *minimizer* instead of the zero of some function. The key algorithmic difference is that querying $\mathcal{P}$ at a single point between two bounds will not always be sufficient to cut the search space in half.

Figure 6.2 illustrates a step of our bisection-like method in two different situations. In general, our method always starts in a *one-branch* phase in which we know a minimizer lies between ℓ and r. If we compute $m = (\ell + r)/2$ and find that $\mathcal{P}(m)$ is between $\mathcal{P}(\ell)$ and $\mathcal{P}(r)$, this does in fact automatically cut our search space in half, as this implies that $\mathcal{P}$ is monotonic on either $[l, m]$ or $[m, r]$ (see Figure 6.2a). However, if $\mathcal{P}(m) < \min\{\mathcal{P}(\ell), \mathcal{P}(r)\}$, then it is possible for the minimizer to reside within either the left branch $[\ell, m]$ or the right branch $[m, r]$ (see Figure 6.2b). In this case, the method enters a *two-branch* phase in which it takes the midpoint of each branch ($\ell_{mid} = (\ell + m)/2$ and $r_{mid} = (m + r)/2$) and evaluates $\mathcal{P}(\ell_{mid})$ and $\mathcal{P}(r_{mid})$. If $\mathcal{P}$ returns the same value for two of the inputs (e.g., $\mathcal{P}(\ell) = \mathcal{P}(m)$), then by property (b) we have found a new interval containing the

Algorithm 12 CheckOneBranch(ℓ, r, ϵ)

```
Base case:
if $r - \ell < \epsilon$ then
    return $\ell$
Recursive call:
Midpoint: $m = (\ell + r)/2$
switch $\ell, m, r$ do
    case $\mathcal{P}(\ell) = \mathcal{P}(m) = \mathcal{P}(r)$
        return $m$
    case $\mathcal{P}(\ell) \leq \mathcal{P}(m) < \mathcal{P}(r)$
        return CheckOneBranch($\ell, m, \epsilon$)
    case $\mathcal{P}(\ell) > \mathcal{P}(m) \geq \mathcal{P}(r)$
        return CheckOneBranch($m, r, \epsilon$)
    case $\mathcal{P}(\ell) > \mathcal{P}(m) < \mathcal{P}(r)$
        return CheckTwoBranches($\ell, m, r, \epsilon$)
```

minimizer(s) of $\mathcal{P}$ that is at most half the length of $[\ell, r]$. Otherwise, we can use property (a) to deduce that the minimizer will be located within $[\ell, m]$, $[m, r]$, or $[\ell_{mid}, r_{mid}]$, and we recurse on the two-branch phase.

Algorithms 12 and 13 handle the one- and two-branch phases of the method respectively. The guarantees of our method are summarized in Theorem 6.3.2. We omit the full proof, since it follows directly from considering different simple cases and applying properties of $\mathcal{P}$ to halve the search space as outline above.

Theorem 6.3.2 *Consider a fixed clustering C_x and a corresponding parameter fitness function $\mathcal{P}_{C_x}$ satisfying the assumptions of Theorem 6.3.1. Running Algorithm 12 with input ℓ, r and a tolerance ϵ will produce a resolution parameter $\tilde{\beta}$ that is within ϵ of the minimizer of $\mathcal{P}_X$ over the interval $[\ell, r]$, in at most $\log_2((r - \ell)/\epsilon)$ recursive calls.*

6.4 Application to Specific Objectives

Theorem 6.3.1 and our approach for minimizing $\mathcal{P}$ can be immediately applied to learn resolution parameters for the LambdaCC global clustering objective and the local flow clustering objective.

Algorithm 13 CheckTwoBranches(ℓ, m, r, ϵ)

```
Base case:
if $r - \ell < \epsilon$ then
    return $m$
Recursive call:
Left midpoint: $\ell_{mid} = (\ell + m)/2$
Right midpoint: $r_{mid} = (m + r)/2$
switch $\ell_{mid}, m, r_{mid}$ do
    case $\mathcal{P}(\ell_{mid}) = \mathcal{P}(m) = \mathcal{P}(r_{mid})$
        return $m$
    case $\mathcal{P}(\ell_{mid}) = \mathcal{P}(m) \neq \mathcal{P}(r_{mid})$
        return CheckOneBranch($\ell_{mid}, m, \epsilon$)
    case $\mathcal{P}(\ell_{mid}) \neq \mathcal{P}(m) = \mathcal{P}(r_{mid})$
        return CheckOneBranch($m, r_{mid}, \epsilon$)
    case $\mathcal{P}(\ell_{mid}) < \mathcal{P}(m) < \mathcal{P}(r_{mid})$
        return CheckTwoBranches($\ell, \ell_{mid}, m, \epsilon$)
    case $\mathcal{P}(\ell_{mid}) > \mathcal{P}(m) > \mathcal{P}(r_{mid})$
        return CheckTwoBranches($m, r_{mid}, r, \epsilon$)
    case $\mathcal{P}(\ell_{mid}) > \mathcal{P}(m) < \mathcal{P}(r_{mid})$
        return CheckTwoBranches($\ell_{mid}, m, r_{mid}, \epsilon$)
```

6.4.1 Local Clustering

For local clustering we consider the objective function f_α given in (6.3) and note that the set of valid clusterings $\mathscr{C}$ is the set of bipartitions. The example clustering we are given at the outset of the problem is $\mathcal{C}_x = \{X, \bar{X}\}$ where $X \subset V$ is some nontrivial set of nodes representing a "good" cluster for a given application. We assume we are also given a reference set R (with $\mathrm{vol}(R) \leq \mathrm{vol}(\bar{R})$) that defines a region of the graph in which we are searching for clusters. As noted previously, f_α can be viewed as a parametric linear program, and furthermore it will evaluate to a non-negative finite number for any $\alpha > 0$. Thus by Theorem 6.2.1, $G(\alpha) = \min_S f_\alpha(S)$ is concave and piecewise linear and we can apply Theorem 6.3.1. More explicitly, the local clustering parameter fitness function is

$$\mathcal{P}_X(\alpha) = \frac{\mathrm{cut}(X) + \alpha\,\mathrm{vol}(\bar{X} \cap R) + \alpha\varepsilon\,\mathrm{vol}(X \cap \bar{R})}{\min_S[\mathrm{cut}(S) + \alpha\,\mathrm{vol}(\bar{S} \cap R) + \alpha\varepsilon\,\mathrm{vol}(S \cap \bar{R})]}. \tag{6.14}$$

If we focus on finding clusters that are subsets of R, using objective (6.4), we have a simplified fitness function:

$$\mathcal{P}_X(\alpha) = \frac{\operatorname{cut}(X) - \alpha \operatorname{vol}(X) + \alpha \operatorname{vol}(R)}{\min_{S \subseteq R} [\operatorname{cut}(S) - \alpha \operatorname{vol}(S) + \alpha \operatorname{vol}(R)]}. \tag{6.15}$$

When we apply Algorithm 12 to minimize (6.14) or (6.15), we can query $\mathcal{P}_X$ in the time it takes to evaluate a linear function and the time it takes to solve the s-t cut problem (6.3). This can be done extremely quickly using localized min-cut computations [Lang and Rao, 2004, Orecchia and Zhu, 2014, Veldt et al., 2016, 2019c].

Functions (6.14) and (6.15) should be minimized over $\alpha \in [\alpha^*, \operatorname{cut}(R)]$, where α^* is either the minimum of (6.2) if we are minimizing (6.14), or the minimum conductance for a subset of R if we are minimizing (6.15). One can show that for any α outside this range, objectives (6.3) and (6.4) will be trivially minimized by $S = R$, so it is not meaningful to optimize these objectives for these α. In practice one can additionally set stricter upper and lower bounds if desired.

6.4.2 Global Clustering Approach

We separately consider the standard and degree-weighted versions of LambdaCC when applying Theorem 6.3.1 to global graph clustering.

Standard LambdaCC For the standard objective, it is useful to consider the scaled version of LambdaCC obtained by dividing (4.26) by $1 - \lambda$ and substituting for a new resolution parameter $\gamma = \lambda/(1-\lambda)$. Then the objective is

$$\min \ \textstyle\sum_{(u,v)\in E}(1-\delta_{uv}) + \sum_{(u,v)\notin E} \gamma \delta_{uv}. \tag{6.16}$$

The denominator of the parameter fitness function for this scaled LambdaCC problem would be

$$G(\gamma) = \min_{\mathbf{x}\in\mathcal{X}} \ \textstyle\sum_{(u,v)\in E} x_{uv} + \sum_{(u,v)\notin E} \gamma(1 - x_{uv}) \tag{6.17}$$

where $\mathcal{X}$ represents the set of constraints for the linear programming relaxation of LambdaCC. Note that $G(\gamma)$ will be finite for every $\gamma \geq 0$, so Theorem 6.2.1 holds. Thus G is concave and piecewise linear as required by Theorem 6.3.1. Next, for a fixed clustering $\mathcal{C}_x$, let P_x be the number of positive mistakes (pairs of nodes that are separated despite sharing an edge) and N_x be the number of negative mistakes (pairs of nodes that are clustered together but share no edge). Then objective (6.16) for this clustering is $P_x + \gamma N_x$, and we see that this fits the linear form given in Theorem 6.3.1 as long as the example clustering satisfies $P_x > 0$ and $N_x > 0$, which will be the case for nearly any nontrivial clustering one might consider. Finally, note that the parameter fitness function for (6.16) would be exactly the same as the parameter fitness function for the standard

LambdaCC objective, since scaling by $(1-\lambda)$ makes no difference if we are going to minimize the ratio between the clustering objective and its LP relaxation. The parameter fitness function for standard LambdaCC is therefore

$$\mathcal{P}_{\mathcal{C}_x}(\lambda) = \frac{(1-\lambda)P_x + \lambda N_x}{\min_{\mathbf{x}} \left[\sum_{uv \in E}(1-\lambda)x_{uv} + \sum_{uv \notin E}\lambda(1-x_{uv})\right]} \tag{6.18}$$

$$= \frac{P_x + \gamma N_x}{\min_{\mathbf{x}} \left[\sum_{uv \in E} x_{uv} + \sum_{uv \notin E}\gamma(1-x_{uv})\right]} \tag{6.19}$$

and it satisfies the assumptions of Theorem 6.3.1 as long as $P_x > 0$, $N_x > 0$, and we optimize over $\lambda \in (0,1)$.

Degree-weighted LambdaCC Showing how Theorem 6.3.1 applies to degree-weighted LambdaCC requires slightly more work, though the same basic principles hold. The LP-relaxation of the objective is still a parametric linear program, thus is still concave and piecewise linear in λ over the interval $(0,1)$. The denominator of the parameter fitness function in this case would be:

$$\min \sum_{(u,v)\in E^+} w_{uv}(1-\delta_{uv}) + \sum_{(u,v)\in E^-} w_{uv}\delta_{uv}. \tag{6.20}$$

where w_{uv} is the weight between nodes u and v defined in the degree-weighted LambdaCC fashion (see Section 3.3). For a fixed example clustering $\mathcal{C}_x$ encoded by a function $\delta_x = (\delta_{uv})$, we can rearrange this into the form $a + \lambda b$. Here, a and b depend on the structure of the graph and the value of λ, since positive and negative edges in the construction of the LambdaCC problem will depend on each node's degree (see Section 3.3). In the case where $\lambda \in (0, 1/(d_{max}))$ where d_{max} is the maximum degree of the graph in question, we can be more explicit. In this case the objective is

$$\textbf{DW-LamCC}(\mathcal{C}) = \sum_{(u,v)\in E}(1-\lambda d_u d_v)(1-\delta_{uv}) + \sum_{(u,v)\notin E}\lambda d_u d_u \delta_{uv} \tag{6.21}$$

$$= \sum_{(u,v)\in E}(1-\delta_{uv}) + \lambda\left[\sum_{(u,v)\notin E} d_u d_v \delta_{uv} - \sum_{(u,v)\in E} d_u d_v (1-\delta_{uv})\right] \tag{6.22}$$

so $a = \sum_{(u,v)\in E}(1-\delta_{uv})$ and $b = \sum_{(u,v)\notin E} d_u d_v \delta_{uv} - \sum_{(u,v)\in E} d_u d_v(1-\delta_{uv})$. One the edge weights are determined for a fixed graph, these values are simple to compute, and as long as they are both nonzero, the results of Theorem 6.3.1 apply. In some extreme cases it is possible that $a = 0$ or $b = 0$, but we expect this to be rare. Furthermore, our general approach may still work even when $a = 0$ or $b = 0$, Theorem 6.3.1 simply does not analyze this case. We leave it as future work to develop more refined sufficient and necessary conditions such that Algorithm 12 is guaranteed to minimize the parameter fitness function $\mathcal{P}$.

