HIERARCHICAL CLUSTERING WITH GLOBAL OBJECTIVES: APPROXIMATION ALGORITHMS AND HARDNESS RESULTS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Evangelos Chatziafratis
June 2020

Abstract

Hierarchical Clustering is an important tool for data analysis, used in diverse areas ranging from Phylogenetics in Biology to YouTube video recommendations and everything in between. The term “hierarchical” is used to contrast with “flat” clustering approaches, like k-means or k-center, which only produce one specific partitioning of a data set. Hierarchical Clustering is a way of understanding the overall structure of large data sets by visualizing them as a collection of successively smaller and smaller clusters, capturing different levels of granularity simultaneously. The end result is a tree, whose internal nodes represent nested clusters and with leaves representing the data points. Despite the plethora of applications and heuristics, the theory behind Hierarchical Clustering was underdeveloped, since no “global” objective function was associated with the final output. A well-formulated objective allows us to compare different algorithms, measure their quality, explain their success or failure and enables continuous optimization methods with gradient descent. This lack of objectives is in stark contrast with the flat clustering literature, where objectives like k-means, k-median or k-center have been studied intensively starting from the 1950s, leading to a comprehensive theory on clustering.

In this thesis, we study approximation algorithms and their limitations, making progress towards building such a theory for objective-based Hierarchical Clustering (HC). For the minimization cost function with pairwise similarities introduced recently by Dasgupta (2016) and its maximization variants, we show how standard tools like Sparsest Cut, Balanced Cut, Max Cut or Max Uncut Bisection can be used in a black box manner, to produce provably good HC and we also complement our algorithms with inapproximability results. Escaping from worst-case analyses, we also study scenarios where extra structure is imposed on the data points (e.g., feature vectors) or qualitative information is provided (e.g., the common *triplet* or *quartet* constraints in Phylogenetics). Our results are inspired by geometric ideas (semidefinite programming relaxations, *spreading* metrics, random projections) and novel connections with *ordering* Constraint Satisfaction Problems.

Overall, this thesis provides state-of-the-art approximation and hardness results for optimizing global HC objectives, reveals unexpected connections with common linkage or graph cut heuristics and advances our understanding of old Phylogenetics problems. As such, we hope it serves the community as the foundation for objective-based Hierarchical Clustering and for future improvements.

Acknowledgments

As I am finishing my thesis from an apartment in New York, I can't help but thinking how lucky I have been so far, especially throughout those 5 years of my PhD. Featuring a single name on the front cover is merely an understatement, as both this thesis and myself would not be the same without the tremendous help and constant support of several amazing individuals, I had the extreme pleasure to meet along my journey in research and in the academic world. Given that no physical PhD defense ever took place at Stanford, I want to take this opportunity to express my gratitude to the people that helped me during this beautiful journey and, more importantly, made it possible.

First and foremost, I can't thank enough my advisor Tim Roughgarden for all he has done for me over these past 5 years. From my second quarter at Stanford, when I first rotated and worked with Tim, to our productive research trip in London School of Economics during his sabbatical, to our path towards New York in Columbia and my next career steps, Tim has always been extremely supportive and patient of my diverse research pursuits, always ready to provide me with his insightful ideas and advice. I owe him so much for his mentorship and ideas, for inspiring me, for teaching me cool stuff and helping me improve my communication skills; and all these, done in the casual environment he created for his group (lunches at Ike's, dinners at NOLA etc.), which allowed me to thrive without ever feeling I was working, rather enjoying research and theory. Outside research, I also want to thank Tim, simply for being lenient with me, accepting my longer-than-usual trips to Greece, without asking questions whether they happened during (early) summer, Christmas, or even March. I also consider myself privileged to have been his unique student simultaneously at Stanford and Columbia, and to have helped with the summer rental car situation in Ikaria and with the cameras and amplifiers during Papafest's talks and band concert.

The second person that I want to thank is Moses Charikar. During the spring quarter of my first year, we started exploring hierarchical clustering, a term I had never heard about back then and, unknowingly, the main chapter in this thesis had already started. Collaborating with Moses and trying to make sense of the variety of clustering questions that appeared on our way over time, has truly been an immense pleasure. His intuition and clarity of thought together with his patience on explaining to me possible ways of attacking the problems have been extremely beneficial. I am also grateful to him for being available for late night Skypes in several occasions and for interrupting his dinner plans just to answer my texts, admittedly right before the submission deadlines, and finally for battling with the different timezones when I was in Switzerland or in Sweden and he was in India.

I also want to thank Jan Vondrak, not only for our collaboration, but also for being a friendly and approachable person in the Math department, that I always felt comfortable talking to and I could always go with questions and have interesting conversations across many different topics. Of course, I am glad he also agreed to be the Chair of my thesis committee. I want to thank Greg Valiant for being in my orals committee, for letting me TA several of his courses and for his support during summer 2017. Also, for his direct and casual character that always makes you feel comfortable. I also owe a big thanks to Li-Yang Tan for agreeing to be in my quals and my orals committee and for sharing much of his love and expertise in complexity theory.

I also want to thank several professors I had the pleasure to work with for a short period: Virginia Vassilevska Williams for an awesome first rotation, and for her great teaching style, Nisheeth Vishnoi for an inspiring summer internship in EPFL during summer 2016, which taught me a lot and Osman Yagan for several visits to CMU in Silicon Valley and in Pittsburgh and for interesting discussions on network robustness. Outside research, I want to thank Omer Reingold for an amazing class on Playback theater and Kumaran Arul for being an excellent and inspiring piano tutor. Watching Ryan Williams playing guitar, singing and improvising during our Theory Music Tuesday Nights has also been quite the experience!

A big role for the development of this thesis was played by my amazing collaborators. First of all, I want to thank Rad Niazadeh for the countless hours we spent drawing trees and clusters on the whiteboard, discussing about how to round SDPs, late-night editing of papers before submissions, and also for giving me career advice. I want to thank Mohammad Mahdian for being an amazing host at Google New York during my summer and winter internships, for the always stimulating conversations that may happen at a microkitchen or while searching for available rooms, over video calls, or even while on vacation in Rhodes, Greece. I want to thank Ioannis Panageas for hosting me in Singapore, for his direct and fun character and for his mathematical depth that helped me explore and appreciate the many interesting connections between deep learning and dynamical systems. Moreover and in no particular order, I also had the pleasure to work with: Neha Gupta, Euiwoong Lee, Sara Ahmadian, Alessandro Epasto, Grigory Yaroslavtsev, Kontantin Makarychev, Sai Ganesh Nagarajan, Xiao Wang, Haris Angelidakis, Avrim Blum, Chen Dan, Pranjal Awasthi, Xue Chen, Aravindan Vijayaraghavan, Osman Yagan, and Joshua Wang. I also want to thank Okke Schrijvers and his team at Facebook for hosting me over the summer of 2018: Julián Mestre, Nicolas Stier-Moses and Birce Tezel.

My time at Stanford and New York would not be as fun and carefree as it has been, without the necessary balance that came from my friends. First, I would like to thank Mary Ann and Bob Poulos and their amazing family Olivia, Lindsey and Lauren that took care of me, for their support and their hospitality. I want to thank my friends in Hellas (Stanford's Greek community) for all the nice events we organized over Easter, Christmas and summer and for all the help and advice they provided during these years: our statistician Nikos Ignatiadis, our tamias (treasurer) Kostis Kaffes and vice-president Faidra Monachou, our doctor Stelios Serghiou and biologist Eirini Tsekitsidou, το (the) Peter Zachares, our petroleum experts Dimitris Belivanis and Philippos Kostakis, our basketball player George Korakitis, our minaras Marios Galanis from Patras, my mentor Yorgos Katsikis and our life-coach Vasilis Verroios. See you in Cholargos, at President's souvlaki or Shisan sushi ;) Also, the other Greeks from my incoming year George Alexopoulos, George Chrysochoos, Georgios Laskaris, Iakovos Anagnostopoulos that helped me adapt to the new environment and Alexios Voulimeneas for the Halloween Sparta costumes. I would also like to thank my friends in New York: Ilias Zadik, Orestis Plevrakis, Themis Melissaris, Lampros Flokas, Manolis Vlatakis, Giannis Karamanolakis, Thodoris Lykouris, and Fotis Iliopoulos for hosting me when in need, for giving me valuable presentation feedback and laughs at online Codenames, and for interesting research discussions.

My time in the theory groups at Stanford and Columbia would not be as fun and memorable, if it weren't for the many board games, theory retreats, research talks, food and concerns shared with my fellow grad students: in no particular order thank you to Amir Abboud, Kostas Kollias, Leo Keselman, Josh Alman, Brian Axelrod, Greg Bodwin, Saba Eskadarian, Shivam Garg, Paris Syminelakis, Nicole Wein, Hilal Asi, Sam Kim, Ofir Geri, Neha Gupta, Reyna Hulett, Ray Li, Noah Shutty, Michael Kim, Andrea Lincoln, Annie Mardsen, Tony Kim, Jimmy Wu, Bruce Spang, Kevin Tian, Warut Suksompong, Vasilis Gkatzelis, Weihao Kong, Okke Schrijvers, Dylan McKay, Aditi Raghunathan, Henry Corrigan-Gibbs, Dima Kogan, Yeganeh Alimohammadi, Benedikt Bünz, Florian Tramer, Josh Wang, Vatsal Sharan, Kiran Shiragur, Shashwat Silas, Hongyang Zhang, and Clayton Sanford. I owe huge thanks to Megan Harris, Ruth Harris, Jay Subramanian and Jam Kiattinant for helping me in anything I needed and always being there for the students. Also, I owe thanks to Katerina Magkel and the Onassis Foundation for supporting in part my studies during the 2019-2020 academic year.

Finally, I want to thank my professors Stathis Zachos, Dimitris Fotakis and Aris Pagourtzis for their excellent work, for inspiring so many students during my undergraduate studies in NTUA and for making us love and eventually follow the path towards Algorithms and Theory. Also, thanks to all my friends in EPFL and ETH, Prodromos Kolyvakis, Georgios Damaskinos, Lefteris Kokoris-kogias and Harris Angelidakis for making my stay in Switzerland so easy and smooth, my friends in Athens, George Banis, Nikos Mavroulis, Alexandros Banis, Manos Sofoulis, Marios Fouzas, Dennis Triantafillis II, Steve Mitropoulos, Nasos Georgiou, Alexia Athanasiou, Alexis Markazos, and Alexandros Georgopoulos for their endless supply of interesting conversations, funny stories, memes, discussions and memorable vacations.

I want to thank my girlfriend Yuuki Miura for being an amazing companion, her hospitality during Big 'Rona and her inspiring kindness and good character. Last but not least, I can't thank enough my family for everything they have done for me and continue to do, for everything they have taught me and continue to teach me and for their endless love and support for whatever I did and continue to do.

Dedicated to my first teachers, my family
Chryssa, Anna-Maria, Telemachos, Andreas, Stratos

Το πρώτο σκαλί

Εις τον Θεόκριτο παραπονιούνταν
μιά μέρα ο νέος ποιητής Ευμένης·
≪Τώρα δυό χρόνια πέρασαν που γράφω
κ' ένα ειδύλιο έκαμα μονάχα.
Το μόνον άρτιόν μου έργον είναι.
Αλλοίμονον, είν' υψηλή το βλέπω,
πολύ υψηλή της Ποιήσεως η σκάλα·
και απ' το σκαλί το πρώτο εδώ που είμαι
ποτέ δεν θ' αναιβώ ο δυστυχισμένος≫.
Ειπ' ο Θεόκριτος· ≪Αυτά τα λόγια
ανάρμοστα και βλασφημίες είναι.
Κι αν είσαι στο σκαλί το πρώτο, πρέπει
νάσαι υπερήφανος κ' ευτυχισμένος.
Εδώ που έφθασες, λίγο δεν είναι·
τόσο που έκαμες, μεγάλη δόξα.
Κι αυτό ακόμη το σκαλί το πρώτο
πολύ από τον κοινό τον κόσμο απέχει.
Εις το σκαλί για να πατήσεις τούτο
πρέπει με το δικαίωμά σου νάσαι
πολίτης εις των ιδεών την πόλι.
Και δύσκολο στην πόλι εκείνην είναι
και σπάνιο να σε πολιτογραφήσουν.
Στην αγορά της βρίσκεις Νομοθέτας
που δεν γελά κανένας τυχοδιώκτης.
Εδώ που έφθασες, λίγο δεν είναι·
τόσο που έκαμες, μεγάλη δόξα≫.