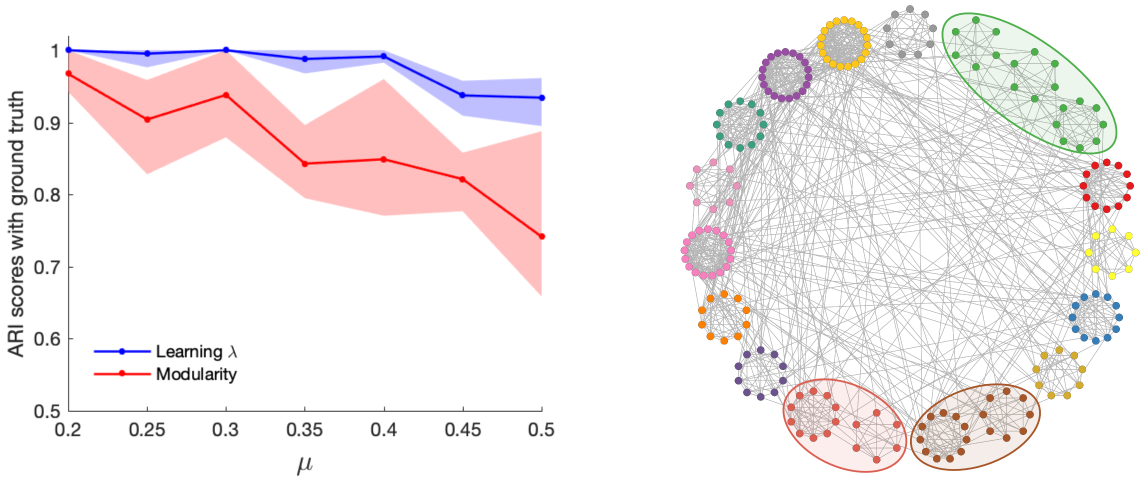


Figure 6.3.: Left: ARI scores for detecting ground truth in LFR graphs. Solid lines indicate median scores, and colored regions show the range of scores across five test graphs for each μ. Right: one of the five LFR test graphs for $\mu = 0.3$. Modularity ($\lambda = 1/(2|E|)$) makes mistakes by putting distinct ground truth clusters together (highlighted). For this example our approach perfectly detects the ground truth.

6.5 Experiments

We consider several local and global clustering experiments in which significant benefit can be gained from learning resolution parameters rather than using previous off-the-shelf algorithms and objective functions. We implement Algorithms 12 and 13 in the Julia programming language for both local and global parameter fitness functions. Computing the LambdaCC linear programming relaxation can be challenging due to the size of the constraint set. For our smaller graphs we apply Gurobi optimization software, and for larger problems we can use the memory-efficient projection methods developed in Chapter 5. We also make use of recent improvements to these methods that perform multiple projection steps in parallel, which allows the projection method to converge more quickly [Ruggles et al., 2019]. For the local-flow objective we use a fast Julia implementation we developed in recent work [Veldt et al., 2019c]. Our experiments were run on a machine with two Intel Xeon E5-2690 v4 processors. Code for our experiments and algorithms are available at `https://github/nveldt/LearnResParams`.

6.5.1 Learning Parameters for Synthetic Datasets

Although modularity is a widely-applied objective function for community detection, Fortunato and Barthélemy [2007] demonstrated that it is unable to accurately detect communities below a certain size threshold in a graph. In our first experiment we demonstrate that learning resolution parameters for LambdaCC allows us to overcome the *resolution limit* of modularity, and better detect community structure in synthetic networks. We generate a large number of synthetic LFR benchmark graphs [Lancichinetti and Fortunato, 2009], in a parameter regime that is chosen to be difficult for modularity. All graphs contain 200 nodes, average degree 10, max degree 20, and community sizes between 5 and 20 nodes. We test a range of mixing parameters μ, which controls the fraction of edges that connect nodes in different communities ($\mu = 0$ means all edges are inside the communities).

For each μ from 0.2 to 0.5, in increments of 0.05, we generate six LFR networks, one for training and five for testing. On the training graph, we minimize the degree-weighted LambdaCC parameter fitness function to learn a resolution parameter λ_{best}. This takes between roughly half an hour (for $\mu = 0.2$) to just over three hours (for $\mu = 0.5$), solving the underlying LambdaCC LP with Gurobi software. We then cluster the five test LFR examples using Lambda-Louvain (see Section 3.5 and Jeub et al. [2011-2017] for implementation). We separately run the method with two resolution parameters: $\lambda = 1/(2|E|)$, the standard setting for modularity, and $\lambda = \lambda_{best}$. Learning λ_{best} significantly improves Adjusted Rand index (ARI) scores for detecting the ground truth (see Figure 6.3).

6.5.2 Local Community Detection

Next we demonstrate that a small amount of semi-supervised information about target communities in real-world networks can allow us to learn good resolution parameters, leading to more robust community identification. Additionally, minimizing the parameter fitness function provides a way to measure the extent to which *functional* communities in a network correspond to topological notions of community structure in networks.

Data We consider four undirected networks, DBLP, Amazon, Orkut, LiveJournal, which are all available on the SNAP repository [Leskovec and Krevl, 2014], and come with sets of nodes that can be identified as "functional communities" (see Yang and Leskovec [2015]). For example, members of the social network Orkut may explicitly identify as being part of a user-formed group. Such user groups can be viewed simply as metadata about the network, though these still correspond to some notion of community organization that may be desirable to detect. We specifically consider the ten

Table 6.1.: We list the number of nodes (n) and edges (m) in each SNAP network, along with average set size $|T|$ and set conductance $\mathbf{cond}(T)$ for the ten largest communities.

Graph	n	m	$\|T\|$	$\mathbf{cond}(T)$
DBLP	317,080	1,049,866	3902	0.4948
Amazon	334,863	925,872	190	0.0289
LiveJournal	3,997,962	34,681,189	988	0.4469
Orkut	3,072,441	117,185,083	3877	0.6512

Table 6.2.: For experiments on SNAP datasets, we give F1 scores, conductance scores, runtimes, and output set sizes for finding the minimum conductance subset (mc), and for the set returned by learning a good resolution parameter (lr). Displayed is the average over results for the 10 largest communities in each network.

Graph	F1		cond.		run.		size	
	mc	lr	mc	lr	mc	lr	mc	lr
DBLP	0.01	**0.47**	0.02	0.16	4.9	11.4	31	11680
Amazon	0.73	**0.80**	0.00	0.02	0.3	0.7	142	288
LiveJournal	0.30	**0.54**	0.06	0.10	13.7	31.3	1556	2940
Orkut	0.44	**0.62**	0.42	0.46	129.8	272.6	2353	5727

largest communities from the 5000 best functional communities as identified by Yang and Leskovec [2015]. The size of each graph in terms of nodes (n) and edges (m), along with average set size $|T|$ and conductance $\mathbf{cond}(T)$ among the largest 10 communities, are given in Table 6.1.

Experimental Setup and Results We treat each functional community as an example cluster X. We build a superset of nodes R by growing X from a breadth first search until we have a superset of size $5|X|$, breaking ties arbitrarily. The size of R is chosen so that it comprises a localized region of a large graph, but is still significantly larger than the target cluster X hidden inside of it. We compare two approaches for detecting X within R. As a baseline approach we extract the best conductance subset of R. Then as our new approach we assume we are given $\text{cut}(X)$ and $\text{vol}(X)$ as additional semi-supervised information. This allows us to minimize the parameter fitness function (6.15), without knowing what X is. This outputs a resolution parameter α_X, and we then minimize $\text{cut}(S) - \alpha_X \text{vol}(S) + \alpha_X \text{vol}(R)$ over $S \subseteq R$ to output a set S_X. Table 6.2

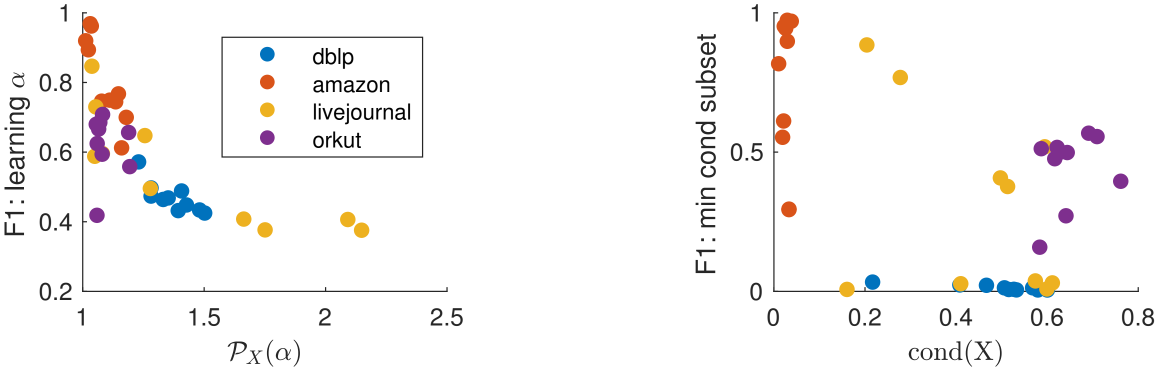


Figure 6.4.: Left: F1 scores for detecting clusters X by learning α vs. the minimum of $\mathcal{P}_X$. Right: F1 scores obtained by finding $X = \text{argmin}_{S \subseteq R} \textbf{cond}(S)$ vs. $\textbf{cond}(X)$. The decreasing, nearly linear pattern in the first plot indicates that the minimum value of $\mathcal{P}_X$ tells us something about how well the targeted functional communities match a notion of structural communities in a network. The right plot indicates $\textbf{cond}(X)$ does little to help us predict how detectable a cluster will be.

reports conductance, set size, runtimes, and F1 scores for both approaches, averaged over the ten communities in each network. Learning resolution parameters leads to significantly better F1 scores on every dataset. Finding the minimum conductance subset typically returns much smaller sets, and frequently finds sets so small that recall is very poor. Additionally, learning resolution parameters for local clustering can be done much more quickly than learning λ for LambdaCC. Our experiment highlights that learning specialized objective functions is more robust than simply minimizing a standard objective like conductance.

New Insights In addition to improving semi-supervised community detection, minimizing $\mathcal{P}_X$ allows us to measure how well a functional community matches the topological notion of a cluster. Figure 6.4 shows a scatter plot of F1 community recovery scores against the minimum of $\mathcal{P}_X$ for each experiment from Table 6.2. We note a downward sloping trend: small values of $\mathcal{P}_X$ near 1 tend to indicate that a cluster is highly "detectable," whereas a higher value of $\mathcal{P}_X$ gives some indication that the functional community may not in fact correspond to a good structural community. We also plot the F1 recovery scores for finding the minimum conductance subset of R against the conductance of functional communities. In this case we do not see any clear pattern, and we learn very little about the relationship between structural and functional communities.

6.5.3 Metadata and Global Clustering

Next we use our techniques to measure how strongly metadata attributes in a network are associated with actual community structure. In general, sets of nodes sharing metadata attributes should not be viewed as "ground truth" clusters [Peel et al., 2017], although they may still shed light on the underlying clustering structure of a network.

Email Network We first consider the largest connected component of an email network [Leskovec et al., 2007, Yin et al., 2017]. Each of the 986 nodes in the graph represents a faculty member at a European university, and edges represent email correspondence between members. We remove edge weights and directions, and consider an example clustering $\mathcal{C}_x$ formed by assigning faculty in the same academic department to the same cluster. We use our bisection method to approximately minimize the global parameter fitness function for the degree-weighted LambdaCC objective. We run our method until we find the best resolution parameter to within a tolerance of 10^{-8}, yielding a resolution parameter $\lambda_x = 6.5 \times 10^{-5}$ and a fitness score of $\mathcal{P}_{\mathcal{C}_x}(\lambda_x) = 1.34$.

To assess how good or bad a score of 1.34 is for this particular application, we construct a new fake metadata attribute by performing a random permutation of the department labels, which gives a clustering $\mathcal{C}_{fake}$. Approximately minimizing $\mathcal{P}_{\mathcal{C}_{fake}}$ yields a resolution parameter $\lambda_{fake} = 3.25 \times 10^{-5}$ and a score $\mathcal{P}_{\mathcal{C}_{fake}}(\lambda_{fake}) = 2.16$. The gap between the minima of $\mathcal{P}_{\mathcal{C}_{fake}}$ and $\mathcal{P}_{\mathcal{C}_x}$ indicates that although the true metadata partitioning does not perfectly map to clustering structure in the network, it nevertheless shares some meaningful correlation with the network's connectivity patterns. To further demonstrate this, we run Lambda-Louvain, using the resolution parameters λ_x and λ_{fake}. Running the clustering heuristic with λ_x outputs a clustering that has a normalized mutual information score (NMI) of 0.71 and an adjusted Rand index (ARI) score of 0.55 with $\mathcal{C}_x$. Using λ_{fake}, we get NMI and ARI scores of only 0.05 and 0.003 respectively when comparing with $\mathcal{C}_{fake}$.

Social Networks We repeat the above experiment on Caltech36, the smallest social network in the Facebook 100 dataset [Traud et al., 2012]. This network is a subset of Facebook with $n = 769$ nodes, defined by users at the California Institute of Technology at a certain point in September 2005. Every node in the network comes with anonymized metadata attributes reporting student/faculty status, gender, major, second major, residence, graduation year, and high school. We treat each metadata attribute as an example clustering $\mathcal{C}_x$. Any node with a value of 0 for an attribute we treat as its own cluster, as this indicates the node has no label for the given attribute. We do not run Algorithm 12 for each individual $\mathcal{C}_x$, since this would involve redundant computations of the LambdaCC LP relaxations for many of the same values of λ. Instead, we evaluate the denominator of $\mathcal{P}$, which is the same for all example clusterings, at 20 equally spaced λ values between $1/(8|E|)$ and $2/(|E|)$. We have chosen values of λ to be inversely proportional to the number of edges, since we expect the effect of a resolution parameter to depend on a network's size. We note for example that the resolution parameter corresponding to modularity is $\lambda = 1/(2|E|)$, which is also inversely proportional to $|E|$. Computing all of the LP bounds is the bottleneck in our computations, and takes just under 2.5 hours using a recently developed parallel solver for the correlation clustering relaxation [Ruggles et al., 2019].

Having evaluated the denominator of $\mathcal{P}$ at these values, we can quickly find the minimizer of $\mathcal{P}$ for each metadata attribute and a permuted fake metadata attribute to within an error of less than 10^{-5}. The smallest values of the parameter fitness function $\mathcal{P}$ obtained for both real and permuted (fake) metadata attributes are given below:

	S/F	Gen	Maj.	Maj. 2	Res.	Yr	HS
$\min \mathcal{P}_{real}$	1.30	1.73	2.03	2.12	1.35	1.57	2.11
$\min \mathcal{P}_{fake}$	1.65	1.80	2.12	2.12	2.11	2.09	2.12

We note that the smallest values of $\mathcal{P}$, as well as the largest gap between $\mathcal{P}$ for true and fake metadata clusterings, are obtained for the student/faculty status, residence, and graduation year attributes. This indicates that these attributes share the strongest correlation with the community structure at this university, which is consistent with independent results on the Facebook 100 dataset [Traud et al., 2012, Veldt et al., 2018b].

6.5.4 Local Clustering in Social Networks

In our final experiment we continue exploring the relationship between metadata and community structure in Facebook 100 networks. We find that minimizing a *local* parameter fitness function $\mathcal{P}$ can be a much better way to measure the community structure of a set of nodes than simply considering the set's conductance.

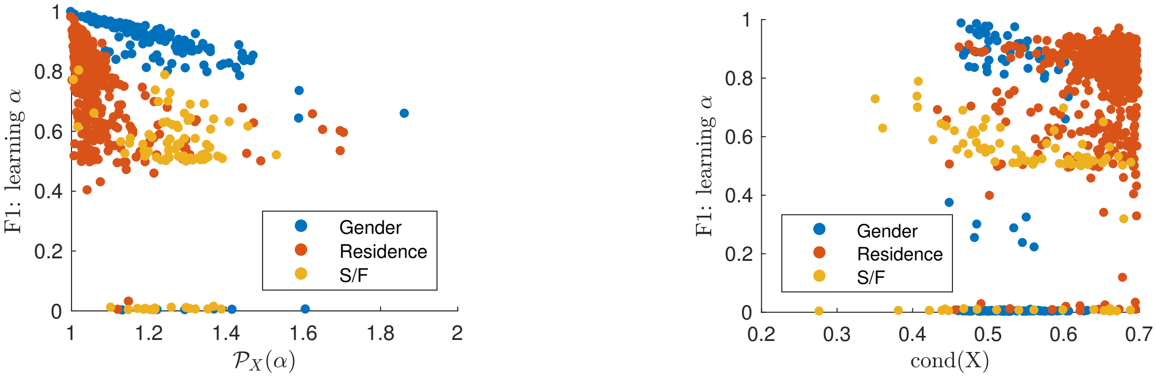
Data We perform experiments on all Facebook 100 networks, focusing on the student/faculty status, gender, residence, and graduation year metadata attributes. For the Caltech dataset in the last experiment, these attained the lowest scores for a *global* parameter fitness function, and furthermore these are the only attributes with a significant number of sets with nontrivial conductance. For the graduation year attribute, we focus on classes between 2006 to 2009, since these correspond to the four primary classes on each campus when the networks were crawled in September of 2005 [Traud et al., 2012].

Experimental Setup We return to an approach similar to our first experiment. For each network and metadata attribute, we consider sets of nodes identified by the same metadata label, e.g., X may represent all students in the class of 2008 at the University of Chicago. We will refer to these simply as *metadata sets*. A label of zero indicates no attribute is known, so we ignore these sets. We also discard sets that are larger than half the graph, or smaller than 20 nodes. We restrict to considering metadata sets with conductance at most 0.7, since conductance scores too close to 1 indicate that a set has little to no meaningful connectivity pattern. For each remaining metadata set X, we grow a superset R around X using a breadth first search, and stop growing when R contains half the nodes in the graph or is three times the size of X. We then minimize $\mathcal{P}_X$ as given by (6.15) to learn a resolution parameter α_X. This allows us to find $S_X = \text{argmin}_{S \subseteq R} \text{cut}(S) - \alpha_X \text{vol}(S)$, and we then compute the F1 score between S_X and X. Our goal here is not to develop a new method for community detection. Rather, computing the F1 score and the minimum of $\mathcal{P}_X$ provide ways to measure how well a metadata set conforms to a topological notion of community structure, and report how detectable the set is from an algorithmic perspective.

Results While computing conductance scores provides a good first order measure of a node's community structure, we find that minimizing $\mathcal{P}$ provides more refined information for the detectability of clusters. In Figure 6.5 we show scatter plots of F1 detection scores against both $\min \mathcal{P}_X$ as well as $\textbf{cond}(X)$ for each metadata set X. We see that especially for the gender and residence metadata sets across all networks, there is a much clearer relationship between F1 scores and $\min \mathcal{P}_X$. Values of $\mathcal{P}_X$ very close to 1 map to F1 scores near 1, and as $\mathcal{P}_X$ increases we see a downward sloping trend in F1 scores. In the conductance plot we do not see the same trend.

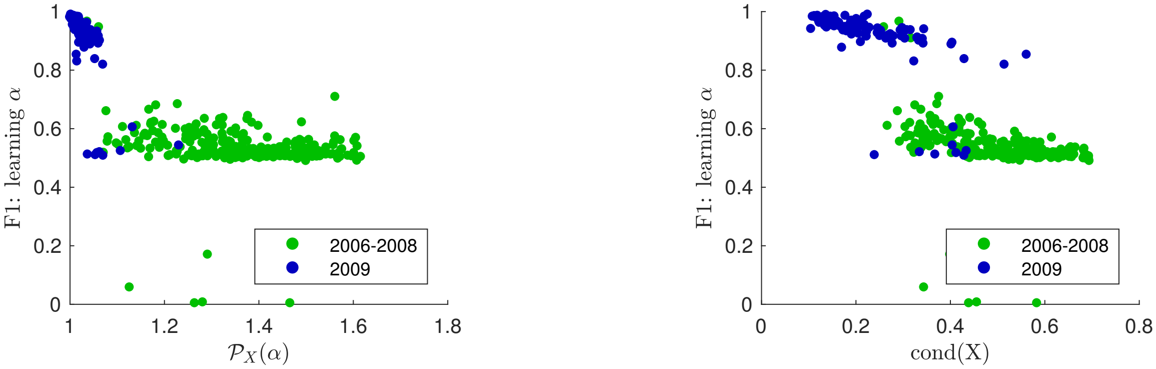
Figures 6.5c and 6.5d show results for metadata sets associated with the 2006-2009 graduation years. For this attribute there appears to be a relationship between both conductance and the $\min \mathcal{P}$ scores. Furthermore, in both plots we see a separation of the points roughly into two clusters. A deeper exploration of these trends reveals that the 2009 graduation class accounts for the majority of one of these two clusters, and there appears to be an especially clear trend between F1 detection scores and both $\textbf{cond}(X)$ and $\mathcal{P}_X$ for this class. In order to explain this, we further investigated the connectivity patterns of the main four student classes across all universities.

New Insights Figure 6.6 shows violin plots for $\textbf{cond}(X)$, $\text{cut}(X)$, and $\text{vol}(X)$ for metadata sets associated with graduation years from 2006 to 2009. Overall, conductance decreases as graduation year increases. We notice that class sizes for the 2009 graduation year are much smaller on average. When these datasets were generated, Facebook users needed a .edu email address to register an account. Thus, in September 2005, the graduation class of 2009 was made up primarily of new freshman who just started college, many of whom had not registered a Facebook account yet. In-



(a) $\mathcal{P}_X$: Gender, S/F, Residence

(b) $\mathbf{cond}(X)$: Gender, S/F, Residence



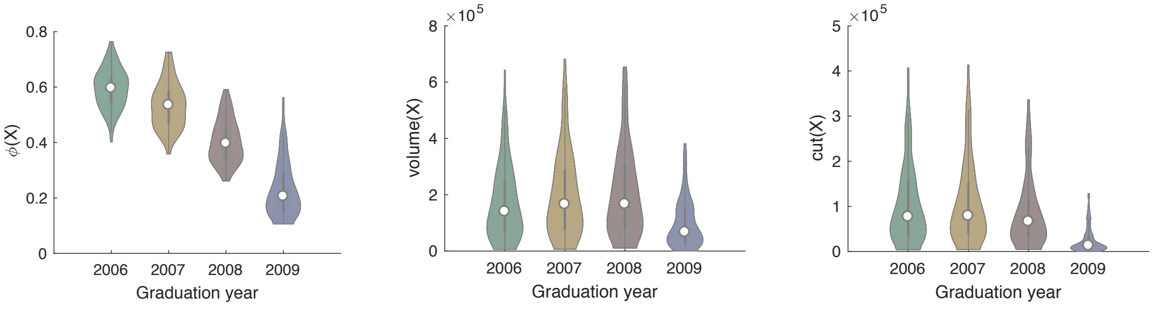
(c) $\mathcal{P}_X$: Grad Year

(d) $\mathbf{cond}(X)$: Grad Year

Figure 6.5.: Minimizing $\mathcal{P}_X$ gives us refined information about the connectivity structure of different sets sharing metadata attributes in Facebook 100 networks. Plotting F1 detection scores against the minimum of $\mathcal{P}_X$ shows especially clear trends for the gender and residence metadata attributes. Plots for the graduation year attribute highlight an anomaly in the connectivity patterns of the 2009 graduating year classes. We explore this in further depth in the main text.

terestingly, we see a slight decrease in the median cut score from 2007 to 2008, and a significant decrease from 2008 to 2009 (Figure 6.6c). This suggests that although there were fewer freshman on Facebook at the time, on average they had a greater tendency to establish connections on Facebook among peers in their same graduation year.

Figure 6.6 suggests that in the early months of Facebook, with each new year, students in the same graduating class tended to form tighter Facebook circles with members in their own class. To further explore this hypothesis, for each of the 100 Facebook networks we consider each node from a graduating class between 2006 and 2009. In each network we compute the average *in-class connection ratio*, i.e., the number of Facebook friends each person has inside the same graduating class, divided by the total number of Facebook connections that the person has across the entire



(a) Conductance (b) Volume (c) Cut

Figure 6.6.: As graduation year increases, conductance scores on the whole tend to decrease. White dots indicate median value. The 2009 graduation year metadata sets tend to be much smaller in volume, but also have very small cut scores, indicating that freshman in 2005 were largely connecting on Facebook with people in their same class.

university. In 97 out of 100 networks (all networks except Caltech36, Hamilton46, and Santa74), this ratio strictly increases as graduation year increases. For Hamilton46 and Santa74, the ratio is still significantly higher for the 2009 graduation class than any other class. If we average this ratio across all networks, as the graduation year increases from 2006 to 2009, the ratios strictly increase: 0.39 for 2006, 0.45 for 2007, 0.57 for 2008, and 0.75 for the class of 2009. In other words, 75% of an average college freshman's Facebook friends were also freshman, whereas only 39% of an average senior's Facebook friends were seniors.

Traud et al. [2012] were the first to note the influence of the graduation year attribute on the connectivity structure of Facebook 100 networks. Later, Jacobs et al. [2015] observed differences in the way subgraphs associated with different graduation years evolved and matured over time. These authors noted in particular that the subgraphs associated with the class of 2009 tend to exhibit very skewed degree distributions and comparatively low average degrees. Our observations complement these results, by highlighting heterogeneous behavior in the way members of different classes interacted and connected with one another during the early months of Facebook.

7. CONCLUSIONS

We end with a summary of major contributions and a discussion of open questions. In this thesis, we have presented a generalized framework for graph clustering, accompanied by several theoretical results and broadly applicable optimization tools. These contributions have allowed us to address two major practical challenges in graph clustering: choosing the appropriate model for community structure to use in different contexts, and dealing with the inherent intractability of graph clustering.

LambdaCC addresses the first of these two challenges by giving practitioners the power to define their own desired balance between internal density and external sparsity in the formation of graph clusters. The value of the resolution parameter λ implicitly controls both the sparsity of cuts (Theorem 3.4.2) and the density of clusters (Theorem 3.4.3) formed by optimizing the objective. LambdaCC also generalizes and interpolates between many already existing techniques (Figure 3.2). Furthermore, when it is not clear a priori how to set this parameter λ, our optimization framework for learning resolution parameters (Chapter 6) outlines a principled method for automatically setting λ to fit a specified example of community structure. This provides a way to learn good models and objective functions to fit a wide range of clustering applications.

We have addressed the intractability of graph clustering in several ways. To begin, the approximation algorithms outlined in Chapter 4 provide ways to obtain provable approximation guarantees for NP-hard objectives in polynomial time. Furthermore, Chapter 5 outlines a framework for metric constrained optimization, which enables practitioners to implement these types of approximation algorithms on a much larger scale than was previously possible. The heuristic methods presented and analyzed in Chapter 3, and the LP integrality gap shown in Chapter 4, also help address the challenge of intractability, albeit in different ways. Heuristic methods allow us to obtain fast results on problems that are too large for LP-based algorithms. Meanwhile, the LP integrality gap proof furthers our understanding of the fundamental limitations of applying certain techniques when attempting to overcoming problem intractability.

7.1 Discussion and Open Questions

The results presented in the later chapters of this thesis arose specifically as answers to open questions posed after the initial presentation of the LambdaCC framework [Veldt et al., 2018b]. The metric constrained optimization framework (Chapter 5), which is applicable in a wide variety of clustering applications, was born out of a desire to compute lower bounds for LambdaCC in order to make plots such as the ones in Figure 3.3. The integrality gap for LambdaCC for small values of λ, first published in ISAAC 2018 [Gleich et al., 2018], answered open questions about why LP rounding techniques failed for arbitrary λ, despite working well for $\lambda \geq 1/2$. Finally, Chapter 6 answers a question left open in Chapter 3, regarding how to find the "right" approach for graph clustering in different applications. The theoretical and experimental results in Chapter 3 showed us that choosing an appropriate objective function or algorithm is essentially equivalent to choosing a proper resolution parameter. Chapter 6 then addresses the resulting open question of what it means to choose a proper parameter to fit a specific example of community structure.