Κωνσταντίνος Π. Καβάφης

The First Step

The young poet Evmenis
complained one day to Theocritus:
"I've been writing for two years now
and I've composed only one idyll.
It's my single completed work.
I see, sadly, that the ladder
of Poetry is tall, extremely tall;
and from this first step I'm standing on now
I'll never climb any higher."
Theocritus retorted: "Words like that
are improper, blasphemous.
Just to be on the first step
should make you happy and proud.
To have reached this point is no small deed:
what you've done already is wonderful.
Even this first step
is a long way above the ordinary world.
To stand on this step
you must be in your own right
a member of the city of ideas.
And it's a hard, unusual thing
to be enrolled as a citizen of that city.
Its councils are full of Legislators
no charlatan can fool.
To have reached this point is no small deed:
what you've done already is wonderful."

Constantine P. Cavafy

This thesis is based on previously published and ongoing works:

1. Chapter 4 is based on the following three papers:
   - "Approximate Hierarchical Clustering via Sparsest Cut and Spreading Metrics" from SODA 2017 [27]
   - "Hierarchical Clustering better than Average-Linkage" from SODA 2019 [28]
   - "Bisect and Conquer: Hierarchical Clustering via Max-Uncut Bisection" from AISTATS 2020 [2].
2. Chapter 5 is based on "Hierarchical Clustering for Euclidean Data" from AISTATS 2019 [29].
3. Chapter 6 is based on the following two papers:
   - "Hierarchical Clustering with Structural Constraints" from ICML 2018 [35]
   - "Aggregating Inconsistent Information in Ranking, Clustering and Phylogenetic Trees", manuscript under submission, May 2020 [34].

Contents

List of Tables

List of Figures

Chapter 1

Introduction

I have this *nightmare*: that scientific historians of the future will say that during the 20th century, there were these sixty very intellectually exciting years during which optimization was *not* just gradient descent.

-Christos Papadimitriou (2019)

This thesis is all about Hierarchical Clustering which, in some sense, is just a fancier way of referring to what computer scientists are taught to call simply, a "tree". Thinking about trees has been a mathematician's recreation for many centuries. In 1857, the great British mathematician Arthur Cayley published an article "On the theory of the analytical forms called trees" and later in 1889 another one "A theorem on trees", where he gave formulas for counting different types of trees [25, 26]. Such arguments are considered standard by now and typically, in undergraduate algorithms, a student learns about binary trees and some of their amazing powers for speeding up basic algorithmic problems. Here, we view trees as a way of understanding data sets and specifically relationships among data points.

Hierarchical Clustering has become by now an essential tool for data analysis useful across diverse areas, ranging from Phylogenetics in Biology to YouTube video recommendations and everything in between. It originated in Biology, where researchers wanted to understand the evolution of species, their genetic similarities, possible speciation events throughout history and build a taxonomy on extinct and extant organisms, usually called the Tree of Life (see Figure 1.1). A major problem then quickly appeared on how to aggregate vast amounts of information and automatically extract groups of organisms that are similar together (e.g., types of different bacteria, birds or mammals etc.).

In computer science, the problem of partitioning a given data set into multiple groups is called clustering and is well-studied; chances are, that any conference on theory or applications in computer

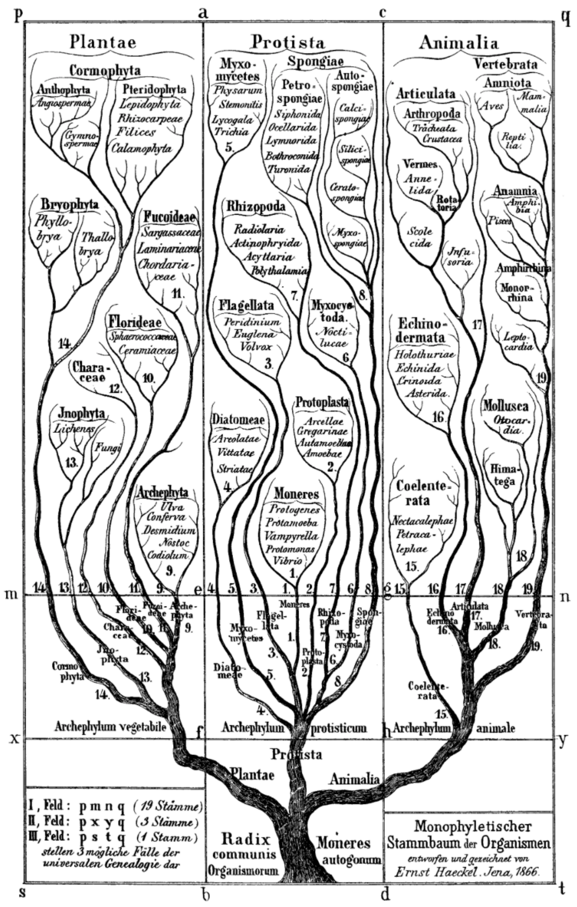
Plantae
Protista
Animalia
Archephylum vegetabile
Archephylum protisticum
Archephylum animale
Radix communis Organismorum
Moneres autogonum
Monophyletischer Stammbaum der Organismen entworfen und gezeichnet von Ernst Haeckel. Jena, 1866.

Figure 1.1: In biology, understanding the origins and relations of species relies on using Hierarchical Clustering.

science features several papers on clustering. However, in the above example from Taxonomy, it is easy to see that since an organism can belong simultaneously to multiple clusters at different levels of granularity (e.g., a cat belongs to the family of "Felidae" which is subclass of the class "Mammals"), standard "flat" clustering is no longer the right tool, as what we want is a hierarchy of clusters to capture the naturally hierarchical structure of the evolution of species.

As a result, we use the term "hierarchical" (etymology comes from the greek word < hierarkhia meaning "sacred rules") to contrast with "flat" clustering approaches, which only produce one specific partitioning of a data set. Hierarchical Clustering is a way of understanding the overall structure of large data sets by conveniently visualizing them as a collection of successively smaller and smaller clusters, capturing different levels of granularity simultaneously. The end result is a tree, whose internal nodes represent nested clusters and where data points lie at its lowest levels, with dissimilar data points being split towards the top of the tree and with similar data points being split towards the bottom of the tree. To construct this tree, there are intuitive and easy to implement bottom-up merging methods or top-down divisive procedures.

Once you are aware of it, you quickly realize that Hierarchical Clustering is everywhere as data sets are of hierarchical nature. When analyzing social networks, solving community detection, organizing and recommending movies, retrieving relevant tweets and doing text analysis, suggesting ads in online platforms, investigating gene relations to cancer, or investing money on large portfolios, at some point hierarchies on data will be required and Hierarchical Clustering sneaks in.

But it's not all good news, as there was something unsatisfactory about Hierarchical Clustering. Despite the plethora of applications and heuristics, the theory behind it was underdeveloped, partly because no "global" objective function, measuring the quality of the final output tree, was studied. A well-formulated objective allows us to compare different algorithms, measure their quality, explain their success or failure and enables continuous optimization methods with gradient descent. This lack of objectives is in stark contrast with the flat clustering literature, where objectives like k-means, k-median or k-center have been studied intensively leading to a comprehensive theory on clustering.

However, as is a common curse in computer science, most problems are hard to optimize, and hierarchical clustering is no exception. Thus, we need to turn our attention to approximation algorithms. Such algorithms try to find good solutions close to the optimum and usually they are accompanied by a proof for their performance.

This thesis, by studying approximation algorithms and their limitations, makes progress towards building a theory for objective-based Hierarchical Clustering (HC from now as the name is simply too long). In a nutshell, We study both minimization and maximization objectives for HC, we derive algorithms and their limitations based on standard graph partitioning problems.

Main Results

Before continuing with the basic definitions for HC, we want to quickly mention our main findings. We present 4 main results: For the minimization cost function with pairwise similarities introduced recently by Dasgupta [43], we show how using Sparsest Cut or Balanced Cut primitives as a black box, produces a HC with no extra loss in approximation. Our result was inspired by a semidefinite programming relaxation (SDP) we originally formulated for HC based on so-called *spreading* metrics. To complement the upper bounds, we also prove that no efficient algorithm can achieve constant factor approximations. This work appeared in SODA 2017 [27].

For two maximization variants of Dasgupta's HC cost, we give the first tight analysis for the performance of Average-Linkage which is an important algorithm originally proposed in the context of Biology. We show how to achieve better approximations using our aforementioned SDP together with geometric ideas from convex analysis. In a follow-up work, where we study one of the maximization variants, we show how using Max-Uncut Bisection as a black box, produces a solution within 42% of optimum and we complement this with hardness results showing there is inherent limitations on achieving approximations close to optimal. These works appeared in SODA 2019 [28] and in AISTATS 2020 [2].

The above results are general, with no assumptions on the input data points. However, in many scenarios, data points lie in high-dimensional Euclidean space and similarities are derived from the popular Gaussian kernel. We show how to exploit this extra structure to get a fast and simple to implement HC algorithm with provable approximation guarantees. It is a one-pass algorithm and is based on a random projection step. This appeared in AISTATS 2019 [29].

Finally, we study the problem of constrained Hierarchical Clustering motivated by the problem of optimal tree reconstruction in Phylogenetics and gene sequencing. In these problems, on top of the pairwise similarities, we are given triplets or quartet constraints that reflect domain expertise or prior knowledge about the data. The goal is to produce a HC that aggregates the expert constraints as consistently as possible. We provide conceptually simple top-down modifications of Sparsest Cut and Balanced Cut and demonstrate both theoretically and experimentally that they enjoy performance guarantees analogous to unconstrained HC. Part of these results appeared in ICML 2018 [35] and recent findings about the triplet and quartet consistency problem are currently under submission [34].

Overall, this thesis provides state of the art approximation and hardness results for optimizing global HC objectives, reveals unexpected connections with common linkage or graph cut heuristics and as such, I hope it serves the community as the foundation of objective-based Hierarchical Clustering and of future improvements.

1.1 Basic Concepts in Hierarchical Clustering (HC)

The undisputed queen of killer applications for HC is in phylogenetics [e.g. in 46], where genomic similarity (or dissimilarity) patterns are used to create taxonomies of organisms, with the goal of shedding light on the evolution of species by understanding the ancestral tree of life. The easiest way to view HC is as a recursive partitioning of a set of datapoints into successively smaller clusters represented by a *dendrogram*; a rooted tree whose leaves are in one-to-one correspondence with the datapoints. If you are interested more in biological, rather than computational, perspectives of HC, we refer the reader to the excellent book “Inferring Phylogenies” by Joseph Felsenstein [50].

1.1.1 Trees, Subtrees and Ancestors

We first define an *unrooted* tree (see Figure 1.2) to be an acyclic connected graph with no vertices of degree two and every leaf (vertex of degree one) labelled uniquely. Internal vertices (vertices that are not leaves) are usually left unlabelled. This, for example, can correspond to a phylogenetic tree in Biology where the leaves are different species. Next, we define a “rooted” tree almost in the same way, as here there is one special internal vertex, which can have degree two, and called the “root” of the tree. The main protagonist in this thesis will be the set of rooted binary trees (see Figure 1.3) as these are the most common in data analysis, since they correspond to natural splits into two, for a given data set. A special and important case of the aforementioned types of trees are caterpillar trees: an unrooted caterpillar tree (see Figure 1.4) has one central path with leaves branching off of it and a rooted caterpillar has leaves appended to a single path from the root to a single leaf. Caterpillar trees can describe outliers in a data set by pointing to data points that should be split from the rest.

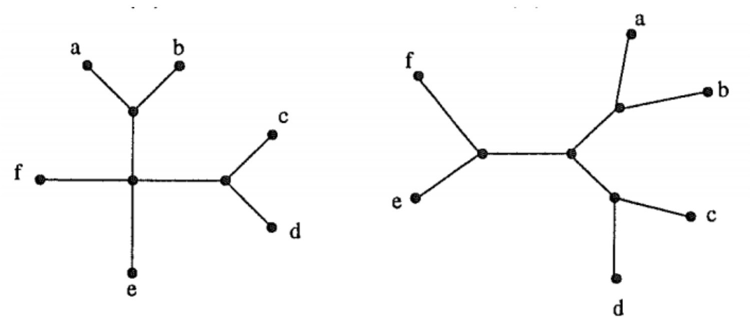

Figure 1.2: Unrooted tree examples.