Along the way, numerous other questions arose that still have not been fully answered. We conclude by visiting several open questions that are the focus of ongoing and future research.

7.1.1 Improved Complexity Results for LambdaCC

The integrality gap in Section 4.4 indicates an inherent limitation in applying LP-rounding techniques to obtain theoretical approximation guarantees for LambdaCC. This does immediately imply any results regarding the inherent hardness of LambdaCC, beyond proving that one type of technique will not lead to approximations better than $O(\log n)$. The obvious question to ask is whether other proof techniques for correlation clustering can lead to a constant factor algorithm. Another direction is to try to prove more refined hardness results for LambdaCC. For example, if we assume the unique games conjecture is true, is it NP-hard to obtain constant factor approximations for LambdaCC when λ is arbitrary? We conjecture here, with some supporting discussion, that the latter direction seems more promising.

Finding better lower bounds seems challenging. Other techniques, such as semidefinite programming relaxations [Swamy, 2004, Charikar et al., 2005], have been applied to approximate the objective of *maximizing agreements*. However, with almost no exception, the best approximation factors for minimizing disagreements rely crucially on the lower bound provided by solving the canonical linear programming relaxation. This is true for specially weighted variants [Ailon et al., 2008, Puleo and Milenkovic, 2015, Veldt et al., 2018b], as well as other variants including higher-order generalizations [Li et al., 2017, Gleich et al., 2018, Fukunaga, 2018], and local-error objective functions for correlation clustering [Puleo and Milenkovic, 2018, Charikar et al., 2017]. To our knowledge, the only other lower bound that has been used to prove approximation results for minimizing disagreements is a lower bound based on fractional packings of bad triangles. This bound was used by Bansal et al. [2004] to obtain the first constant factor approximation for the unweighted case. Ailon et al. [2008]

later used a fractional packing argument to develop a fast 3 approximation for the unweighted case. Unfortunately, the best fractional packing lower bound is equivalent to solving what amounts to a weaker LP relaxation. For the unweighted case, this corresponds to

$$\begin{array}{ll} \text{minimize} & \sum_{i<j} x_{ij} \\ \text{subject to} & x_{ij} + x_{ik} + x_{jk} \geq 1 \text{ for all } \{i,j,k\} \in T \\ & x_{ij} \geq 0 \text{ for all } i,j \end{array} \tag{7.1}$$

where T represents the set of “bad” triplets of nodes, in which two edges are positive and the other is negative. In this LP, the variable x_{ij} can be interpreted as an indicator for whether or not a mistakes is made at edge ij. The constraint $x_{ij} + x_{ik} + x_{jk} \geq 1$ encodes the fact that we have to make at least one mistake at every bad triangle. If for every negative edge $(i,j) \in E^-$ we replace variable x_{ij} with $1 - \hat{x}_{ij}$, then (7.1) can be re-written

$$\begin{array}{ll} \text{minimize} & \sum_{(i,j)\in E^-} (1 - \hat{x}_{ij}) + \sum_{(i,j)\in E^+} x_{ij} \\ \text{subject to} & \hat{x}_{ij} - x_{ik} - x_{jk} \geq 0 \text{ for all } \{i,j,k\} \in T, \text{ where } (i,j) \in E^-, \\ & x_{ij} \leq 0 \text{ for all } (i,j) \in E^+ \\ & \hat{x}_{ij} \leq 1 \text{ for all } (i,j) \in E^-. \end{array} \tag{7.2}$$

Note that (7.2) is just an LP relaxation containing a subset of triangle inequality constraints, and therefore gives a looser lower bound than the canonical LP relaxation. This is also true for weighted variants. Therefore, the integrality gap for the fractional packing lower bound for LambdaCC will also be $\Omega(\log n)$ at best. Furthermore, in numerical experiments, we found that even when the canonical LP relaxation of LambdaCC provides good lower bounds for certain graphs when λ is small, the fractional packing LP gives poor lower bounds, which get increasingly worse as λ decreases.

Can we show new hardness results for LambdaCC? To summarize the previous section, all of the best approximation algorithms for specially weighted variants of correlation clustering rely on rounding the canonical LP relaxation. The other known lower bound corresponds to a weaker relaxation.[1] Thus, if we are to obtain a constant factor approximation for LambdaCC when λ is arbitrary, this will require developing new ways to obtain good lower bounds for correlation clustering. A new approach would need to differ substantially from the techniques that have been used in the past two decades of research on correlation clustering. This seems like a very challenging direction.

Perhaps a more promising direction would be to prove new hardness results for LambdaCC when λ is arbitrary. We have already shown that in terms of LP integrality gaps, LambdaCC with small λ

---

[1] Other techniques do exist for minimizing disagreements when the number of clusters is bounded [Giotis and Guruswami, 2006, Coleman et al., 2008], but this is a different direction altogether.

is in some sense “just as hard” as the the most general instances of correlation clustering. However, there are additional hardness results known about general correlation clustering that do not directly apply to LambdaCC. General correlation clustering is known to be equivalent to the minimum multicut problem [Emanuel and Fiat, 2003, Demaine and Immorlica, 2003, Demaine et al., 2006]. This holds even in the case where edges are either positive with weight one, negative with weight one, or zero. Given this equivalence, we immediately inherent UG-hardness results for general correlation clustering [Chawla et al., 2015]. For example, constant factor approximations are NP-hard if the unique games conjecture is true. This also provides evidence that obtaining an approximation better than $O(\log n)$ for general correlation clustering is unlikely, as this would imply improved approximation results for the extensively studied multicut problem. Two related open question arise: is LambdaCC also equivalent to minimum multicut? Even if it is not, is it possible to prove that LambdaCC is also NP-hard to approximate to within any constant factor if the unique games conjecture is true? Given existing hardness results for general correlation clustering, this direction seems more promising than trying to develop completely new approximation techniques in hope of a constant factor approximation for arbitrary λ. However, obtaining new hardness results is still challenging, and initial exploration of this direction has yet to yield a definitive answer.

Better than worst case analysis for LP relaxations? Independent of the previous questions regarding worst case complexity results, another open direction is better-than-worst-case analysis for the LP integrality gap. This direction arises from the fact that despite our worst case analysis (which applies specifically to a specially constructed expander graph), the LP relaxation tends to provide very good lower bounds in many real-world instances. In fact, in numerous experimental results, we have confirmed that the LambdaCC LP relaxation provides tighter bounds for small λ than for $\lambda \geq 1/2$. This is in some sense precisely the opposite of what the worst-case theory predicts.

As an example, consider the curves in the Figure 3.3. On all three graphs, the approximation guarantees for Lambda-Louvain are much better for small λ than for λ near 1. This pattern also holds true in experiments on small graphs presented in early work on the LambdaCC framework (see experiments in Section 5.1 of Veldt et al. [2018b]). We conjecture that this is *not* because the algorithm is doing a better job optimizing the LambdaCC objective for small λ. Instead, we believe this pattern arises because the LP lower bound is in fact tighter for small λ than it is for large λ. For example, we confirmed that this is exactly the case for two very small real world graphs, Karate [Zachary, 1977], and Dolphins [Lusseau et al., 2003] (both graphs are available in the Newman group of the SuiteSparse matrix collection [Davis and Hu, 2011]). For these small graphs, we computed both the LP relaxations as well as the optimal LambdaCC clusterings for a wide range

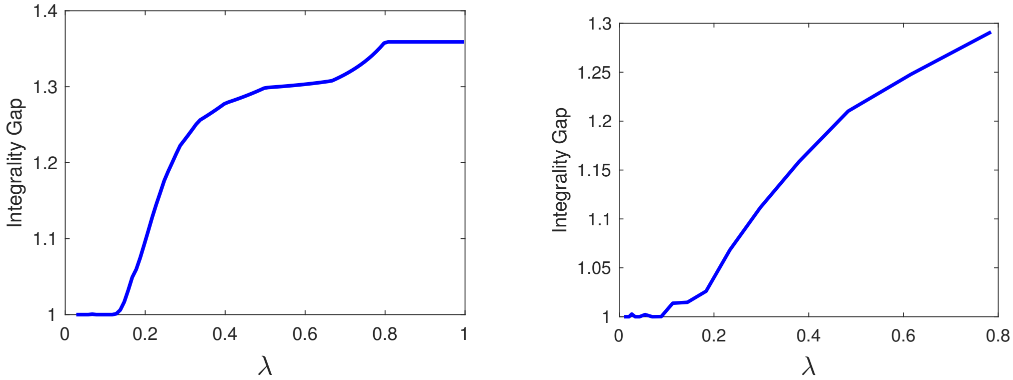


Figure 7.1.: Ratios between optimal LambdaCC clustering score and the LambdaCC LP lower bound, as λ increases. The left plot shows this integrality gap for Karate (34 nodes), and the right plot shows the gap for Dolphins (62 nodes).

of λ values. Figures 7.1 show ratio between optimal scores and LP lower bounds for these graphs. In all cases, the gap increases as λ increases.

Given these observations, it remains an open question to explore what theoretical assumptions can be made in order to show when the LP relaxation will have a more favorable integrality gap. Are there special properties of many real-world graphs that make this lower bound better than it may seem based on worst case analysis? Understanding the behavior of the LP relaxation will also lead to an improved understanding of our framework for learning resolution parameters presented in Chapter 6, which fundamentally relies on lower bounds provided by LP relaxations.

7.1.2 $O(n^2)$-memory algorithms for metric optimization

The metric constrained optimization framework developed in Chapter 5 relies on applying Dykstra's projection method to solve problems with $O(n^2)$ variables and $O(n^3)$ constraints. In addition to the $O(n^2)$ primal variables, Dykstra's methods also maintains a dual variable for each constraint. Therefore, in the worst case, this approach still requires cubic memory. It would be a significant breakthrough to develop techniques for metric constrained optimization that require only quadratic memory, and satisfy good convergence properties. We outline an approach that provides a first step in this direction, though the overall procedure ultimately fails due to poor convergence results.

There do exist projection methods whose memory requirement is given in terms of the number of variables, rather than the number of constraints. In particular, we implemented variants of Bauschke's projection method [Bauschke, 1996] and Haugazeau's projection method [Haugazeau, 1968, Bauschke and Combettes, 2017], which are primal-only methods and do not compute dual variables as Dykstra's does. Thus, when applied to metric constrained optimization, these methods require only $O(n^2)$ memory, and produce a sequence of iterates that is guaranteed to converge to the optimal solution to the metric optimization problem. Unfortunately, the convergence rate of these projection methods is significantly worse than the linear convergence rate that Dykstra exhibits for half-space constraints. Previous work on Bauschke's method and related projection methods [Halpern, 1967, Lions, 1977, Wittmann, 1992] focus simply on asymptotic convergence proofs. Furthermore, since these methods do not keep track of dual variables, we cannot compute lower bounds on the optimal objective in order to know when the method is near convergence. Other helpful results on projection methods are detailed in the work of Bauschke and Koch [2015] and Censor [2006].

To illustrate the shortcomings of these $O(n^2)$-memory techniques, we include several numerical experiments for using Bauschke's method for metric constrained optimization. Without providing full details, we note that Bauschke's projection method computes iterates in the following way:

$$\mathbf{x}_{k+1} = \lambda_{k+1}\mathbf{z} + (1-\lambda_{k+1})P_M P_{M-1}\cdots P_2 P_1 \mathbf{x} \tag{7.3}$$

where $\mathbf{x}_k$ is the kth iterate ($\mathbf{x}$ stores distance variables x_{ij}), and $\mathbf{z}$ is the starting point that we wish to project onto a set of constraints. For metric optimization, $\mathbf{z}$ is set in a manner similar to what is shown in Algorithm 10. The value λ_{k+1} is called a "steering sequence," chosen so that the method converges to the projection of $\mathbf{z}$ onto a certain convex set. For metric optimization, this convex set is defined by the feasible set of a quadratic or linear program that includes metric constraints. Here each P_i is a projection operator onto a single smaller convex set. In this case, each smaller convex set corresponds to a specific linear constraint (e.g., a triangle inequality constraint) in a quadratic program, and $M = O(n^3)$ is the number of constraints.

The standard steering sequence used for Bauschke's method is

$$\lambda_k = \frac{\sigma}{k+1}$$

where σ is any fixed constant. Regardless of the value of σ, Bauschke's method will converge asymptotically to the optimal solution of the optimization problem. However, the value of σ will significantly change the behavior of the algorithm, and it is not clear how to best set it in practice.

We tested both Bauschke's method and Dykstra's method in solving the quadratic regularization of the Leighton-Rao LP relaxation for sparsest cut (given in expression (5.3) in Chapter 5). We tested each method on a number of the graphs considered in Chapter 5. For these experiments, we tested a range of values for γ, which controls the relationship between the original LP and the quadratic program that the projection methods are actually solving. For each γ, we also varied the parameter σ. Convergence plots are shown in Figures 7.2-7.5. Dykstra's method maintains a primal objective score and a dual objective score (shown in blue and green respectively). The red dotted line corresponds to the optimal solution to the quadratic program. For Bauschke's method, we include a different curve for the primal objective score computed by running the algorithm for each value of σ. In theory, all of these methods asymptotically converge to the optimal solution (the red dotted line). However, these curves begin to make very slow progress after the first several hundred iterations. The curves in fact appear to level off almost completely, with each curve leveling off at a different place depending on the value of σ. In practice, Bauschke's method does not have a way to check how close to optimality it is, and it is unclear how to choose σ to obtain the best results for different problems. Note in particular that we only know the location of the optimal objective (given by the red line) because of running Dykstra's method on the same problem. Thus, Bauschke's method, despite avoiding the $O(n^3)$ memory requirement, exhibits severe limitations. It remains an open problem whether it is possible to develop an $O(n^2)$-memory algorithm with better convergence properties.

7.1.3 Parallel projection methods for metric optimization

We conclude with a brief discussion of an ongoing line of research that has already led to notable improvements in metric constrained optimization, based on performing simultaneous projections steps. We first note that parallel versions of Dykstra's method and Hildreth's method already exist [Iusem and De Pierro, 1991]. These work by averaging a large number of very small changes at each iteration, equal to the number of constraints. For metric optimization there are $O(n^3)$ constraints, and we find that, in practice, this approach leads to changes that are so small that no meaningful progress is made from one iteration to the next. However, progress can be made by identifying blocks of triangle inequalities at which projection steps can be performed in parallel without conflicts or locking variables.

More specifically, consider performing projections steps associated with two triplets of nodes, $t_1 = \{a, b, c\}$, and $t_2 = \{i, j, k\}$. In a metric optimization problem, triplet t_1 involves variables $\{x_{ab}, x_{bc}, x_{ac}\}$. Similarly, triplet t_2 is associated with variables $\{x_{ij}, x_{jk}, x_{ik}\}$. Note that if these triplets share two indices in common (e.g., $a = i$ and $b = j$), then we cannot perform projections at both constrains in parallel without conflict, since one variable (e.g., $x_{ab} = x_{ij}$) would be updated by both projections. However, if t_1 and t_2 share at most one index in common, then $\{x_{ab}, x_{bc}, x_{ac}, x_{ij}, x_{jk}, x_{ik}\}$ are all distinct and we can perform projection steps for Dykstra's method at t_1 and t_2 at the same time.

Based on this observation, in ongoing work we have developed approaches for ordering node triplets in such a way that large block of constraints can be visited simultaneously, and multiple projection steps can be performed at once. Full details are contained in a preprint available online [Ruggles et al., 2019], and code for the resulting methods is given in `https://github.com/nveldt/ParallelDykstras`. To illustrate the performance improvements, Table 7.1 shows results for running the parallel approach for 20 iterations on the correlation clustering problems introduced in Section 5.7.5. Speedups are displayed when using 8, 16, and 32 cores. In addition, we compared the original serial projection method against the parallel approach on a problem with over 17,000 nodes, corresponding to a metric optimization problem with nearly 3 trillion constraints.

Table 7.1.: Results for running the parallel version of Dykstra's method in solving the metric constrained LP relaxation of correlation clustering.

Graph	# constraints.	# Cores	Times (s)	Speedup
ca-GrQc	3.6×10^{10}	1	2632	1
$n = 4158$		8	562	4.68
		16	429	6.14
		32	358	7.35
Power	6.0×10^{10}	1	4521	1
$n = 4941$		8	890	5.08
		16	696	6.50
		32	576	7.85
ca-HepTh	3.2×10^{11}	1	19826	1
$n = 8638$		8	4682	4.23
		16	3252	6.10
		32	2603	7.62
ca-HepPh	7.0×10^{11}	1	47309	1
$n = 11204$		8	10313	4.59
		16	7066	6.70
		32	5889	8.03
ca-AstroPh	2.9×10^{12}	1	187045	1
$n = 17903$		8	40146	4.66
		16	35397	5.28
		32	24374	7.67
		64	16325	11.46

Although the speedup obtained on these problems is modest, this serves as a first step in accelerating Dykstra's method specifically for metric constrained optimization. The parallel approach outlined here was in fact already used in several of the experiments in Chapter 6. This approach led to faster runtimes for solving the LambdaCC relaxation than we would have obtained using only the serial techniques. It may be possible to further improve these techniques by exploring other ways to visit metric constraints in parallel. More broadly, this motivates future work on developing efficient parallel projection methods for other large scale optimization problems that come with large, sparse, and specially structured constraint matrices.

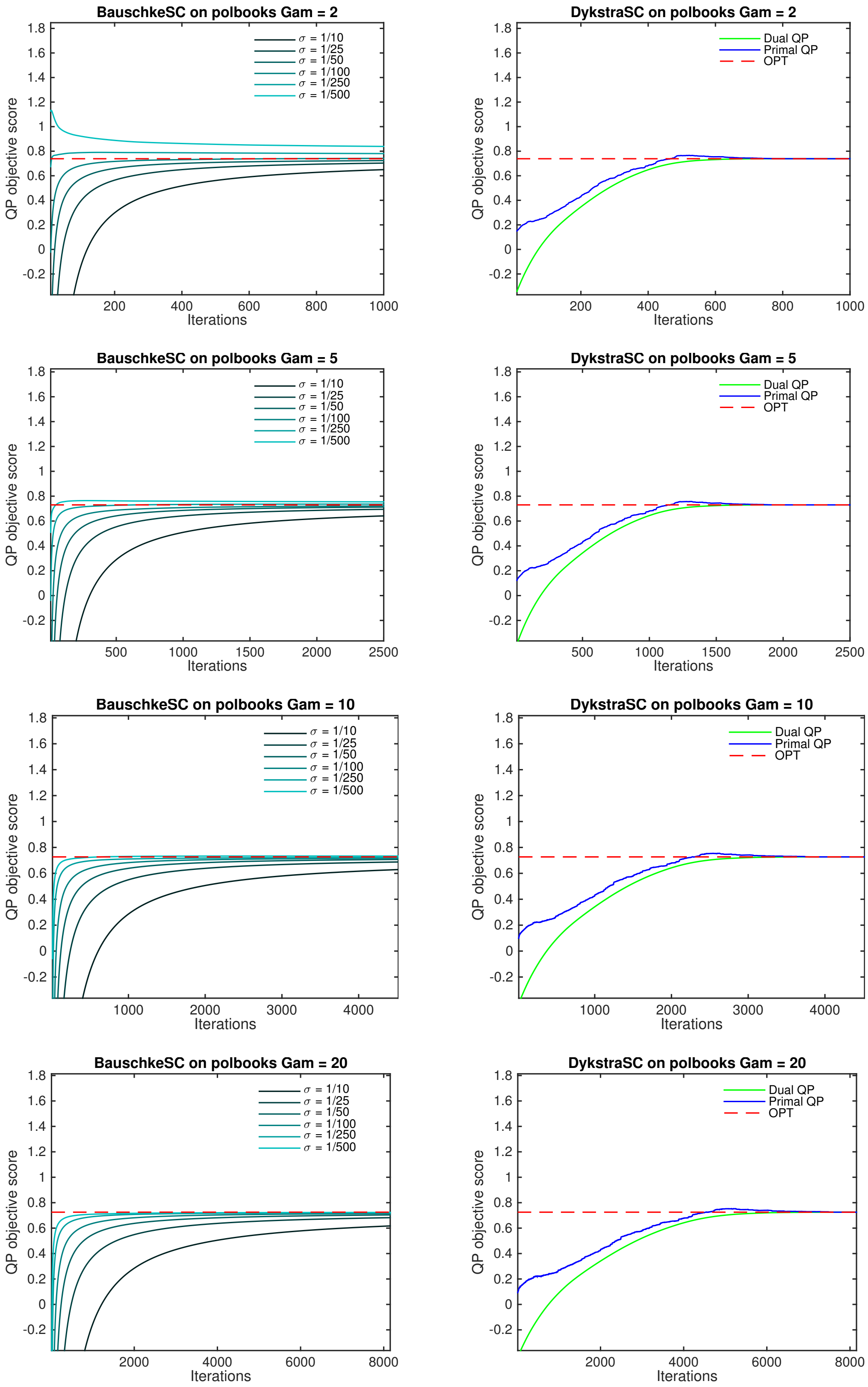
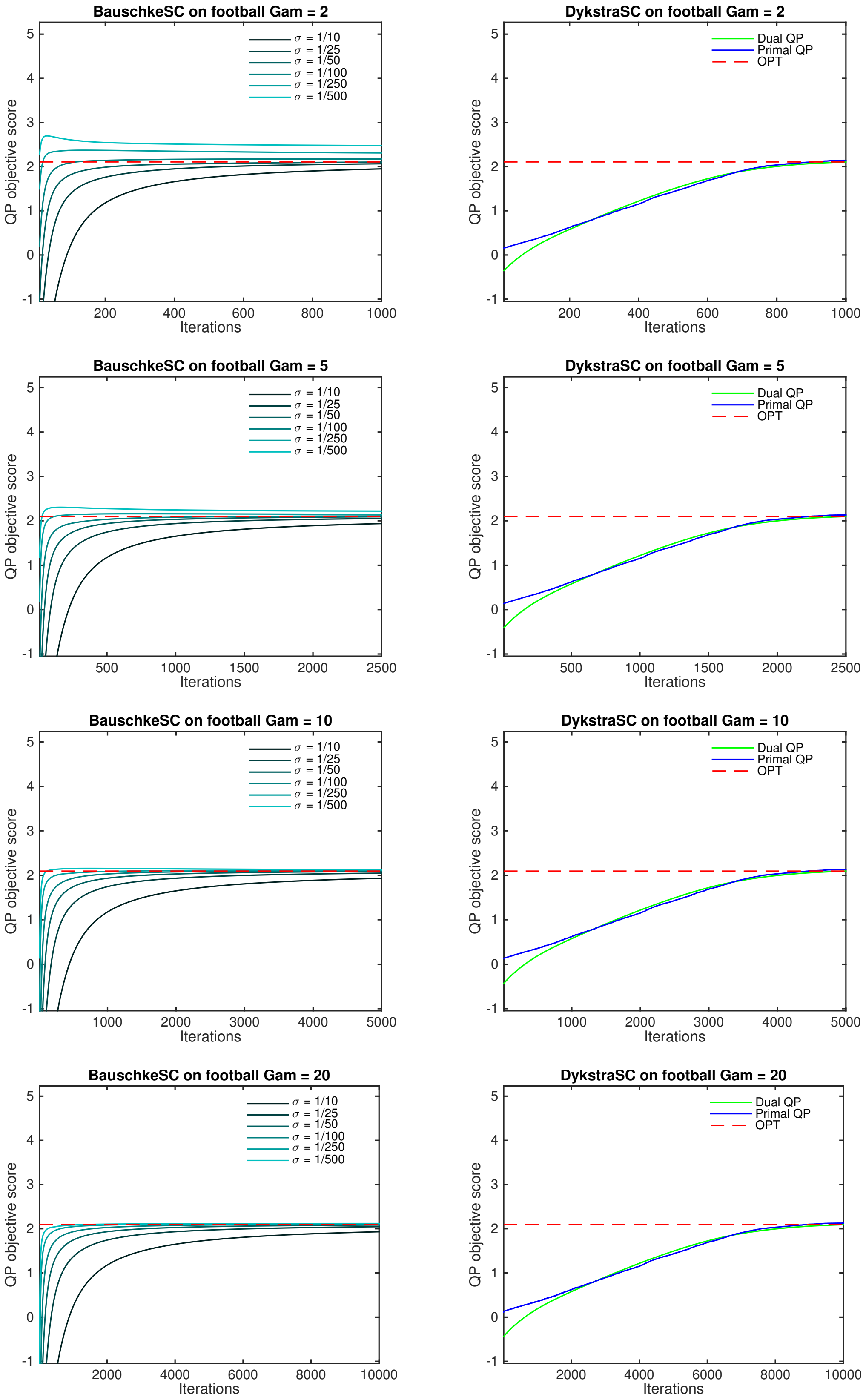


Figure 7.2.: Convergence plots for Polbooks.



BauschkeSC on football Gam = 2

DykstraSC on football Gam = 2

BauschkeSC on football Gam = 5

DykstraSC on football Gam = 5

BauschkeSC on football Gam = 10

DykstraSC on football Gam = 10

BauschkeSC on football Gam = 20

DykstraSC on football Gam = 20

Figure 7.3.: Convergence plots for Football.