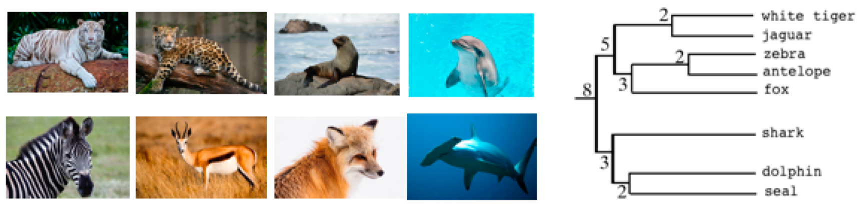

Figure 1.3: HC dendrogram with 8 datapoints (leaves); numbers are the sizes of the clusters (tree nodes).

A vertex u in a rooted tree is called a *descendant* of a vertex v if the path from u to the root r passes through v. Then, v is called an *ancestor* of u. The vertices that lie below a vertex, i.e., that are descendants of the vertex are called the *children* of the vertex, whereas and an adjacent ancestor vertex is called the *parent* of that vertex. An important concept is that of the *lowest common ancestor* of a set of vertices L, which is defined as the unique ancestor of L that is a descendant of all the ancestors of L.

From now on let T denote a rooted tree with n leaves. We want to understand what kind of meaningful groupings or clusters this tree implies. If we remove the edge between v and its parent, we will get two connected subgraphs. The set of leaves that are descendants of v is called a cluster. Rooting the subgraph containing v at the vertex v, we get subtree of T rooted at v which we denote T_v. If r is the root of T then the subtrees branching off at the root are called the maximal subtrees

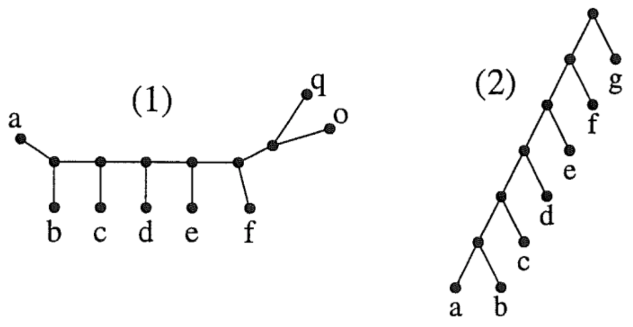

Figure 1.4: Caterpillar tree examples.

of T.

As we shall see later, our goal will be to find optimum hierarchical clusterings for a given data set. It is reasonable then to wonder "how many trees are there?", since understanding this question enables us to get a sense of the space over which we optimize.

1.1.2 – How many trees are there? – Simply, too many!

This is considered by now a standard computation as the counting of trees has been a mathematician's recreation since the fundamental work of Cayley in 1857 and 1889 [25, 26].

Starting with something simple, if we only had three elements of interest a, b, c there is only one unrooted tree with that leaf set, though there are four unrooted trees for any set of four leaves. Generally if we want an n-species tree, we can get:

$$1 \times 3 \times 5 \times \ldots \times (2n-5) = \frac{(2n-4)!}{(n-2)!2^{n-2}}$$

possible (labeled) unrooted binary trees.

As binary rooted trees with n leaves will be the main protagonist when our optimization problem is formally defined, we review here some observations to help us better understand the complexity of the problem. The proof we will present here for counting them is from Felsenstein's book [50] and was originally given by Cavalli and Edwards [24].

For every three leaves a, b, c there are exactly three binary rooted trees with leaf set a, b, c: $ab|c$ or $ac|b$ or $bc|a$. These are called rooted triplets or sometimes *resolved* rooted triplets as they describe a pair of leaves connected to a third leaf via the root. Generally, with n leaves, there are:

$$1 \times 3 \times 5 \times \ldots \times (2n-5) \times (2n-3) = \frac{(2n-3)!}{(n-2)!2^{n-2}}$$

different (labeled) binary rooted trees (left or right children are indistinguishable as the trees are considered to be unordered). See Figure 1.5 for the possible combinations with 4 leaves.

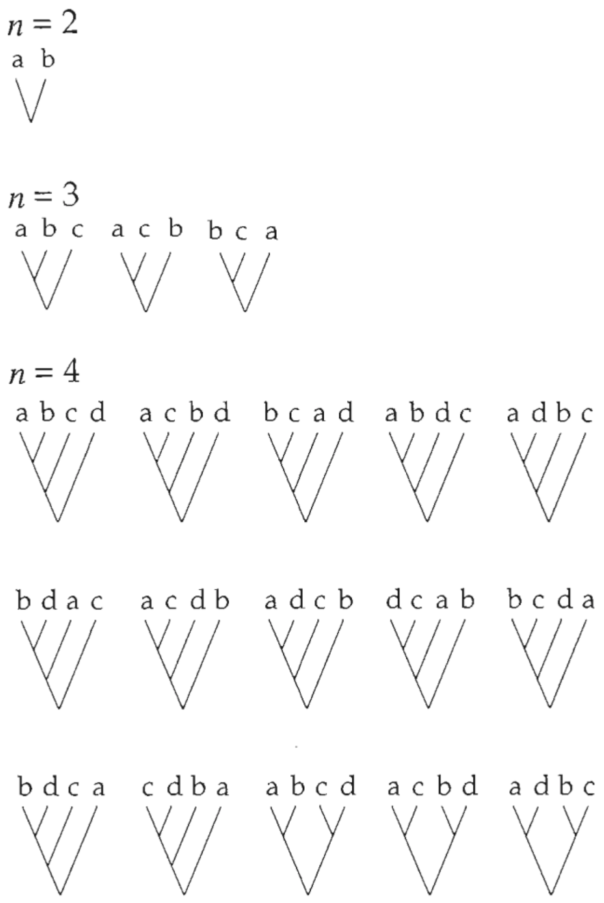

Figure 1.5: Possible combinations for generating binary rooted trees.

To prove such a formula one easy way is to try to build all possible trees, by adding one new leaf at a time. By starting from a list of all possible trees with n species and adding the new species $n+1$, in all possible places, we will generate all of the trees on $n+1$ species, each only once. Observe that the tree is bifurcating both before and after the addition, so the new species cannot be connected to an existing interior node, so it must instead be connected to a new node, which is placed in the middle of an existing edge. This implies that each internal edge of a tree is a potential location for an addition.

Again, one reasonable question to ask is “will this process lead to all trees?” or “do we generate different trees with this process?”. The truth of both statements is argued below. For some number k, let’s consider the process of adding species k to a tree that consists of species 1 through $k-1$.

Consider also the operation of removing species k from a tree that contains species 1 through k. These two operations are inverses of each other.

Suppose that we have a particular tree with n species and we remove one by one species $n, n-1, n-2$ and so on until species $k+1$ is removed. At this point what is left must be one particular tree with species 1 through k. Since the removal operation reverses the addition of the species, there must then be some particular sequence of places to add species $k+1, k+2, \ldots$ onto that k-species tree to end up with that n-species tree. Furthermore no other k-species tree can, when those $n-k$ missing species are added, yield that particular n-species tree. If there were another k-species tree that could yield it, then that tree too would be reached by removal of those species from the n-species tree. But that is a logical impossibility, as the same sequence of removals cannot result in two different trees. Thus any n-species tree can be reached from one and only one k-species tree.

Therefore, each possible addition sequence leads to a different n-species tree, and all such trees can be generated in that way. When we add species to a tree, the number of ways in which we can do that are equal to the number of internal edges, including a potential edge at the root of the tree, if the new species gets separated directly from the rest. There are 3 such branches in a two-species tree. Every time that we add a new species, it adds a new interior node, plus two new edges. Thus after choosing one of the 3 possible places to add the 3^{rd} species, the 4^{th} can be added in any of 5 places, the 5^{th} in any of 7, and so on, which leads to:

$$1 \times 3 \times 5 \times \ldots \times (2n-5) \times (2n-3) = \frac{(2n-3)!}{(n-2)!2^{n-2}}$$

Given the vast space of possibilities (e.g., $n = 20$ gives rise to $8,200,794,532,637,891,559,375$ different trees on 20 species), it is surprising that we will be able to derive conceptually simple and implementable in practice polynomial time algorithms with provable guarantees against the (unknown) optimum tree. But before we do this, let's formally define the hierarchical clustering optimization problem as was proposed by Dasgupta [43].

1.2 Global Objectives for Hierarchical Clustering

HC owes its widespread success to several advantages that this tree offers compared to the more traditional "flat" clustering approaches like k-means, k-median or k-center. It is a non-parametric method for unsupervised learning as it does not require a fixed number k of clusters and it provides richer information at all levels of granularity, simultaneously displayed in an intuitive form. Importantly, there are many fast and easy to implement algorithms commonly used in practice to find the tree. Examples are simple linkage-based agglomerative procedures, with *Single-Linkage* and *Average-Linkage* being perhaps the most popular [62, 51, 84]. We will define these and related algorithms in later sections.

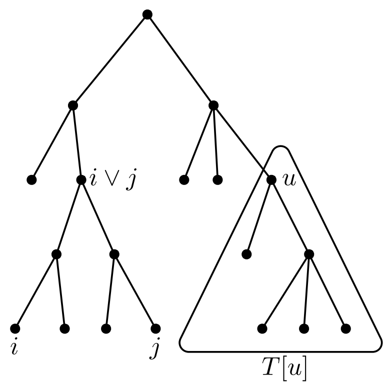

Figure 1.6: Induced subtrees and least common ancestors.

For now, let's assume that the input consists of pairwise similarities between n data points of interest. These can be for example, a set of n animals that you want to build a tree on. We would like to hierarchically cluster the n points in a way that is mindful of the given similarity structure. This requires a little terminology.

Let T be any rooted, not necessarily binary, tree whose leaves are in one-to-one correspondence with V. For any node u of T, let T_u (or $T[u]$) be the subtree rooted at u, and let $\texttt{leaves}(T_u) \subseteq V$ (or simply $|T_{ij}|$) denote the leaves of this subtree. For leaves $i, j \in V$, the term T_{ij} (or $i \vee j$) denotes their lowest common ancestor in T. Equivalently, T_{ij} is the smallest subtree whose leaves include both i and j (Figure 1.6).

The input can be a matrix or equivalently an undirected graph $G = (V, E, w)$, with one node for each point, edges between pairs of similar points, and positive edge weights $w(e)$ that capture the degree of similarity. We will sometimes omit w, in which case all edges are taken to have unit weight.

The edges $\{i, j\}$ in G, and their strengths w_{ij}, reflect locality. When clustering, we would like to avoid cutting too many edges. But in a hierarchical clustering, all edges do eventually get cut. All we can ask, therefore, is that edges be cut *as far down the tree as possible.*

Accordingly, Dasgupta in his seminal HC paper [43] defined the cost of T to be

$$\mathrm{cost}_G(T) = \sum_{\{i,j\} \in E} w_{ij}\, |\texttt{leaves}(T_{ij})| = \sum_{\{i,j\} \in E} w_{ij}\, |T_{ij}| \tag{1.1}$$

Often, it will be clear which graph G we are referring to so we will omit the subscript G. If an edge $\{i, j\}$ of unit weight is cut all the way at the top of the tree, it incurs a penalty of n. If it is cut further down, in a subtree that contains c fraction of the data, then it incurs a smaller penalty of cn. See Figure 1.7 for a simple numerical example.

Having an objective function for HC is crucial not only for guiding the optimization, but also for

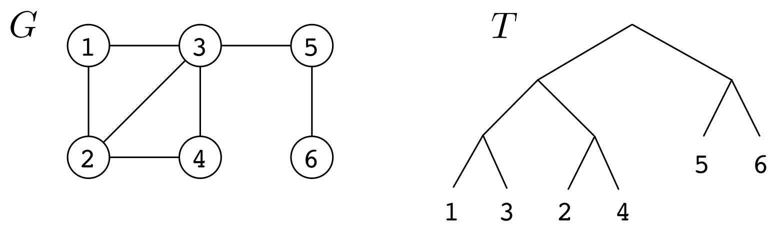

Figure 1.7: Calculating the HC cost function of Dasgupta. Edge $\{3, 5\}$ is cut at the root of T and incurs a cost of 6. Edges $\{1, 2\}, \{2, 3\}, \{3, 4\}$ each incur a cost of 4, and the remaining three edges cost 2 apiece. Thus $\text{cost}_G(T) = 6 + 3 \times 4 + 3 \times 2 = 24$.

having theoretically grounded solutions and comparing different algorithms. This may be obvious since much of the theory of standard flat clustering has been developed around objectives like k-means etc., however for HC it wasn't until 2016 that this lack of global optimization objectives was addressed by Dasgupta [43].