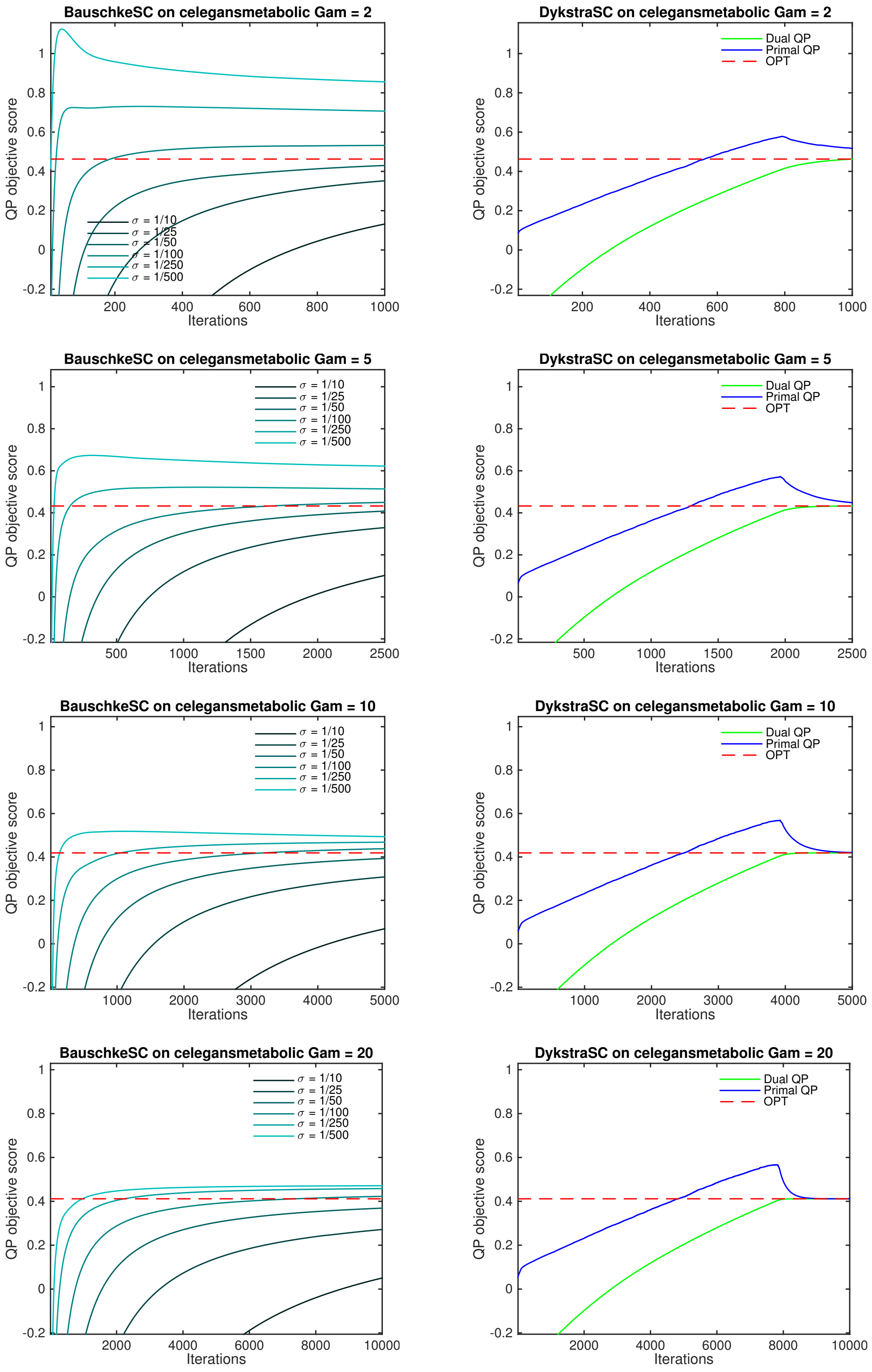
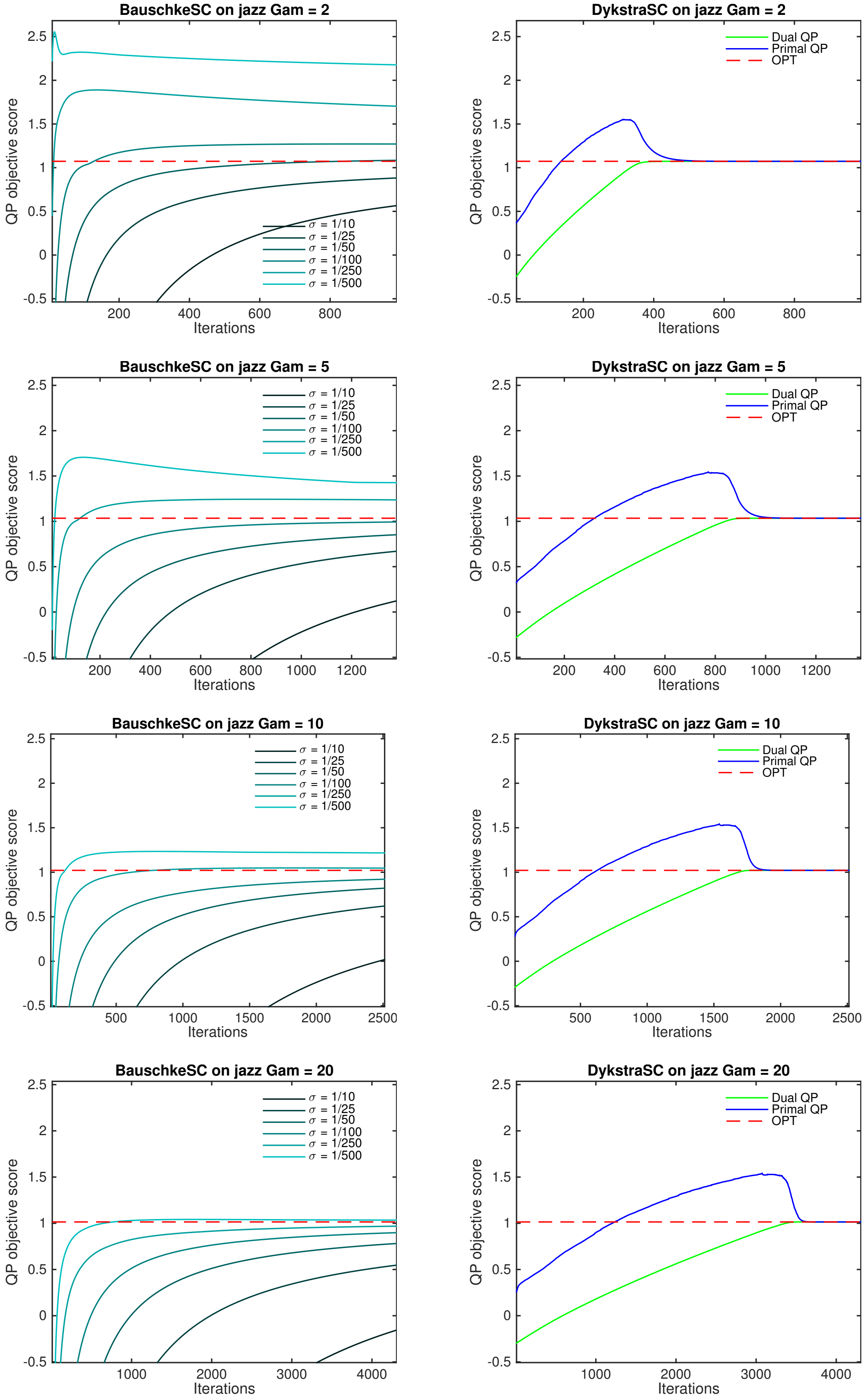


Figure 7.4.: Convergence plots for C Elegans Metabolic.


BauschkeSC on jazz Gam = 2

DykstraSC on jazz Gam = 2

BauschkeSC on jazz Gam = 5

DykstraSC on jazz Gam = 5

BauschkeSC on jazz Gam = 10

DykstraSC on jazz Gam = 10

BauschkeSC on jazz Gam = 20

DykstraSC on jazz Gam = 20

Figure 7.5.: Convergence plots for Jazz.

REFERENCES

REFERENCES

Emmanuel Abbe. Community detection and stochastic block models: Recent developments. *Journal of Machine Learning Research*, 18(177):1–86, 2018.

Lada A. Adamic and Natalie Glance. The political blogosphere and the 2004 u.s. election: Divided they blog. In *Proceedings of the 3rd International Workshop on Link Discovery*, LinkKDD '05, pages 36–43, New York, NY, USA, 2005. ACM. ISBN 1-59593-215-1. doi: 10.1145/1134271.1134277. URL `http://doi.acm.org/10.1145/1134271.1134277`.

Ilan Adler and Renato D.C. Monteiro. A geometric view of parametric linear programming. *Algorithmica*, 8:161–176, 1992.

G. Agarwal and D. Kempe. Modularity-maximizing graph communities via mathematical programming. *The European Physical Journal B*, 66(3):409–418, Dec 2008. ISSN 1434-6036. doi: 10.1140/epjb/e2008-00425-1. URL `https://doi.org/10.1140/epjb/e2008-00425-1`.

Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2237–2246. JMLR.org, 2015. URL `http://dl.acm.org/citation.cfm?id=3045118.3045356`.

Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.

Daniel Aloise, Sonia Cafieri, Gilles Caporossi, Pierre Hansen, Leo Liberti, and Sylvain Perron. Column generation algorithms for exact modularity maximization in networks. *Physical Review E : Statistical, Nonlinear, and Soft Matter Physics*, 82(4):pp 046112–1 – 046112–9, October 2010. doi: 10.1103/PhysRevE.82.046112. URL `https://hal-enac.archives-ouvertes.fr/hal-00934661`.

Charles J Alpert and Andrew B Kahng. Recent directions in netlist partitioning: a survey. *Integration, the VLSI journal*, 19(1-2):1–81, 1995.

Yael Anava, Noa Avigdor-Elgrabli, and Iftah Gamzu. Improved theoretical and practical guarantees for chromatic correlation clustering. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 55–65, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-3469-3. doi: 10.1145/2736277.2741629. URL `https://doi.org/10.1145/2736277.2741629`.

Reid Andersen and Kevin Lang. An algorithm for improving graph partitions. In *Proceedings of the 19th annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, pages 651–660, January 2008.

Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using PageRank vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 2006. URL `http://www.math.ucsd.edu/~fan/wp/localpartition.pdf`.

Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2), 2009.

Megathenis Asteris, Anastasios Kryillidis, Dimitris Papailiopoulos, and Alexandros G. Dimakis. Bipartite correlation clustering: Maximizing agreements. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 51, 2016.

David A Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. *Graph partitioning and graph clustering*, volume 588. American Mathematical Soc., 2013.

Shai Bagon and Meirav Galun. Large scale correlation clustering optimization. *arXiv preprint arXiv:1112.2903*, 2011.

Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56: 89–113, 2004.

Vladimir Batagelj and Andrej Mrvar. Pajek datasets. `http://vlado.fmf.uni-lj.si/pub/networks/data/`, 2006.

Heinz H. Bauschke. The approximation of fixed points of compositions of nonexpansive mappings in Hilbert space. *Journal of Mathematical Analysis and Applications*, 202(1):150 – 159, 1996. ISSN 0022-247X. doi: https://doi.org/10.1006/jmaa.1996.0308. URL `http://www.sciencedirect.com/science/article/pii/S0022247X9690308X`.

Heinz H Bauschke and Patrick L Combettes. *Convex analysis and monotone operator theory in Hilbert spaces*, volume 2011. Springer, 2017.

Heinz H Bauschke and Valentin R Koch. Projection methods: Swiss army knives for solving feasibility and best approximation problems with halfspaces. *Contemp. Math*, 636:1–40, 2015.

T. Beier, T. Kroeger, J. H. Kappes, U. Kthe, and F. A. Hamprecht. Cut, glue, amp; cut: A fast, approximate solver for multicut partitioning. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 73–80, June 2014. doi: 10.1109/CVPR.2014.17.

Thorsten Beier, Fred A Hamprecht, and Jorg H Kappes. Fusion moves for correlation clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3507–3516, 2015.

Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of computational biology*, 6(3-4):281–297, 1999.

Aditya Bhaskara, Samira Daruki, and Suresh Venkatasubramanian. Sublinear algorithms for MAX CUT and correlation clustering. In *Proc. International Conference on Automata, Logic and Programming*, 2018.

Anindya Bhattacharya and Rajat K. De. Divisive correlation clustering algorithm (dcca) for grouping of genes: detecting varying patterns in expression profiles. *Bioinformatics*, 24(11):1359–1366, 04 2008. ISSN 1367-4803. doi: 10.1093/bioinformatics/btn133. URL `https://dx.doi.org/10.1093/bioinformatics/btn133`.

Ernesto G. Birgin and Marcos Raydan. Robust stopping criteria for Dykstra's algorithm. *SIAM Journal on Scientific Computing*, 26(4):1405–1414, 2005. doi: 10.1137/03060062X.

Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre1. Fast unfolding of communities in large networks. *arxiv*, 2008. arXiv 2008.

Sebastian Böcker and Jan Baumbach. Cluster editing. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *The Nature of Computation. Logic, Algorithms, Applications*, pages 33–44, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39053-1.

Sebastian Böcker and Peter Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717 – 721, 2011. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/j.ipl.2011.05.003. URL `http://www.sciencedirect.com/science/article/pii/S0020019011001232`.

Ludvig Bohlin, Daniel Edler, Andrea Lancichinetti, and Martin Rosvall. Community detection and visualization of networks with the map equation framework. In *Measuring Scholarly Impact*, pages 3–34. Springer, 2014.

Marianna Bolla. Penalized versions of the newman-girvan modularity and their relation to normalized cuts and k-means clustering. *Phys. Rev. E*, 84:016108, Jul 2011. doi: 10.1103/PhysRevE.84.016108. URL `https://link.aps.org/doi/10.1103/PhysRevE.84.016108`.

Francesco Bonchi, Aristides Gionis, Francesco Gullo, Charalampos E. Tsourakakis, and Antti Ukkonen. Chromatic correlation clustering. *ACM Trans. Knowl. Discov. Data*, 9(4):34:1–34:24, June 2015. ISSN 1556-4681. doi: 10.1145/2728170. URL `http://doi.acm.org.ezproxy.lib.purdue.edu/10.1145/2728170`.

Flavia Bonomo, Guillermo Duran, Amedeo Napoli, and Mario Valencia-Pabon. A one-to-one correspondence between potential solutions of the cluster deletion problem and the minimum sum coloring problem, and its application to p4-sparse graphs. *Information Processing Letters*, 115: 600–603, 2015a.

Flavia Bonomo, Guillermo Duran, and Mario Valencia-Pabon. Complexity of the cluster deletion problem on subclasses of chordal graphs. *Theoretical Computer Science*, 600:59–69, 2015b.

Justin Brickell, Inderjit S. Dhillon, Suvrit Sra, and Joel A. Tropp. The metric nearness problem. *SIAM Journal on Matrix Analysis and Applications*, 30(1):375–396, 2008. doi: 10.1137/060653391. URL `https://doi.org/10.1137/060653391`.

Andrzej Cegielski. *Iterative methods for fixed point problems in Hilbert spaces*, volume 2057. Springer, 2012.

Yair Censor. Computational acceleration of projection algorithms for the linear best approximation problem. *Linear Algebra and its Applications*, 416(1):111–123, 2006.

Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360 – 383, 2005. ISSN 0022-0000. doi: http://dx.doi.org/10.1016/j.jcss.2004.10.012. URL `http://www.sciencedirect.com/science/article/pii/S0022000004001424`. Learning Theory 2003.

Moses Charikar, Neha Gupta, and Roy Schwartz. Local guarantees in graph cuts and clustering. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 136–147. Springer, 2017.

Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal lp rounding algorithm for correlationclustering on complete and complete k-partite graphs. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 219–228. ACM, 2015.

Flavio Chierichetti, Nilesh Dalvi, and Ravi Kumar. Correlation clustering in mapreduce. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 641–650. ACM, 2014.

Fan Chung. A local graph partitioning algorithm using heat kernel pagerank. In Konstantin Avrachenkov, Debora Donato, and Nelly Litvak, editors, *Algorithms and Models for the Web-Graph*, pages 62–75, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-540-95995-3.

Fan Chung and Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6(2):125–145, Nov 2002. ISSN 0219-3094. doi: 10.1007/PL00012580. URL `https://doi.org/10.1007/PL00012580`.

Fan R. K. Chung. *Spectral Graph Theory*, volume 92. American Mathematical Society, 1997. ISBN 978-1-4704-2452-7. doi: 10.1090/cbms/092.

Tom Coleman, James Saunderson, and Anthony Wirth. A local-search 2-approximation for 2-correlation-clustering. In Dan Halperin and Kurt Mehlhorn, editors, *Algorithms - ESA 2008*, pages 308–319, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-87744-8.

Peter Damaschke. *Bounded-Degree Techniques Accelerate Some Parameterized Graph Algorithms*, pages 98–109. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-11269-0. doi: 10.1007/978-3-642-11269-0_8. URL `http://dx.doi.org/10.1007/978-3-642-11269-0_8`.

Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011. ISSN 0098-3500. doi: 10.1145/2049662.2049663. URL `http://doi.acm.org/10.1145/2049662.2049663`.

Achiya Dax. On theory and practice of row relaxation methods. In Dan Butnariu, Yair Censor, and Simeon Reich, editors, *Inherently Parallel Algorithms in Feasibility and Optimization and their Applications*, volume 8 of *Studies in Computational Mathematics*, pages 153 – 186. Elsevier, 2001. doi: https://doi.org/10.1016/S1570-579X(01)80011-2. URL `http://www.sciencedirect.com/science/article/pii/S1570579X01800112`.

Achiya Dax. The adventures of a simple algorithm. *Linear Algebra and its Applications*, 361:41 – 61, 2003. ISSN 0024-3795. doi: 10.1016/S0024-3795(01)00600-0.

Wenceslas Fernandez de la Vega and Claire Kenyon-Mathieu. Linear programming relaxations of maxcut. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2007, pages 53–61, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5. URL `http://dl.acm.org/citation.cfm?id=1283383.1283390`.

J.-C. Delvenne, Sophia N Yaliraki, and Mauricio Barahona. Stability of graph communities across time scales. *Proceedings of the National Academy of Sciences*, 107(29):12755–12760, 2010.

Erik D. Demaine and Nicole Immorlica. *Correlation Clustering with Partial Information*, pages 1–13. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-45198-3.

Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2):172 – 187, 2006. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2006.05.008. URL `http://www.sciencedirect.com/science/article/pii/S0304397506003227`. Approximation and Online Algorithms.

Anders Dessmark, Jesper Jansson, Andrezej Lingas, Eva-Marta Lundell, and Mia Person. On the approximability of maximum and minimum edge clique partition problems. *International Journal of Foundations of Computer Science*, 18(02):217–226, 2007a.

Anders Dessmark, Jesper Jansson, Andrezej Lingas, Eva-Marta Lundell, and Mia Person. On the approximability of maximum and minimum edge clique partition problems. *International Journal of Foundations of Computer Science*, 18(02):217–226, 2007b.

Inderjit S Dhillon, Suvrit Sra, and Joel A Tropp. The metric nearness problems with applications. Technical report, The University of Texas at Austin, 2003.

Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11), 2007.

Thang N. Dinh, Xiang Li, and My T. Thai. Network clustering via maximizing modularity: Approximation algorithms and theoretical limits. In *Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM)*, ICDM '15, pages 101–110, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-9504-5. doi: 10.1109/ICDM.2015.139. URL `http://dx.doi.org/10.1109/ICDM.2015.139`.

Thang N Dinh, Xiang Li, and My T Thai. Network clustering via maximizing modularity: Approximation algorithms and theoretical limits. *arXiv*, 2016. arxiv Feb 2016.

Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Phys. Rev. E*, 72:027104, Aug 2005. doi: 10.1103/PhysRevE.72.027104. URL `https://link.aps.org/doi/10.1103/PhysRevE.72.027104`.

Richard L Dykstra. An algorithm for restricted least squares regression. *Journal of the American Statistical Association*, 78(384):837–842, 1983.

Dotan Emanuel and Amos Fiat. *Correlation Clustering – Minimizing Disagreements on Arbitrary Weighted Graphs*, pages 208–220. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-39658-1. doi: 10.1007/978-3-540-39658-1_21. URL `http://dx.doi.org/10.1007/978-3-540-39658-1_21`.

Julie Falkner, Franz Rendl, and Henry Wolkowicz. A computational study of graph partitioning. *Mathematical Programming*, 66(1):211–239, Aug 1994. ISSN 1436-4646. doi: 10.1007/BF01581147. URL `https://doi.org/10.1007/BF01581147`.

Chenglin Fan, Benjamin Raichel, and Gregory Van Buskirk. Metric violation distance: Hardness and approximation. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 196–209, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics. ISBN 978-1-6119-7503-1. URL `http://dl.acm.org/citation.cfm?id=3174304.3175280`.

Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75 – 174, 2010. ISSN 0370-1573. doi: https://doi.org/10.1016/j.physrep.2009.11.002. URL `http://www.sciencedirect.com/science/article/pii/S0370157309002841`.

Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007. ISSN 0027-8424. doi: 10.1073/pnas.0605965104. URL `https://www.pnas.org/content/104/1/36`.

Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1 – 44, 2016. ISSN 0370-1573. doi: https://doi.org/10.1016/j.physrep.2016.09.002. URL `http://www.sciencedirect.com/science/article/pii/S0370157316302964`. Community detection in networks: A user guide.

Takuro Fukunaga. Lp-based pivoting algorithm for higher-order correlation clustering. In Lusheng Wang and Daming Zhu, editors, *Computing and Combinatorics*, pages 51–62, Cham, 2018. Springer International Publishing. ISBN 978-3-319-94776-1.

Yong Gao, Donovan R Hare, and James Nastos. The cluster deletion problem for cographs. *Discrete Mathematics*, 313(23):2763–2771, 2013.

A. C. Gilbert and L. Jain. If it ain't broke, don't fix it: Sparse metric repair. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 612–619, Oct 2017. doi: 10.1109/ALLERTON.2017.8262793.

Anna C. Gilbert and Rishi Sonthalia. Generalized metric repair on graphs. *CoRR*, abs/1807.07619, 2018a. URL `http://arxiv.org/abs/1807.07619`.

Anna C. Gilbert and Rishi Sonthalia. Unrolling swiss cheese: Metric repair on manifolds with holes. *CoRR*, abs/1807.07610, 2018b. URL `http://arxiv.org/abs/1807.07610`.

Anna C. Gilbert and Rishi Sonthalia. Unrolling swiss cheese: Metric repair on manifolds with holes. *CoRR*, abs/1807.07610, 2018c. URL `http://arxiv.org/abs/1807.07610`.

Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 1167–1176, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. ISBN 0-89871-605-5. URL `http://dl.acm.org/citation.cfm?id=1109557.1109686`.

M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002. ISSN 0027-8424. doi: 10.1073/pnas.122653799. URL `https://www.pnas.org/content/99/12/7821`.

David F. Gleich, Nate Veldt, and Anthony Wirth. Correlation Clustering Generalized. In *29th International Symposium on Algorithms and Computation*, volume 123 of *ISAAC 2018*, pages 44:1–44:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-094-1. doi: 10.4230/LIPIcs.ISAAC.2018.44. URL `http://drops.dagstuhl.de/opus/volltexte/2018/9992`.

Pablo M. Gleiser and Leon Danon. Community structure in jazz. *Advances in Complex Systems*, 06(04):565–573, 2003. doi: 10.1142/S0219525903001067. URL `https://doi.org/10.1142/S0219525903001067`.

Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. *Graph-Modeled Data Clustering: Fixed-Parameter Algorithms for Clique Generation*, pages 108–119. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-44849-5. doi: 10.1007/3-540-44849-7_17. URL `http://dx.doi.org/10.1007/3-540-44849-7_17`.

Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, Jul 2005. ISSN 1433-0490. doi: 10.1007/s00224-004-1178-y. URL `https://doi.org/10.1007/s00224-004-1178-y`.

R. Guimerà, L. Danon, A. Díaz-Guilera, F. Giralt, and A. Arenas. Self-similar community structure in a network of human interactions. *Phys. Rev. E*, 68:065103, Dec 2003. doi: 10.1103/PhysRevE.68.065103. URL `https://link.aps.org/doi/10.1103/PhysRevE.68.065103`.

Roger Guimerà, Marta Sales-Pardo, and Luís A. Nunes Amaral. Modularity from fluctuations in random graphs and complex networks. *Phys. Rev. E*, 70:025101, Aug 2004. doi: 10.1103/PhysRevE.70.025101. URL `https://link.aps.org/doi/10.1103/PhysRevE.70.025101`.

Anshul Gupta. Fast and effective algorithms for graph partitioning and sparse-matrix ordering. *IBM J. Res. Dev.*, 41(1-2):171–183, January 1997. ISSN 0018-8646. doi: 10.1147/rd.411.0171. URL `http://dx.doi.org/10.1147/rd.411.0171`.

Benjamin Halpern. Fixed points of nonexpanding maps. *Bulletin of the American Mathematical Society*, 73(6):957–961, 1967.

Yves Haugazeau. *Sur les inéquations variationnelles et la minimisation de fonctionnelles convexes*. PhD thesis, Universite de Paris, 1968.

Clifford Hildreth. A quadratic programming procedure. *Naval Research Logistics (NRL)*, 4(1): 79–85, 1957.

Jack P. Hou, Amin Emad, Gregory J. Puleo, Jian Ma, and Olgica Milenkovic. A new correlation clustering method for cancer mutation analysis. *Bioinformatics*, 32(24):3717–3728, 2016. doi: 10.1093/bioinformatics/btw546. URL `http://dx.doi.org/10.1093/bioinformatics/btw546`.

Alfredo N Iusem and Alvaro R De Pierro. On the convergence of Han's method for convex programming with quadratic objective. *Mathematical Programming*, 52(1-3):265–284, 1991.

Abigail Z. Jacobs, Samuel F. Way, Johan Ugander, and Aaron Clauset. Assembling thefacebook: Using heterogeneity to understand online social network assembly. In *Proceedings of the ACM Web Science Conference*, WebSci '15, pages 18:1–18:10, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3672-7. doi: 10.1145/2786451.2786477. URL `http://doi.acm.org/10.1145/2786451.2786477`.

Lucas G. S. Jeub, Marya Bazzi, Inderjit S. Jutla, and Peter J. Mucha. A generalized louvain method for community detection implemented in MATLAB, 2011-2017. URL `http://netwiki.amath.unc.edu/GenLouvain`.

Lucas GS Jeub, Olaf Sporns, and Santo Fortunato. Multiresolution consensus clustering in networks. *Scientific reports*, 8(1):3259, 2018. doi: 10.1038/s41598-018-21352-7.

Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 767–775, New York, NY, USA, 2002. ACM. ISBN 1-58113-495-9. doi: 10.1145/509907.510017. URL `http://doi.acm.org/10.1145/509907.510017`.

Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang D. Yoo. Higher-order correlation clustering for image segmentation. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1530–1538. Curran Associates, Inc., 2011. URL `http://papers.nips.cc/paper/4406-higher-order-correlation-clustering-for-image-segmentation.pdf`.

Isabel M. Kloumann, Johan Ugander, and Jon Kleinberg. Block models and personalized pagerank. *Proceedings of the National Academy of Sciences*, 114(1):33–38, 2017. ISSN 0027-8424. doi: 10.1073/pnas.1611275114. URL `https://www.pnas.org/content/114/1/33`.

Donald Ervin Knuth. *The Stanford GraphBase: a platform for combinatorial computing.* ACM Press New York, 1993.

V. Krebs. Books about US Politics, 2004. URL `http://networkdata.ics.uci.edu/data.php?d=polbooks`. Hosted at UCI Data Repository.

Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E*, 80:016118, Jul 2009. doi: 10.1103/PhysRevE.80.016118. URL `https://link.aps.org/doi/10.1103/PhysRevE.80.016118`.

Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical review E*, 78(4):046110, 2008.

Kevin Lang and Satish Rao. Finding near-optimal cuts: an empirical evaluation. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA 1993, pages 212–221. Society for Industrial and Applied Mathematics, 1993.

Kevin Lang and Satish Rao. A flow-based method for improving the expansion or conductance of graph cuts. In *Integer Programming and Combinatorial Optimization*, volume 3064 of *Lecture Notes in Computer Science*, pages 325–337. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-540-25960-2_25.