While optimizing this cost is NP-hard, approximation algorithms provide good approximation factors of the unknown optimum solution. In particular, one interesting aspect of this objective, as we will see, is that running Recursive Sparsest Cut (an important algorithm in theoretical computer science and in practice) would produce a HC with provable guarantees with respect to his cost function [27]. Subsequent work was able to shed light to Average Linkage performance, which is a common algorithm for HC; specifically, [78] studied the complement to Dasgupta's cost function and showed that Average Linkage will find a solution within 33% of the optimizer. Further techniques based on semidefinite programming relaxations led to improved approximations [28, 2], however with a significant overhead in runtime.

Another positive side-effect of having an objective for a problem is that faster continuous methods can be deployed giving rise to gradient-descent-based Hierarchical Clustering, e.g., see Monath et al. [77] who propose a differentiable HC objective by representing intermediate nodes in trees using hyperbolic embeddings and optimizing such embeddings. While this approach yields improvements in scalability and downstream task performance, it also has several limitations: this method requires additional post-processing to map the learned continuous tree representation to an exact tree structure. However, recently hyperbolic hierarchical clustering has received a lot of attention and is in the center of current research focus especially for ontology graphs that are common for networks [79].

1.3 Heuristics for Hierarchical Clustering

Many different heuristics have been proposed based on the idea that similar points should be merged further down the tree, i.e., earlier than less similar points (or clusters of points). Many different ways can be used based on the idea of merging "nearest neighbours" according to what one defines as the next nearest pair to be merged. These constitute a family of algorithms that build the tree in a bottom-up manner and are usually called agglomerative algorithms. One could also use a top-down approach to build the tree which splits the data set into two (or more) clusters and then recursively continues in each of the produced clusters.

Bottom-up Algorithms: Single, Complete, Average Linkage

We briefly describe algorithms that have been used as bottom up approaches to construct a tree and that originated in Biology and Phylogenetics. Agglomerative Hierachical Clustering is one of the first suite of algorithms developed to solve HC (also referred to as bottom-up linkage methods). These are simple and easy to implement algorithms that recursively merge similar data points to form some small clusters and then gradually larger and larger clusters emerge. Well-known heuristics include Single-Linkage, Complete-Linkage and Average-Linkage, that we briefly describe here. All three heuristics, start with n datapoints forming singleton clusters initially and perform exactly $n-1$ merges in total until they produce a binary tree corresponding to the HC output. At any given step, if A and B denote two already formed clusters, the criterion for which clusters to merge next is to *minimize* the minimum, maximum and average distance within clusters, for Single, Complete and Average Linkage respectively.

- For Single Linkage: $\min_{a\in A, b\in B} dist(a,b)$
- For Complete Linkage: $\max_{a\in A, b\in B} dist(a,b)$
- For Average Linkage: $\sum_{a\in A, b\in B} dist(a,b)$

If instead of pairwise distances, the input was given as a similarity graph, analogous formulas can be used, where the criterion is to maximize the respective quantities. These algorithms can be made to run in time $O(n^2 \log n)$. Recent work has also made it possible to run approximate versions of such algorithms in sub-quadratic time [1] when the data are Euclidean. As we will use them in the context of similarities we also provide the pseudocode for them in terms of similarities and some comments for each of them.

Average-Linkage: One of the main algorithms used in practice for HC, it starts by merging clusters of data points that have the highest average similarity. As we shall see, it can be shown that it achieves 1/3 approximation for the Moseley-Wang objective (one of the HC global objectives that we study in the thesis) and this is tight in the worst case. For a formal description, please refer to Algorithm 1.

Single-Linkage: One of the simplest algorithms used in practice for HC. For a formal description, please refer to Algorithm 2. This was the only HC algorithm associated with a global optimization objective before Dasgupta's work. This simple algorithm is equivalent to Kruskal's algorithm that builds the maximum spanning tree (viewed of course as a hierarchical clustering).

Complete-Linkage: Another algorithm that sometimes does better in presence of outliers by avoiding long chains of points due to noisy unreliable edges is Complete-Linkage. For a formal description, please refer to Algorithm 3.

Algorithm 1 Average-Linkage

```
1: input: Similarity matrix $w \in \mathbb{R}_{\geq 0}^{n \times n}$.
2: Initialize clusters $\mathcal{C} \leftarrow \cup_{v \in V}\{v\}$.
3: while $|\mathcal{C}| \geq 2$ do
4:     Pick $A, B \in \mathcal{C}$ to maximize:
               $w(A,B) \coloneqq \frac{1}{|A||B|} \sum_{a \in A, b \in B} w_{ab}$
5:     Set $\mathcal{C} \leftarrow \mathcal{C} \cup \{A \cup B\} \setminus \{A, B\}$
6: end while
```

Algorithm 2 Single-Linkage

```
1: input: Similarity matrix $w \in \mathbb{R}_{\geq 0}^{n \times n}$.
2: Initialize clusters $\mathcal{C} \leftarrow \cup_{v \in V}\{v\}$.
3: while $|\mathcal{C}| \geq 2$ do
4:     Pick $A, B \in \mathcal{C}$ to maximize:
               $w(A,B) \coloneqq \max_{a \in A, b \in B} w_{ab}$
5:     Set $\mathcal{C} \leftarrow \mathcal{C} \cup \{A \cup B\} \setminus \{A, B\}$
6: end while
```

Algorithm 3 Complete-Linkage

```
1: input: Similarity matrix $w \in \mathbb{R}_{\geq 0}^{n \times n}$.
2: Initialize clusters $\mathcal{C} \leftarrow \cup_{v \in V}\{v\}$.
3: while $|\mathcal{C}| \geq 2$ do
4:     Pick $A, B \in \mathcal{C}$ to maximize:
               $w(A,B) \coloneqq \min_{a \in A, b \in B} w_{ab}$
5:     Set $\mathcal{C} \leftarrow \mathcal{C} \cup \{A \cup B\} \setminus \{A, B\}$
6: end while
```

Top-down Algorithms: Sparsest/Balanced Cut, Bisecting k-means

Sparsest and Balanced cut are two problems that have been studied for many years in theoretical computer science. We defer the formal definitions to the Preliminaries Section as they will both be important in our later discussions for Dasgupta's objective. The idea behind both of them is to split a given data set into two clusters so that points inside the clusters are more similar to each other compared to points outside the clusters. The way to formalize this is based on the size of the cut normalized by the sizes of the two sides. These can be used to generate HC, simply by recursively splitting each of the two produced clusters. As these algorithms may be slow or require special engineering to accelerate, simpler variants can be used like Bisecting k-means, which simply is the recursive 2-means algorithm based for example on standard implementations of Lloyd's k-means algorithm [63] with the parameter $k = 2$.

Remark 1 *We would like to note that depending on the input characteristics, different algorithms may have better or worse performance and there is no single silver bullet. During my internships at Facebook and Google, I came to realize the many different variations of bottom-up or top-down algorithms that can be deployed each having drawbacks or advantages. One interesting family of variants is so-called "percentage-linkage" algorithms which is defined similarly to aforementioned linkage heuristics, except that only some top percentile $X\%$ (e.g., $50\%, 75\%$ or 95%) of points contribute to the calculation of the linkage or cut criterion. This can have some robustness advantages in practice due to the presence of outliers.*

1.4 Constrained Hierarchical Clustering

So far, we thought of the input as being presented in the form of pairwise similarities or distances. This is the standard way of collecting the input for clustering. However, hierarchies are much richer objects than partitions and the input can be given in other forms as well. Specifically, information can take the form of *triplets* or *quartets* which are extremely useful in phylogenetic tree reconstruction [50, 3].

1.4.1 Connections to Consistency in Phylogenetics

The three unrooted trivalent (i.e., internal nodes having degree 3) trees with four leaves are called quartets. These are $ab|cd$ or $ac|bd$ or $ad|bc$ (see also Figure 1.8). We say that a quartet $ab|cd$ is obeyed (or satisfied or followed) by a tree T if the path from a to b in T does not intersect the path from c to d in T. If this is not the case, we say a quartet was disobeyed (or violated or not satisfied) by the tree.

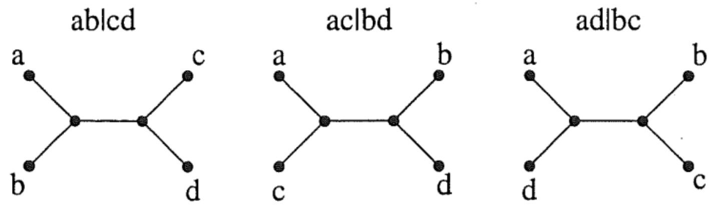

Figure 1.8: Resolved quartets on three elements are basic building blocks for phylogenetic tree reconstruction.

Similarly, a rooted triplet (see Figure 1.9) is obeyed/violated by a binary rooted tree T if the path from a to b does not share any vertices with the path from c to the root. In other words, the rooted triplet indicates which of the three nodes is least similar (or the "odd" element among the three) and should be separated first. For example, we could have the triplet {Tuna, Salmon|Lion}.

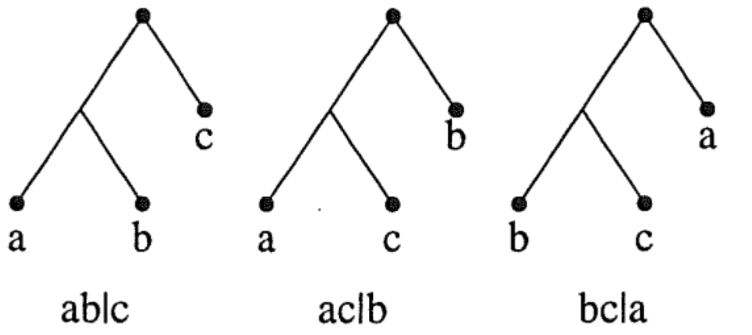

Figure 1.9: Resolved triplets on three elements are basic building blocks for binary HC.

Triplets and quartets can be thought as the basic building block for conveying local qualitative information on 3 or 4 elements respectively. They are also easy to obtain via experiments or expert advice. However, many interesting questions arise: given a set of triplets or quartets, can we combine them in a tree consistently? If not, can we maximize agreements or minimize disagreements with the given triplets/quartets?

These are all old questions in the field of Phylogenetics and form interesting combinatorial problems. We briefly review some results here that are relevant for our discussion and our later results.

All 4 problems (max satisfied triplets/quartets or min violated triplets/quartets) are NP-hard. Detecting consistency can be done in polynomial time only for the triplets case and a simple algorithm for this is BUILD from Aho et al. [3]. For the maximization version both of triplets or quartets consistency, it is easy to see that a random solution (i.e., a random rooted or unrooted binary tree respectively) will achieve a $\frac{1}{3}$-approximation. This is also obtained by simple greedy bottom-up or top-down heuristics, however no one till this day was able to find a better algorithm, despite significant efforts in different communities [37, 23, 89]. Here we will show that there is reason for that as perhaps a better algorithm would imply that the Unique Games Conjecture is false. Here we show that no polynomial time can obtain better than a $\frac{2}{3}$-approximation (assuming the Unique Games Conjecture to be defined in the Preliminaries) and we conjecture that $\frac{1}{3}$ is the best constant one can achieve in polynomial time.

In fact, even more variations can be defined since we may want to avoid a triplet or quartet, i.e., not include it in the final tree. This captures scenarios where a certain triplet/quartet is highly unlikely to be what one is looking for. This problem is called the forbidden triplets or forbidden quartets problem. Again a random solution achieves a $\frac{2}{3}$-approximation and this is currently the best performance. As we prove later, assuming the Unique Games Conjecture [65], one cannot hope to beat this trivial baseline, in polynomial time (see also Conjecture 1 in Chapter 7).

For the minimization versions, the situation is slightly clearer. As said previously, detecting consistency in quartets is NP-complete, so there is no hope for multiplicative factor approximation algorithms. For minimizing violated triplets, there is a hardness result due to [37], saying that for every $\epsilon > 0$, the problem is hard to approximate within a factor of $2^{\log^{1-\epsilon} n}$, where n are the number of leaves, even on trees formed by multi-edges (coincidentally, and perhaps ironically, I had been working for this problem several months now and while writing the acknowledgments for this thesis, I realized I had stumbled upon the work of [37], years back, as their reduction is inspired by influence maximization that was the start of my undergraduate thesis [32, 57]). This means that there is no sub-linear approximation factor for MinRTI, matching the best (and relatively straightforward) approximation so far which is an n-approximation, based on recursively taking Min-Cuts [60]. Here, we mention a novel and easy way of getting $O(\log n \log\log n)$-approximation in the special case of caterpillar trees: we give a reduction to Feedback Arc Set (FAS); simply introduce two arcs $a \to b$ and $a \to c$ for every triplet $bc|a$ in the input of the triplets consistency problem and run approximately the FAS algorithm to produce an ordering. This closes this problem almost completely as an improved algorithm would imply better Feedback Arc Set algorithms which would be a big deal in the approximation algorithms literature, as Feedback Arc Set is a fundamental problem (see [48]).

1.4.2 Connections to Correlation Clustering and Rankings

The triplets and quartets constraints have their analogue in standard clustering and in rankings (i.e., permutations on n labels). These constraints sometimes are referred to as qualitative constraints or ordinal constraints and are used in correlation clustering and voting schemes.

"Must-link/Cannot-Link"

In clustering, the most common types of such qualitative information are "must-link/cannot-link" constraints, specifying that in the desired ground-truth clustering two data points should be in the same or in different clusters. This naturally gives rise to instances of Correlation Clustering [17, 4] and constrained clustering [97, 98] which are both important and well-studied problems in data analysis.

Betweenness and other Ordering Constraint Satisfaction Problems (CSPs)

When talking about ranking players in a tournament or ranking ads in an ad platform, or ranking presidential candidates according to votes etc. it is usually the case that we have ordering constraints. The simplest ordering constraint takes the form of a pairwise relationship $a < b$ indicating that in the output permutation, a should be before b. This is a fundamental problem in computer science called the Maximum Acyclic Subgraph or Minimum Feedback Arc Set which is related to topological sorting. Again let's discuss about consistency. When the given constraints are consistent, i.e., there exists one permutation satisfying all of them, then a simple linear time algorithm based on Depth-First-Search can find the optimum solution. In the case of inconsistencies, we try to maximize the number (or weight) of forward edges or minimize the number of backward edges in a permutation.

Surprisingly, again a random solution is the best one can achieve for the maximization version of Maximum Acyclic Subgraph [59], in polynomial time (assuming the Unique Games Conjecture). A random solution achieves a $\frac{1}{2}$-approximation and the same guarantee is actually obtained by an arbitrary permutation or its reversed. For the Feedback Arc Set problem, the current best is an $O(\log n \log\log n)$-approximation [48].

Going from pairs to triplet constraints in the case of rankings, one common constraint is that of a betweenness constraint $a|b|c$, indicating that b should be between a and c in the optimum permutation. Even though it may seem as a simple problem, checking consistency of a set of betweenness constraints is NP-complete. Furthermore, a random permutation obtains a $\frac{1}{3}$-approximation for maximizing obeyed triplets by the permutation, and this is the best we can do in polynomial time [58].

Part of our contribution in the later chapters in this thesis, is to formally make a connection between ordering problems on permutations (so-called ordering CSPs [58]) and ordering problems on trees, getting near-optimal hardness results for the latter.

1.5 Further Related Work

As we mentioned, there is a large body of literature on HC (we refer the reader to [19] for a survey) starting with early works in phylogenetics by [88, 61]. Average-Linkage was one of several methods originating in this field that were subsequently adapted for general-purpose data analysis. Other major applications include image and text classification [90], community detection in social networks [70, 75], bioinformatics [45], finance [94] and more.

Following the formulation of HC as a combinatorial optimization problem, and escaping from worst-case analysis Cohen-Addad et al. [38] studied hierarchical extensions of the stochastic block model and showed that an older spectral algorithm of [76] augmented with linkage methods results in an $O(1)$-approximation to Dasgupta's objective.

In another line of work, hierarchical clustering in the context of "dynamic" or "incremental" clustering, using standard flat-clustering objectives like k-means, k-median or k-center as proxies, has been studied ([30, 42, 81, 72]). Furthermore, there has been recent attention on the "semi-supervised" or "interactive" versions of HC by [12, 13, 15, 16], showing that interactive feedback in the form of cluster split/merge requests can lead to significant improvements, and by [96], providing techniques for incorporating prior knowledge to get better hierarchical trees.

1.6 Structure of this PhD Thesis

In the next chapter, we give a gentle, yet technical overview of our main results by providing the statements of our main theorems all concentrated together. In Chapter 3, we provide the necessary technical background and preliminaries from the areas of approximation algorithms and hardness of approximation. The main part of the thesis starts in Chapter 4, where we present our findings on approximation algorithms for HC based on standard primitives from graph partitioning and convex optimization, together with some negative impossibility results. Continuing in Chapter 5, we study a variation where input points have features and we want a fast HC algorithm. In the last technical chapter of the main body, Chapter 6, we study HC with constraints that constitute a common way of incorporating prior knowledge or expert advice in biology and make progress in old Biology problems especially giving hardness for maximizing triplets consistency. Chapter 7 contains a list of future research questions and conjectures. Finally, omitted proofs and experiments are given in Appendix A.

Chapter 2

Gentle Overview of Main Results

I often quote myself.
It adds spice to my conversation.

-George Bernard Shaw

This chapter is meant to serve as a brief, yet technical summary of the thesis for someone who just wants to dive into our main results about Hierarchical Clustering. We assume some familiarity with graph algorithms as our purpose is to only state our main theorems.

2.1 Black-box Approximations for Hierarchical Clustering

Hierarchical Clustering (HC) objectives are defined and optimized over the space of binary trees with n leaves, where n is the number of data points. Every binary tree with n leaves can be seen as indicating a series of exactly $n-1$ splits: at each internal node of the tree, simply group together the leaves that belong to the left or right child. The main message of this section is that we can exploit well-known techniques for graph partitioning in a black-box manner and get approximation algorithms for the HC objectives. These well-known techniques include algorithms like Sparsest Cut, Balanced Cut, Max Cut, Max Uncut Bisection or *spreading metrics* semidefinite programming (SDPs) relaxations and randomized hyperplane rounding. For the hardness of approximation results, we rely on the Minimum Linear Arrangement problem and Small-Set Expansion hypothesis.

2.1.1 Minimizing Dasgupta's cost via Graph Cuts and Spreading Metrics

Recall that on an input similarity graph G, the Dasgupta's cost for a given tree T is the following:

$$\text{cost}(T) = \sum_{(i,j)\in E} w_{ij}\,|T_{ij}| \tag{2.1}$$

where w_{ij} are pairwise similarities and $|T_{ij}|$ is the number of leaves contained in the subtree rooted at the lowest common ancestor of i, j. Finding the optimum tree is an NP-hard task so we need to result in approximate solutions. We prove that repeatedly using Sparsest Cut or Balanced Cut to generate a candidate tree T will produce a good solution for Dasgupta's objective:

Theorem 2.1.1 *Given a weighted graph G and access to an α_n-approximation of either* Sparsest Cut *or* Balanced Cut*, we can get an $O(\alpha_n)$ approximation for the hierarchical clustering problem based on Dasgupta's objective.*

The current best known value for α_n is $O(\sqrt{\log n})$. The above theorem may come as a surprise because both algorithms are looking for small cuts normalized by sizes, i.e., *ratio* cuts, even though there is nothing in Dasgupta's cost function that involves any ratio cuts; the ratio aspect is just emerging organically from it because the penalties depending on the leaves $|T_{ij}|$ are accumulated in a hierarchical manner. The above theorem is also a justification for recursive top-down partitioning methods that have been previously proposed in practice. Independently of our work, Roy and Pokutta [85] achieved a $O(\log n)$-approximation using a spreading metrics linear programming. Shortly after our publication, and independently, Cohen-Addad et al. [39] also presented the same theorem, however with a different proof approach.

We note here that similar results hold for a more general version of the HC objective given by:

$$\text{cost}(T) = \sum_{(i,j)\in E} w_{ij}\, f(|T_{ij}|) \tag{2.2}$$

where f is defined on the nonnegative reals, is strictly increasing, and has $f(0) = 0$. For instance, we could take $f(x) = \log(1+x) or f(x) = x^2$. We have:

Theorem 2.1.2 *Given a weighted graph G and access to an α_n-approximation of either* Sparsest Cut *or* Balanced Cut*, we can get an $O(c_f \cdot \alpha_n)$ approximation for the hierarchical clustering problem based on Dasgupta's objective (2.2), where $c_f \triangleq \max_{1\le r\le n} \frac{f(r)}{f(r/2)-f(r/4)}$.*

A natural way to attack the question of approximating the HC cost function (2.1) is by formulating a linear programming (LP) or a semidefinite programming (SDP) relaxation. In fact, our Theorem 2.1.1 was originally inspired by our analysis of a hierarchical SDP with *spreading* metrics. Using our SDP and exploiting a connection with k-Balanced Partitioning studied in [67], we prove that it is an $O(\sqrt{\log n})$ approximation for both the simple and the generalized cost function.

The advantage of the SDP relative to the Sparsest Cut or Balanced Cut is that it provides an explicit lower bound on the optimal solution and could potentially yield an even better approximation for hierarchical clustering. Finally, before considering an SDP, one can analyze a spreading metric Linear Programming (LP) relaxation and show that it has integrality gap $O(\log n)$, hence it gives an $O(\log n)$ approximation for Dasgupta's objective.

Theorem 2.1.3 *Using the vectors returned by* SDP-HC*, and a rounding algorithm for the* k-Balanced Partitioning *propblem, we can produce a tree within a factor of* $O(\sqrt{\log n})$ *from the optimum tree, as measured by (2.1).*

Below we give the SDP-HC formulation, although a detailed presentation behind each of the constraints will be postponed until Chapter 4.

We view a hierarchical clustering of n data points as a collection of partitions of the data, one for each level $t = n-1, \dots, 0$. The partition at level $(t-1)$ is a refinement of the partition at level t. A level t defines a partitioning of the data points into *maximal* clusters of size at most t. Note that the partition corresponding to $t = 1$ must consist of n singleton clusters. We represent the partition at level t by the set of variables x_{ij}^t, $i, j \in V$, where $x_{ij}^t = 1$ if i and j are in different clusters in the partition at level t and $x_{ij}^t = 0$ if i and j are in the same cluster. Each of the vectors v_i^t lies in $\mathbb{R}^n$.

$$\min \sum_{t=0}^{n-1} \sum_{ij \in E} x_{ij}^t w_{ij} = \min \sum_{t=0}^{n-1} \sum_{ij \in E} \frac{1}{2} \|v_i^t - v_j^t\|_2^2 w_{ij}, \tag{2.3}$$

$$\text{such that:} \quad x_{ij}^t \le x_{ij}^{t-1}, \quad t = n-1, n-2, \dots, 1 \qquad \text{(SDP-HC)}$$

$$x_{ij}^0 = 1, \forall i, j \in V \text{ and } x_{ij}^t \le 1, \forall i, j, t$$

$$x_{ij}^t = \frac{1}{2}\|v_i^t - v_j^t\|_2^2 \text{ and } \|v_i^t\|_2^2 = 1, \forall i, t \in V$$

$$x_{ij}^t \le x_{jk}^t + x_{ik}^t, \forall i, j, k \in V, \ \forall t \text{ and } \sum_j x_{ij}^t \ge n - t, \forall i, t$$

The constraints $\sum_j x_{ij}^t \ge n - t, \forall i, t$ try to impose the vectors to separate encoding the fact that a tree splits data points as we move towards the leaves. These are the so-called *spreading* constraints and that's why we say that spreading metrics are used in the relaxation.

2.1.2 Maximization HC: Variations on a Theme

As we all know, in flat clustering and graph partitioning there is not just one objective to optimize. One may study Mininum Cut, Maximum Cut, k-means, k-median and many more. Each objective may be relevant in different contexts or may serve different purposes, like explaining the success of a popular algorithm used in practice. The situation was no different for hierarchical clustering and shortly after Dasgupta's proposal for an HC objective, several works proposed variations on it. The two most important were the maximization objective by Moseley and Wang [78] and the dissimilarity objective by Cohen-Addad et al. [39], which was also a maximization problem.

The objective that Moseley and Wang proposed is the following:

$$\text{reward}(T) = \sum_{(i,j)\in E} w_{ij}\,(n - |T_{ij}|) \tag{2.4}$$

Again here the graph is given as pairwise similarities w_{ij} and the term $(n - |T_{ij}|)$ denotes the number of non-leaves of the lowest common ancestor of i, j, i.e., the number of data points lying outside T_{ij}. We note that this is a maximization and not a minimization problem as we had previously. It is easy to see that this objective is the *complement* of Dasgupta's cost function, hence the two optimization problems have the same optimum solution.