Kevin J Lang, Michael W Mahoney, and Lorenzo Orecchia. Empirical evaluation of graph partitioning using spectral embeddings and flow. In *International Symposium on Experimental Algorithms*, SEA 2009, pages 197–208. Springer, 2009.

Jan-Hendrik Lange, Andreas Karrenbauer, and Bjoern Andres. Partial optimality and fast lower bounds for weighted correlation clustering. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2898–2907, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/lange18a.html`.

Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)*, 46(6):787–832, November 1999.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007. ISSN 1556-4681. doi: 10.1145/1217299.1217301. URL `http://doi.acm.org/10.1145/1217299.1217301`.

Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 695–704, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi: 10.1145/1367497.1367591. URL `http://doi.acm.org/10.1145/1367497.1367591`.

P. Li, H. Dau, G. Puleo, and O. Milenkovic. Motif clustering and overlapping clustering for social network analysis. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, May 2017. doi: 10.1109/INFOCOM.2017.8056956.

Pan Li, Gregory J. Puleo, and Olgica Milenkovic. Motif and hypergraph correlation clustering. *CoRR*, abs/1811.02089, 2018. URL `http://arxiv.org/abs/1811.02089`.

P. L. Lions. Approximation de points fixes de contractions. *C. R. Acad. Sci. Ser. A-B Paris*, 284: 1357–1359, 1977.

Guimei Liu and Limsoon Wong. *Effective Pruning Techniques for Mining Quasi-Cliques*, pages 33–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN "978-3-540-87481-2". URL `"https://doi.org/10.1007/978-3-540-87481-2_3"`.

David Lusseau, Karsten Schneider, Oliver J. Boisseau, Patti Haase, Elisabeth Slooten, and Steve M. Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, Sep 2003. ISSN 1432-0762. doi: 10.1007/s00265-003-0651-y. URL `https://doi.org/10.1007/s00265-003-0651-y`.

O. L. Mangasarian. Normal solutions of linear programs. *Mathematical Programming at Oberwolfach II*, pages 206–216, 1984.

Atsushi Miyauchi, Tomohiro Sonobe, and Noriyoshi Sukegawa. Exact clustering via integer programming and maximum satisfiability. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1387–1394, 2018. URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16176`.

C. Moler. *Numerical Computing with Matlab*. Society for Industrial and Applied Mathematics, 2004. doi: 10.1137/1.9780898717952. URL `https://epubs.siam.org/doi/abs/10.1137/1.9780898717952`.

Assaf Natanzon. Complexity and approximation of some graph modification problems. Master's thesis, Tel Aviv University, 1999.

Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 65–77. Springer, 1999.

M. E. J. Newman. Community detection and graph partitioning. *EPL (Europhysics Letters)*, 103 (2):28003, 2013. URL `http://stacks.iop.org/0295-5075/103/i=2/a=28003`.

M. E. J. Newman. Equivalence between modularity optimization and maximum likelihood methods for community detection. *Phys. Rev. E*, 94:052315, Nov 2016. doi: 10.1103/PhysRevE.94.052315. URL `https://link.aps.org/doi/10.1103/PhysRevE.94.052315`.

Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(026113), 2004.

Sebastian Nowozin and Stefanie Jegelka. Solution stability in linear programming relaxations: Graph partitioning and unsupervised learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 769–776, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553473. URL `http://doi.acm.org/10.1145/1553374.1553473`.

Lorenzo Orecchia and Zeyuan Allen Zhu. Flow-based algorithms for local graph clustering. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms*, SODA 2014, pages 1267–1286, 2014. URL `http://arxiv.org/abs/1307.2855`.

Xinghao Pan, Dimitris Papailiopoulos, Samet Oymak, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Parallel correlation clustering on big graphs. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 82–90. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5814-parallel-correlation-clustering-on-big-graphs.pdf`.

Leto Peel, Daniel B. Larremore, and Aaron Clauset. The ground truth about metadata and community detection in networks. *Science Advances*, 3(5), 2017. doi: 10.1126/sciadv.1602548. URL `http://advances.sciencemag.org/content/3/5/e1602548`.

Svatopluk Poljak and Zsolt Tuza. The expected relative error of the polyhedral approximation of the max-cut problem. *Operations Research Letters*, 16(4):191–198, November 1994. ISSN 0167-6377. doi: 10.1016/0167-6377(94)90068-X.

Mason A Porter, Jukka-Pekka Onnela, and Peter J Mucha. Communities in networks. *Notices of the AMS*, 56(9):1082–1097, 2009.

G. Puleo and O. Milenkovic. Correlation clustering with constrained cluster sizes and extended weights bounds. *SIAM Journal on Optimization*, 25(3):1857–1872, 2015. doi: 10.1137/140994198. URL `https://doi.org/10.1137/140994198`.

G. J. Puleo and O. Milenkovic. Correlation clustering and biclustering with locally bounded errors. *IEEE Transactions on Information Theory*, 64(6):4105–4119, June 2018. ISSN 0018-9448. doi: 10.1109/TIT.2018.2819696.

Jörg Reichardt and Stefan Bornholdt. Detecting fuzzy community structures in complex networks with a potts model. *Phys. Rev. Lett.*, 93:218701, Nov 2004. doi: 10.1103/PhysRevLett.93.218701. URL `https://link.aps.org/doi/10.1103/PhysRevLett.93.218701`.

Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74(016110), 2006.

O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 3–13, Nov 2000. doi: 10.1109/SFCS.2000.892006.

Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. *Electronic Colloquium on Computational Complexity (ECCC)*, 8(18), 2001. URL `http://eccc.hpi-web.de/eccc-reports/2001/TR01-018/index.html`.

Ryan A. Rossi, David F. Gleich, and Assefaw H. Gebremedhin. Parallel maximum clique algorithms with applications to network analysis. *SIAM Journal on Scientific Computing*, 37(5):C589–C616, 2015. doi: 10.1137/14100018X.

Martin Rosvall, Daniel Axelsson, and Carl T Bergstrom. The map equation. *The European Physical Journal-Special Topics*, 178(1):13–23, 2009.

Cameron Ruggles, Nate Veldt, and David Gleich. A parallel projection method for metric constrained optimization. *arXiv preprint arXiv:1901.10084*, 2019.

Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27 – 64, 2007. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2007.05.001. URL `http://www.sciencedirect.com/science/article/pii/S1574013707000020`.

Michael T. Schaub, Renaud Lambiotte, and Mauricio Barahona. Encoding dynamics for multiscale community detection: Markov time sweeping for the map equation. *Phys. Rev. E*, 86:026112, Aug 2012. doi: 10.1103/PhysRevE.86.026112. URL `https://link.aps.org/doi/10.1103/PhysRevE.86.026112`.

Kirk Schloegel, George Karypis, and Vipin Kumar. Graph partitioning for high-performance scientific simulations. In Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, editors, *Sourcebook of Parallel Computing*, pages 491–541. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. ISBN 1-55860-871-0. URL `http://dl.acm.org/citation.cfm?id=941480.941499`.

Christian Schulz. *High Quality Graph Partitioning*. PhD thesis, Karlsruhe Institute of Technology, May 2013.

Christian Schulz, Sebastian Korbinian Bayer, Christoph Hess, Christian Steiger, Marvin Teichmann, Jan Jacob, Fellipe Bernardes-lima, Robert Hangu, and Sergey Hayrapetyan. Course notes: Graph partitioning and graph clustering in theory and practice. Lecture notes at Institute for Theoretical Informatics Karlsruhe Institute of Technology (KIT), May 2016.

C. Seshadhri, Tamara G. Kolda, and Ali Pinar. Community structure and scale-free collections of Erdős-Rényi graphs. *Physical Review E*, 85(5):056109, May 2012. doi: 10.1103/PhysRevE.85.056109.

Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144:173–182, 2004.

Priyanka Sharma and Manoj Singh. Multi chromatic balls with relaxed criterion to detect larger communities in social networks. In Aynur Unal, Malaya Nayak, Durgesh Kumar Mishra, Dharm Singh, and Amit Joshi, editors, *Smart Trends in Information Technology and Computer Communications*, pages 196–203, Singapore, 2016. Springer Singapore. ISBN 978-981-10-3433-6.

Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, August 2000. doi: 10.1109/34.868688.

Suvrit Sra, Joel Tropp, and Inderjit S. Dhillon. Triangle fixing algorithms for the metric nearness problem. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 361–368. MIT Press, 2005. URL `http://papers.nips.cc/paper/2598-triangle-fixing-algorithms-for-the-metric-nearness-problem.pdf`.

Chaitanya Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 526–527. Society for Industrial and Applied Mathematics, 2004.

Paul Swoboda and Bjoern Andres. A message passing algorithm for the minimum cost multicut problem. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4990–4999, July 2017. doi: 10.1109/CVPR.2017.530.

Amanda L. Traud, Peter J. Mucha, and Mason A. Porter. Social structure of Facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012. doi: 10.1016/j.physa.2012.03.0. URL `https://ideas.repec.org/a/eee/phsmap/v391y2012i16p4165-4180.html`.

Luca Trevisan. Lecture notes on expansion, sparsest cut, and spectral graph theory, 2013.

Stijn Marinus Van Dongen. *Graph clustering by flow simulation.* PhD thesis, Utrecht University, 2000.

Jurgen Van Gael and Xiaojin Zhu. Correlation clustering for crosslingual link detection. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI 2007, pages 1744–1749, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1625275.1625558`.

Anke van Zuylen and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Mathematics of Operations Research*, 34(3):594–620, 2009. doi: 10.1287/moor.1090.0385. URL `https://doi.org/10.1287/moor.1090.0385`.

Nate Veldt, David Gleich, and Michael Mahoney. A simple and strongly-local flow-based method for cut improvement. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *ICML 2016*, pages 1938–1947, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `http://proceedings.mlr.press/v48/veldt16.html`.

Nate Veldt, Anthony I. Wirth, and David F. Gleich. Correlation clustering with low-rank matrices. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1025–1034, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4913-0. doi: 10.1145/3038912.3052586. URL `https://doi.org/10.1145/3038912.3052586`.

Nate Veldt, David Gleich, Anthony Wirth, and James Saunderson. A projection method for metric-constrained optimization. *arXiv preprint arXiv:1806.01678*, 2018a.

Nate Veldt, David F. Gleich, and Anthony Wirth. A correlation clustering framework for community detection. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 439–448, Republic and Canton of Geneva, Switzerland, 2018b. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-5639-8. doi: 10.1145/3178876.3186110. URL `https://doi.org/10.1145/3178876.3186110`.

Nate Veldt, David Gleich, Anthony Wirth, and James Saunderson. Metric-constrained optimization for graph clustering algorithms. *SIAM Journal on Mathematics of Data Science (to appear)*, 2019a.

Nate Veldt, David F. Gleich, and Anthony Wirth. Learning resolution parameters for graph clustering. In *Proceedings of the 28th International Conference on World Wide Web*, WWW '19, Republic and Canton of Geneva, Switzerland, 2019b. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-6674-8/19/05. doi: 10.1145/3308558.3313471. URL `https://doi.org/10.1145/3308558.3313471`.

Nate Veldt, Christine Klymko, and David F. Gleich. Flow-based local graph clustering with better seed set inclusion. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, 2019c.

Di Wang, Kimon Fountoulakis, Monika Henzinger, Michael W. Mahoney, and Satish Rao. Capacity releasing diffusion for speed and locality. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3598–3607, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL `http://proceedings.mlr.press/v70/wang17b.html`.

Yashen Wang, Heyan Huang, Chong Feng, and Zhirun Liu. Community detection based on minimum-cut graph partitioning. In Xin Luna Dong, Xiaohui Yu, Jian Li, and Yizhou Sun, editors, *Web-Age Information Management*, pages 57–69, Cham, 2015. Springer International Publishing. ISBN 978-3-319-21042-1.

Yubo Wang, Linli Xu, Yucheng Chen, and Hao Wang. A scalable approach for general correlation clustering. In Hiroshi Motoda, Zhaohui Wu, Longbing Cao, Osmar Zaiane, Min Yao, and Wei Wang, editors, *Advanced Data Mining and Applications*, pages 13–24, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-53917-6.

Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440 EP –, 06 1998. URL `https://doi.org/10.1038/30918`.

John Graham White, Eileen Southgate, J.N. Thomson, and Sydney Brenner. The structure of the nervous system of the nematode caenorhabditis elegans. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 314, 1986. URL `https://doi.org/10.1098/rstb.1986.0056`.

Anthony Ian Wirth. *Approximation Algorithms for Clustering.* PhD thesis, Princeton University, November 2004. Computer Science Technical Report 716.

Rainer Wittmann. Approximation of fixed points of nonexpansive mappings. *Archiv der Mathematik*, 58(5):486–491, 1992.

Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, Jan 2015. ISSN 0219-3116. doi: 10.1007/s10115-013-0693-z. URL `https://doi.org/10.1007/s10115-013-0693-z`.

Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 2017, pages 555–564, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3098069. URL `http://doi.acm.org/10.1145/3097983.3098069`.

Linbin Yu and Chris Ding. Network community discovery: Solving modularity clustering via normalized cut. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, MLG '10, pages 34–36, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0214-2. doi: 10.1145/1830252.1830257. URL `http://doi.acm.org/10.1145/1830252.1830257`.

Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

HongFang Zhou, Jin Li, JunHuai Li, FaCun Zhang, and YingAn Cui. A graph clustering method for community detection in complex networks. *Physica A: Statistical Mechanics and its Applications*, 469:551 – 562, 2017. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2016.11.015. URL `http://www.sciencedirect.com/science/article/pii/S0378437116308202`.