The objective that Cohen-Addad et al. proposed is useful whenever the input has pairwise dissimilarities d_{ij} instead of similarities:

$$\text{score}(T) = \sum_{(i,j)\in E} d_{ij}\,|T_{ij}| \tag{2.5}$$

This is again a maximization problem. We prove the following results:

Theorem 2.1.4 *In the worst case, the approximation ratio achieved by Average-Linkage is no better than* $1/3$ *for the Moseley-Wang maximization objective (2.4) and no better than* $2/3$ *for the Cohen-Addad et al. maximization objective (2.5).*

Theorem 2.1.5 *We beat the approximation guarantees of Average-Linkage for both objectives: for the Moseley-Wang maximization objective (2.4) we use the hierarchical* SDP-HC *to get a* 0.336379*-approximation and for the Cohen-Addad et al. maximization objective (2.5) we use* Max Cut *to get a* 0.667078*-approximation.*

Finally, the SDP-HC used in the analysis for the 0.336379-approximation also paved the way to get improved guarantees for objective (2.4) via a black-box connection to Max Uncut Bisection:

Theorem 2.1.6 *For objective (2.4), one can get a* 0.4246*-approximation via* Max Uncut Bisection.

Remark 2 *Subsequent work by Alon et al. [5], during the writing of this thesis, showed that our algorithm based on* Max Uncut Bisection *actually yields a* 0.585*-approximation, thus improving our analysis from* 0.4246.

2.1.3 Two Hardness Results from Small-Set Expansion

All 3 objectives defined previously are NP-hard to optimize exactly, but what about inapproximability? Our two hardness of approximation results in this section are based on the Minimum Linear

Arrangement problem and the Small-Set Expansion (SSE) Hypothesis. They hold even for unweighted graphs (i.e., w_{ij} is either 0 or 1) and are the following:

Theorem 2.1.7 *For Dasgupta's cost (2.1), for every $\epsilon > 0$, it is SSE-hard to distinguish between the following two cases for a given graph $G = (V, E)$, with $|V| = n$:*
Yes: *There exists a decomposition tree T of the graph such that $cost_G(T) \leq \epsilon n|E|$*
No: *For any decomposition tree T of the graph $cost_G(T) \geq c\sqrt{\epsilon}n|E|$.*

In other words, this implies that no constant factor approximation algorithm can exist that runs in polynomial time, as this would refute the Small-Set Expansion hypothesis.

Theorem 2.1.8 *Assuming* Small-Set Expansion, *there exists $\epsilon > 0$, such that it is* NP*-hard to approximate the Moseley-Wang objective (2.4) function within a factor* $(1 - \epsilon)$.

In other words, this is an APX-hardness result for objective (2.4).

Finally, regarding the dissimilarity objective (2.4), a similar APX-hardness result can be proved based on the Unique Games Conjecture. As this is unpublished work in progress [33], we omit the details in this thesis.

2.2 Hierarchical Clustering for Euclidean Data

In the previous sections, we focused on optimizing the various HC objective functions under arbitrary similarity measures w_{ij} or dissimilarities d_{ij}. Although this is a general setting, in practice more information is sometimes provided for our data points: specifically, each data point is usually represented with a vector whose coordinates represent features and we need a scalable algorithm. Experimental evidence in small and large datasets (up to 10 million data points) demonstrate orders of magnitude improvement in speed-ups while still maintaining the quality of the output compared to more expensive methods.

Our main result here is a fast, one-pass algorithm to handle the high-dimensional case and gets a slightly better approximation factor compared to the slower Average-Linkage:

Theorem 2.2.1 *For any input set of vectors $v_1, \ldots, v_n \in \mathbb{R}^d$ the algorithm* Projected Random Cut *gives an α-approximation (in expectation) for the objective (2.4) under the Gaussian kernel similarity measure $w_{ij} \sim e^{-\|v_i - v_j\|_2^2/2\sigma^2}$ where $\alpha = (1 + \delta)/3$ for $\delta = \min_{i,j} \exp(-\frac{\|v_i - v_j\|_2^2}{2\sigma^2})$.*

More details about the Projected Random Cut algorithm and the motivation behind the Gaussian kernel use, are deferred to the relevant section. Here we also analyzed Average-Linkage and its performance guarantees for different dimensions:

Theorem 2.2.2 *For $d = 1$, under monotone distance-based similarity measures $w_{ij} = w(\|v_i - v_j\|)$,* Average-Linkage *is a $\frac{1}{2}$-approximation for the objective (2.4). Moreover, there exists a set of* *vectors* $v_1, \dots, v_n \in \mathbb{R}^d$ *for* $d = poly(\log n)$ *for which* AVERAGE-LINKAGE *gets at most* $\frac{1}{3} + o(1)$ *of the optimum.*

2.3 Hierarchical Clustering with Structural Constraints

In this final section of the thesis, we explore the question of approximation algorithms for *constrained* hierarchical clustering and as it turns out we uncover interesting relations with some old computational biology problems. Furthermore, we provide some experimental evidence that the final hierarchical clustering gets improved when on top of the geometry of the data (i.e., pairwise similarities w_{ij}), the input contains triplet constraints as they can guide the algorithm towards better trees overall. What is a *constraint* in this context? In several applications it may be easier to acquire information on *triplets* of points, instead of *pairwise* information w_{ij}. The triplet information is of the form $ab|c$ indicating that a, b are closer in the hierarchy than a, c or a, b. For example, in the Tree of Life, such a constraint could be {lion, tiger}|penguin since species like lions and tigers share a common genetic signature compared to penguins. How can we change the algorithms mentioned so far to incorporate these triplet constraints? Can we achieve approximation guarantees subject to satisfying all triplet constraints? The answer depends on the consistency of the triplet constraints and the objective at hand. We say the constraints are consistent if there exists a tree compatible with all the given triplets.

Our main result here is that a modified version of SPARSEST CUT and BALANCED CUT will still be reasonably good approximation algorithms:

Theorem 2.3.1 *Given a weighted graph* $G(V, E, w)$ *with* k *consistent triplet constraints* $ab|c$ *for* $a, b, c \in V$, *the* CONSTRAINED SPARSEST CUT *algorithm outputs a HC respecting all triplet constraints and achieves an* $O(k\alpha_n)$*-approximation for Dasgupta's objective (2.1), where* $\alpha_n = \sqrt{\log n}$.

Similar results can be achieved in the case some of the triplets may be inconsistent. This happens for example when the triplets are generated by asking users or domain experts and there is some conflicting opinion regarding some of them. For this, we need to modify Dasgupta's objective accordingly to reflect the fact that we would like to satisfy as many constraints as possible, as early in the hierarchy as possible, or in other words, to violate as few as possible as late as possible. We postpone the formal definition of the modified objective to the relevant section.

2.3.1 Connections to Rankings, Correlation Clustering and Phylogeny Reconstruction

Here we present mainly hardness results for important problems in computational biology and bioinformatics on triplets and quartets consistency. As our paper is currently under submission [34], here we will only informally state the results and later provide some proofs. We defer the complete proofs to the final version of our paper.

By exploiting a connection with ordering constraint satisfaction problems [58], we are able to obtain near-optimal (optimal in some cases) hardness of approximation, thus giving an explanation for why many efforts were stuck at the approximation thresholds obtained by trivial baselines like random solutions.

Theorem 2.3.1 *For maximizing rooted triplets consistency or forbidden triplets, one cannot achieve better than $\frac{2}{3}$-approximation, in polynomial time. For the second problem, this is tight as it is obtained by a random tree. Similar results, hold for maximizing quartets consistency.*

In our ongoing work [34], we show how to beat the trivial random baselines in a simple random access model, where we can ask noisy queries and obtain triplets or quartets or ordering constraints. Our approach uses interesting variants of Max Cut with negative weights and can also be extended to correlation clustering where we are able to beat the current best 0.766-approximation of Swamy [91]. Finally, we unveil some interesting connection between the recent global objectives that are the focus of this thesis (Dasgupta, Moseley-Wang and Cohen-Adad et al. objectives) with the aforementioned triplets consistency problems. As a byproduct we get a polynomial time approximation scheme for HC instances that are dense. We omit details from the current thesis.

Chapter 3

Preliminaries

You shall know a word by the company it keeps.

- John Rupert Firth

Here, we would like to briefly discuss some important problems and definitions that will frequently come up in the rest of the thesis. Some additional definitions and facts may be presented in the sections for which they are relevant. Section 3.2 where we introduce the Hierarchical Clustering global objectives should be new to most readers; however, a reader with background in theoretical computer science is most likely familiar to the majority of the problems and results presented here.

3.1 Background in Algorithms and Hardness

Linkage-based hierarchical clustering. Among various algorithms popular in practice for HC, we focus on two very popular ones: SINGLE-LINKAGE and AVERAGE-LINKAGE which are two simple agglomerative clustering algorithms that recursively merge pairs of nodes (or super nodes) to create the final binary tree. Recall that SINGLE-LINKAGE picks the pair of super nodes with the maximum similarity weight between their data points, i.e., merges A and B maximizing $\max_{i\in A, j\in B} w_{ij}$. On the contrary, AVERAGE-LINKAGE picks the pair of super-nodes with maximum average similarity weight at each step, i.e., merges A and B maximizing $\frac{w_{AB}}{|A|\cdot|B|}$, where $w_{AB} := \sum_{i\in A, j\in B} w_{ij}$.

Basic Graph Primitives. All approximations mentioned in this thesis will be multiplicative factor approximations with respect to the unknown optimum solution. Let's review some standard graph partitioning problems:

Sparsest Cut. Given a weighted, undirected graph $G = (V, E, w)$ ($|V| = n$) we want to find a set $S \neq \emptyset, V$ that minimizes the ratio: $\frac{w(S, V\setminus S)}{|S|\cdot|V\setminus S|}$. It is an NP-hard problem for which many important results are known including the LP relaxation of Leighton-Rao [69] with approximation ratio $O(\log n)$ and the SDP relaxation with triangle inequality of Arora, Rao, Vazirani [8] with approximation ratio $O(\sqrt{\log n})$; it is a major open question if we can improve this approximation ratio.

c-Balanced Cut. This is another variation of balanced partitioning where we want to have some kind of guarantee on the size of the smallest part produced by the cut. Formally, we are given a weighted undirected graph G on n vertices and the goal is to partition the vertices into 2 pieces $(S, \bar{S})$ with sizes $cn \leq |S|, |\bar{S}| \leq (1 - c)n$ and of course with the total weight of the edges connecting the 2 components being small. It is known ([8]) how to get an $O(\sqrt{\log n})$ pseudo-approximation algorithm for this problem, in the sense that the algorithm will produce a c'-Balanced Cut for some constant $c' < c$ and such that the cost of the solution is at most $O(\sqrt{\log n})$ times the optimum c-Balanced Cut. Here c' and c are constants in $(0, 1/2]$. We note that the above two problems (Sparsest and Balanced Cut) can be approximated up to a $O(\sqrt{\log n/\epsilon})$ factor in time $\tilde{O}(m + n^{1+\epsilon})$ by combining [86] with [87] or [64] (see also [80]).

k-Balanced Partitioning. Given a weighted undirected graph G on n vertices, the goal is to partition the vertices into k equally sized components of size roughly n/k so that the total weight of the edges connecting different components is small. It is an important generalization of well-known graph partitioning problems, including minimum bisection (k=2) and minimum balanced cut and it has applications in VLSI design, data mining (clustering), social network analysis etc. It is an NP-hard problem and the authors of [67] present a bi-criteria (which means that pieces may have size $2n/k$ rather than n/k) approximation algorithm achieving an approximation of $O(\sqrt{\log n \log k})$. Their result will be useful in our analysis for our spreading metrics SDP in Section 4.1.4. However, for us the dependence on k will be unimportant since in our analysis we only need k to be a small constant (e.g. k=4).

Hardness. We will make use in two places of this thesis of the so-called Small Set Expansion (SSE) Hypothesis [82, 83] that is tightly related with an important question in approximability and hardness of approximation called the Unique Games Conjecture (UGC) [65]. As we will not rely on UGC directly, we omit technical details from here and focus only on SSE.

Small-Set Expansion. SSE is a hardness assumption that informally tells us the following: Given a graph G, it should be hard to distinguish between the case where there exists a small set S that has only a few edges leaving it versus the case where for all small sets S there are many edges leaving the sets. For a formal statement see Section 4.2.6. This hardness assumption is closely connected to the Unique Games Conjecture (UGC) of [65] and its variants. In particular, the SSE Hypothesis implies UGC([82]) and it has been used to prove many inapproximability results for problems like balanced separator and minimum linear arrangement ([83]).

Minimum Linear Arrangement. Given a weighted undirected multigraph $G(V,E,w)$ ($|V| = n$) we want to find a permutation $\pi : V \to \{1,2,\dots,|V|\}$ that minimizes: $\sum_{(x,y)\in E, x<y} w(x,y)\cdot|\sigma(y)-\sigma(x)|$. A factor $O(\sqrt{\log n}\log\log n)$ approximation for MLA was shown in [31, 49]. In addition, some recent hardness results are also known: in [83] it is shown that it is SSE-hard to approximate MLA to within any fixed constant factor and in [6] the authors prove that MLA has no polynomial time approximation scheme, unless NP-complete problems can be solved in randomized subexponential time.

Gaussian Graphs. For a constant $\rho \in (-1,1)$, let $\mathcal{G}(\rho)$ denote the infinite graph over $\mathbb{R}$ where the weight of an edge (x,y) is the probability that two standard Gaussian random variables X, Y with correlation ρ equal x and y respectively. The expansion profile of Gaussian graphs is given by $\Phi_{\mathcal{G}(\rho)}(\mu) = 1 - \Gamma_\rho(\mu)/\mu$ where the quantity $\Gamma_\rho(\mu)$ defined as

$$\Gamma_\rho(\mu) \triangleq \mathbb{P}_{(x,y)\sim\mathcal{G}_\rho}(x \ge t, y \ge t),$$

where $\mathcal{G}_\rho$ is the 2-dimensional Gaussian distribution with covariance matrix:

$$\begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$$

and $t \ge 0$ is such that $\mathbb{P}_{(x,y)\sim\mathcal{G}_\rho}\{x \ge t\} = \mu$

3.2 Global Objectives for Hierarchical Clustering

Recall the setting for Dasgupta's cost function [43]. The edges $\{i,j\}$ in G, and their strengths w_{ij}, reflect similarity. When clustering, we would like to avoid cutting too many edges. But in a hierarchical clustering, all edges do eventually get cut. All we can ask, therefore, is that edges be cut *as far down the tree as possible.* Accordingly, he defined the cost of T to be

$$\text{cost}_G(T) = \sum_{\{i,j\}\in E} w_{ij}\,|\texttt{leaves}(T[i \vee j])|. \tag{3.1}$$

where from now on we will simply use $|T_{ij}$ instead of $|\texttt{leaves}(T[i \vee j])|$.

Shortly after Dasgupta's work, two recent (and independent) works took this objective function viewpoint to understand the performance of Average-Linkage. In the first work, Moseley and Wang [78] introduced a new objective that explicitly favors postponing the cutting of "heavy" edges to when the clusters become small, which is in some sense dual to the objective introduced by Dasgupta:

$$T^* = \underset{\text{all trees } T}{\operatorname{argmax}} \sum_{(i,j)\in E} w_{ij} \cdot (n - |T_{ij}|) \tag{3.2}$$

In many applications, the geometry in the data is given by dissimilarity scores instead of similarities. In the second work, Cohen-Addad, Kanade, Mallmann-Trenn and Mathieu [39] took this view and studied a maximization version of Dasgupta's objective where pairwise weights w_{ij} denote dissimilarities between the endpoints:

$$T^* = \underset{\text{all trees } T}{\operatorname{argmax}} \sum_{(i,j)\in E} w_{ij} \cdot |T_{ij}| \tag{3.3}$$

For maximizing the similarity-based objective, the first work showed that Average-Linkage obtains a $\frac{1}{3}$-approximation. Interestingly, for maximizing the dissimilarity-based objective, the second work also showed that Average-Linkage gives a $\frac{2}{3}$-approximation [39]. Besides helping with understanding the performance of Average-Linkage, a comprehensive list of desirable properties of the aforementioned objectives together with their proofs can be found in [43, 78, 39, 38]. Here we will only mention some key properties to give some intuition:

- Relations of different objectives: As can be seen by the formulations in 3.1, 3.2, 3.3, all three objectives look similar. Indeed in terms of optimization these are equivalent. However, in terms of approximation the landscapes change as is common in approximation algorithms.

- Interpretation of 3.1: A natural interpretation of the cost function is in terms of cuts. Each internal node of the tree corresponds to a *split* in which some subset of nodes $S \subseteq V$ is partitioned into two or more pieces. For a binary split $S \to (S_1, S_2)$, the splitting cost is $|S|\, w(S_1, S_2)$, where $w(S_1, S_2)$ is the weight of the cut,

  $$w(S_1, S_2) \;=\; \sum_{\{i,j\}\in E:\ i\in S_1, j\in S_2} w_{ij}.$$

  This extends in the obvious way to k-ary splits $S \to (S_1, S_2, \ldots, S_k)$, whose cost is $|S|w(S_1, \ldots, S_k)$, with

  $$w(S_1, \ldots, S_k) \;=\; \sum_{1\le i<j\le k} w(S_i, S_j).$$

  The cost of a tree T is then the sum, over all internal nodes, of the splitting costs at those nodes:

  $$\text{cost}(T) = \sum_{\text{splits } S \to (S_1,\ldots,S_k) \text{ in } T} |S|w(S_1, \ldots, S_k). \tag{3.4}$$

We would like to find the hierarchy T that minimizes this cost.

- NP-hardness: A consequence of the above discussion is that the optimum tree can always be assumed to be binary. However, still the problem is computationally hard. Dasgupta showed via a reduction from not-all-equal SAT that optimizing for his objective is NP-hard.

- Recovery of Planted Partitions: In the case the graph has disconnected components, one can prove that the optimum tree will first split apart these disconnected components. This agrees with intuition as they encode different clusters. A strengthening of this argument was also given under a probabilistic generative model for the input graph, the Stochastic Block Model. For more details see [43]. Moreover, an even stronger version of the recovery result under planted partitions, is the one in [39] where they define a notion of Hierarchical Stochastic Block Model (HSBM). This is a probabilistic model suitable to capture hierarchical structures; they assume that the ground-truth is given by a binary tree (similar to assuming the ground truth is a balanced bipartition in the standard stochastic block model) and the uncertainty for the presence of edges is modeled under the assumption that for two nodes their probability of being connected or not, only depends on their lowest common ancestor in the ground-truth tree. They show that for instances drawn from the HSBM, one can achieve better approximations than what is possible in the worst-case and also that Dasgupta's optimizer recovers (almost) the ground truth.

- Axiomatic Approach to HC: In [39], they also take an axiomatic approach to HC objectives, trying to formulate specific desiderata that should be satisfied by any "reasonable" objective. For example, one of these desiderata is that whenever the input similarities are generated from a ground-truth instance (that they define based on the notion of *ultrametrics*), then the optimizer of 3.1 should coincide with the ground-truth. An interesting condition that arises in their framework as a necessary property of such "reasonable" HC objectives, is the fact that when the input is a unit-weight clique, then every tree has the same cost exactly. In some sense, this captures that cliques have no hierarchy to be found.

Chapter 4

Black-box Approximations for Hierarchical Clustering

Nice theorem; it seems that we now have two proofs for it, and one counterexample.

-Telemachos

This chapter is the main part of the thesis where we study 3 different global objectives for Hierarchical Clustering from a worst-case analysis perspective. We will devise approximation algorithms based on standard graph partitioning tools like Sparsest Cut or Balanced Cut, we will review certain related work about Average-Linkage and also give several hardness results.

4.1 Minimizing HC cost: Graph Cuts and Spreading Metrics

HC via Sparsest Cut and Balanced Cut

Given a graph G with weighted edges, we have seen that it is NP-hard to find a tree that minimizes $\text{cost}(T)$. We now consider top-down heuristics that begin by choosing a split $V \to (S, V \setminus S)$ according to some criterion, and then recurse on each half. What is a suitable split criterion?

The cost of a split $(S, V \setminus S)$ is $|V| \cdot w(S, V \setminus S)$. Ideally, we would like to shrink the node set substantially in the process, since this reduces the multiplier on subsequent splits. For a node chosen at random, the expected amount by which its cluster shrinks as a result of the split is

$$\frac{|S|}{|V|} \cdot |V \setminus S| + \frac{|V \setminus S|}{|V|} \cdot |S| = \frac{2|S||V \setminus S|}{|V|}.$$

A natural greedy criterion would therefore be to choose the split that yields the maximum shrinkage per unit cost, or equivalently, the minimum ratio

$$\frac{w(S, V \setminus S)}{|S| \cdot |V \setminus S|}.$$

This is known as the *sparsest cut* and has been studied intensively in a wide range of contexts. Although it is NP-hard to find an optimal cut, a variety of good approximation algorithms have been developed [73, 9]. We will assume simply that we have a heuristic whose approximation ratio, on graphs of n nodes, is at most α_n times optimal, for some positive nondecreasing sequence (α_n). For instance, the Leighton-Rao algorithm [73] has $\alpha_n = O(\log n)$ and the Arora, Rao, Vazirani algorithm [9] has $\alpha_n = O(\sqrt{\log n})$.

```
function MakeTree(V)
If $|V| = 1$:  return leaf containing the singleton element in $V$
Let $(S, V \setminus S)$ be an $\alpha_n$-approximation to the sparsest cut of $V$
LeftTree = MakeTree($S$)
RightTree = MakeTree($V \setminus S$)
Return [LeftTree, RightTree]
```

Figure 4.1: A top-down heuristic for finding HC that approximately minimizes cost(T).

The resulting hierarchical clustering algorithm is shown in Figure 4.1. The output can be thought of as a binary tree of the sequence of cuts performed by the algorithm. It is interesting that even though the cost function over hierarchies does not explicitly ask for sparse cuts ($\text{cost}_G(T)$ contains no ratios), the sparsest cut objective emerges organically when one adopts a greedy top-down approach to constructing the tree. We will now see that this algorithm returns a tree of cost at most $O(\alpha_n)$ times the optimal.

4.1.1 Recursive Sparsest Cut

Initially, Dasgupta [43] showed that this simple top-down recursive Sparsest Cut (RSC) heuristic, that uses an α_n-approximation algorithm for uniform sparsest cut gives an approximation of $O(\alpha_n \log n)$ for hierarchical clustering.

In this section, by drawing inspiration from our SDP construction and analysis presented later in Section 4.1.4, we present an improved analysis for this simple heuristic, dropping the $\log n$ factor and showing that it actually yields an $O(\alpha_n)$ approximation. This is satisfying since any improvement for Sparsest Cut would immediately yield a better approximation result for hierarchical clustering.

Analysis of RSC heuristic

Let the given graph be $G = (V, E)$. We suppose for clarity of presentation that it is unweighted; the analysis applies directly to weighted graphs and later, we see how to generalize it for more general cost functions. Let OPT be the optimal solution for hierarchical clustering (we abuse notation slightly by using OPT to denote both the solution as well as its objective function value). Let $\mathrm{OPT}(t)$ be the maximal clusters in OPT of size at most t. Note that $\mathrm{OPT}(t)$ is a partition of V.

We denote $E_{\mathrm{OPT}}(t)$ the edges that are cut in $\mathrm{OPT}(t)$, i.e. edges with end points in different clusters in $\mathrm{OPT}(t)$. For convenience, we also define $E_{\mathrm{OPT}}(0) \triangleq E$ (see Figure 4.2).

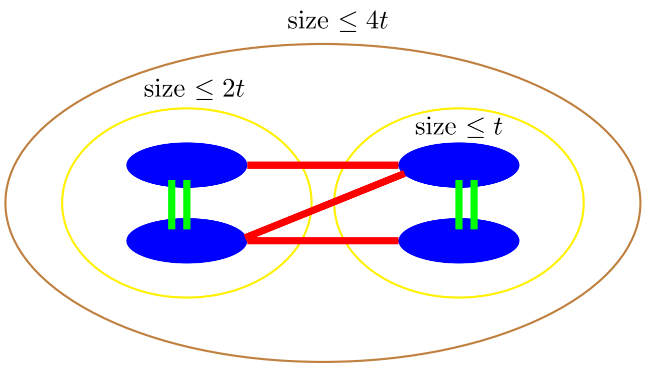

Figure 4.2: Illustration for the sets $E_{\mathrm{OPT}}(t)$, $E_{\mathrm{OPT}}(2t)$, $E_{\mathrm{OPT}}(4t)$. Looking at clusters that are maximal and of size at most t (filled in blue), $2t$ (yellow border), $4t$ (brown border), we have that the red edges belong to $E_{\mathrm{OPT}}(2t)$, whereas in $E_{\mathrm{OPT}}(t)$, we have the red and the green edges. Any edge that goes out of the big cluster of size $\leq 4t$ (brown) contributes to all terms $E_{\mathrm{OPT}}(t)$, $E_{\mathrm{OPT}}(2t)$, $E_{\mathrm{OPT}}(4t)$.

Claim 4.1.1 $\mathrm{OPT} = \sum_{t=0}^{n-1} |E_{\mathrm{OPT}}(t)|$.

Proof. Consider any edge $(u, v) \in E$. Suppose that the size of the minimal cluster in OPT that contains both u and v is r. Then the contribution of (u, v) to the LHS is r. On the other hand, $(u, v) \in E_{\mathrm{OPT}}(t)$ for all $t \in \{0, \dots, r-1\}$. Hence the contribution to the RHS is also r. ∎

It will be convenient to use the following bound that is directly implied by the above claim:

$$2 \cdot \mathrm{OPT} = 2 \cdot \sum_{t=0}^{n-1} |E_{\mathrm{OPT}}(t)| \geq \sum_{t=0}^{n} E_{\mathrm{OPT}}(\lfloor t/2 \rfloor) \tag{4.1}$$

Let's look at a cluster A with size $|A| = r$ in the solution produced by RSC. Using a sparsest cut approximation algorithm, we create two clusters B_1, B_2 with sizes $s, (r - s)$ respectively, with B_1 being the smaller, i.e. $s \leq \lfloor r/2 \rfloor$. The contribution of this cut to the hierarchical clustering objective function is: $|E(B_1, B_2)| \cdot r$. We basically want to charge this cost to $\mathrm{OPT}(\lfloor r/2 \rfloor)$ and for that we first observe that the edges cut in OPT($\lfloor r/2 \rfloor$), when restricted to the cluster A (i.e. having both endpoints in A), satisfy the following:

$$s \cdot |E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A| \leq \sum_{t=r-s+1}^{r} |E_{\text{OPT}}(\lfloor t/2 \rfloor) \cap A| \tag{4.2}$$

This follows easily from the fact that $|E_{\text{OPT}}(t) \cap A| \leq |E_{\text{OPT}}(t-1) \cap A|$ (by definition, the smaller the value of t, the more edges are cut). Now in order to explain our charging scheme, let's look at the partition $A_1, ..., A_k$ induced inside the cluster A by OPT($\lfloor r/2 \rfloor$) $\cap A$, where by design the size of each $|A_i| = \gamma_i |A|$, $\gamma_i \leq 1/2$. We have:

$$\frac{|E(A_i, A \setminus A_i)|}{|A_i||A \setminus A_i|} = \frac{|E(A_i, A \setminus A_i)|}{\gamma_i(1-\gamma_i)r^2}, \forall i \in \{1, \dots, k\}$$

We take the minimum over all i (an upper bound on the sparsest cut in A) and we have:

$$\min_i \frac{|E(A_i, A \setminus A_i)|}{\gamma_i(1-\gamma_i)r^2} \leq \frac{\sum_i |E(A_i, A \setminus A_i)|}{\sum_i \gamma_i(1-\gamma_i)r^2} \leq 2 \cdot \frac{|E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A|}{r^2/2} = 4 \cdot \frac{|E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A|}{r^2}$$

The first inequality above, trivially follows by definition for the minimum and the second inequality holds because $\sum_{i=1}^{k} \gamma_i = 1, \sum_{i=1}^{k} \gamma_i^2 \leq 1/2$ (note that all $\gamma_i \leq 1/2$) and the factor of 2 is introduced since we double counted every edge. We partition A using an α_r-approximation for sparsest cut and we get (since $\alpha_r \leq \alpha_n$):

$$\frac{|E(B_1, B_2)|}{s(r-s)} \leq \alpha_n \cdot \frac{4}{r^2} \cdot |E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A|$$

since the RHS (without the α_n factor) is an upper bound of the optimal sparsest cut value. The contribution of this step to the hierarchical clustering objective function is:

$$r|E(B_1, B_2)| \leq \frac{4\alpha_n s(r-s)}{r} \cdot |E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A| \leq 4\alpha_n s \cdot |E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A| \tag{4.3}$$

We claim the following:

Claim 4.1.2 *Let A be a cluster of size r_A in our hierarchical clustering solution, that we split into 2 pieces (B_1, B_2) of sizes $s_A, r_A - s_A$ respectively with $|B_1| \leq |B_2|$ (so s_A stands for the size of the small piece B_1 after we split A). Then, summing over all clusters A we get:*

$$\sum_A \sum_{t=r_A-s_A+1}^{r_A} |E_{\text{OPT}}(\lfloor t/2 \rfloor) \cap A| \leq \sum_{t=0}^{n} |E_{\text{OPT}}(\lfloor t/2 \rfloor)|$$

Proof. For a fixed value of t and A, the LHS is: $|E_{\text{OPT}}(\lfloor t/2 \rfloor) \cap A|$. Consider which clusters A contribute such a term to the LHS. From the fact that $r_A - s_A + 1 \leq t \leq r_A$, we need to have that

$|B_2| < t$ and since B_2 is the larger piece that was created when A was split, we deduce that A is a **minimal** cluster of size $|A| \geq t > |B_2| \geq |B_1|$, i.e. if both A's children are of size less than t, then this cluster A contributes such a term. The set of all such A form a disjoint partition of V because of the definition for minimality (in order for them to overlap in the hierarchical clustering, one of them needs to be ancestor of the other and this cannot happen because of minimality). Since $E_{\text{OPT}}(\lfloor t/2 \rfloor) \cap A$ for all such A forms a disjoint partition of $E_{\text{OPT}}(\lfloor t/2 \rfloor)$, the claim follows by summing up over all t. ▮

Theorem 4.1.3 *Given an unweighted graph G, the Recursive Sparsest Cut algorithm achieves an $O(\alpha_n)$ approximation for the hierarchical clustering problem.*

Proof. The proof follows easily by combining (4.1), (4.2), (4.3), Claim 4.1.2 and summing over all clusters A created by RSC. In particular, we get the following result for the overall performance guarantee:

$$cost_{RSC} = \sum_A r_A \cdot |E(B_1, B_2)| \leq \sum_A 4\alpha_n s_A |E_{\text{OPT}}(\lfloor r_A/2 \rfloor) \cap A| \leq$$

$$\leq 4\alpha_n \textstyle\sum_A \sum_{t=r_A - s_A+1}^{r_A} |E_{\text{OPT}}(\lfloor t/2 \rfloor) \cap A| \leq 4\alpha_n \sum_{t=1}^{n} |E_{\text{OPT}}(\lfloor t/2 \rfloor)| \leq 8\alpha_n \cdot \text{OPT}. ▮$$

4.1.2 Using Balanced Cut instead of Sparsest Cut

We are going to give an analysis similar to the above, but instead of using the Sparsest Cut, we are going to use c-BALANCED CUT as a black box. We will get the same approximation factor and at the end, we will have a brief discussion on comparing the two approaches.

We will follow the same notation as above and we will use some of the facts and inequalities we previously proved about $E_{\text{OPT}}(t)$. Suppose again that we have a cluster A of size r and take $|E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A|$. The important observation here is that the partition $A_1, ..., A_k$ induced inside the cluster A by $\text{OPT}(\lfloor r/2 \rfloor) \cap A$ (where by design the size of each $|A_i| = \gamma_i |A|$, $\gamma_i \leq 1/2$), can be separated into two groups, let's say (C_1, C_2) such that $r/3 \leq |C_1|, |C_2| \leq 2r/3$ (see Figure 4.3). In other words we can demonstrate a c-BALANCED CUT for $c = 1/3$. We know that:

$$|E(C_1, C_2)| \leq |E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A|$$

since we cut fewer edges when creating C_1, C_2.

At this point, we can invoke an α-pseudo-approximation algorithm for Balanced Cut [8] for the cluster A (let OPT_{BC} be the optimal cost of partitioning A into two clusters with sizes at least $r/3$) and we can get a partition into (B_1, B_2) with $c'r \leq |B_1|, |B_2| \leq (1-c')r$, (the constant c' depends only on c and $c' < 1/3$, that's why we call it pseudo-approximation) with the cost guarantee of:

$$|E(B_1, B_2)| \leq \alpha \cdot \text{OPT}_{BC} \leq \alpha \cdot |E(C_1, C_2)| \leq$$

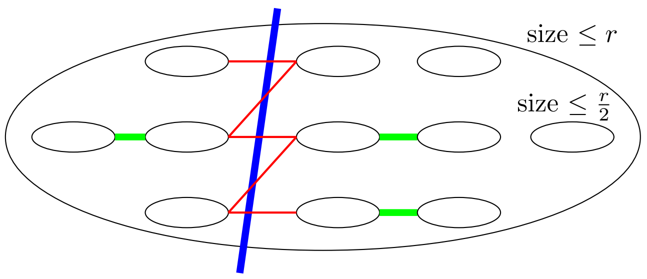

Figure 4.3: An illustration of $|E_{\text{OPT}}(r/2) \cap A|$ for a cluster A with size r. The small clusters have size $\leq \frac{r}{2}$ and the induced partition can be separated into two groups (C_1, C_2) that are $\frac{1}{3} : \frac{2}{3}$ balanced. The edges between those two groups (red) are also present in $E_{\text{OPT}}(r/2)$ (red and green).

$$\leq \alpha \cdot |E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A|$$

Looking at the cost of the hierarchical clustering which is $r \cdot |E(B_1, B_2)|$ and charging it as we did before to $s \cdot |E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A|$ where now $s \geq c'r \implies r \leq s/c'$ we get:

$$r \cdot |E(B_1, B_2)| \leq \alpha r \cdot |E(C_1, C_2)| \leq \alpha \frac{s}{c'} \cdot |E(C_1, C_2)| \leq$$

$$\leq \alpha \frac{s}{c'} \cdot |E_{\text{OPT}}(\lfloor r/2 \rfloor) \cap A| \leq \frac{\alpha}{c'} \sum_{t=r-s+1}^{r} |E_{\text{OPT}}(\lfloor t/2 \rfloor) \cap A|$$

Finally, all we have to do is sum up over all the clusters A (now in the summation we should write r_A, s_A instead of just r, s, since there is dependence in A) produced by the recursive Balanced Cut algorithm for Hierarchical Clustering and we get that we can approximate the HC objective function up to $O(\sqrt{\log n})$ (here we just substituted the best known value for the approximation guarantee α).

Remark 3 *The results for Balanced Cut generalize for the weighted case and for general cost functions. The reason we were interested in Balanced Cut is twofold: Firstly, recall that the running time for Sparsest Cut and Balanced Cut on a graph with n nodes and m edges is $\tilde{O}(m + n^{1+\epsilon})$. Now, the running time of Recursive Sparsest Cut might be worse off by a factor of n (you could have very unbalanced splits at every step) in the worst case. However, the running time for Recursive Balanced Cut is still $\tilde{O}(m + n^{1+\epsilon})$, because at each step you ensure that you have similar sized pieces and thus you make progress in the recursion. Secondly, we know that an α_n-approximation for Sparsest Cut yields an $O(\alpha_n)$-approximation for Balanced Cut, but not the other way. This means that if we could find a better Balanced Cut algorithm without improving Sparsest Cut, we could still use it for better Hierarchical Clustering.*

4.1.3 Generalized HC Cost Function Approximation

In the original [43] paper introducing the objective function of hierarchical clustering, Dasgupta also considered the more general cost function: $cost_G(T) = \sum_{ij \in E} w_{ij} f(|\textbf{leaves}(T[i \vee j])|)$, where f is defined on the non-negative reals, is strictly increasing, and has $f(0) = 0$ (e.g. $f(x) = \ln(1+x)$ or $f(x) = x^2$). For this more general cost function, he proved that a slightly modified greedy top-down heuristic (using $\frac{w(S,V\setminus S)}{\min((f|S|),f(|V\setminus S|))}$ with $\frac{1}{3}|V| \le |S| \le \frac{2}{3}|V|$, instead of Sparsest Cuts) continues to yield an $O(\alpha_n \cdot \log n \cdot c_f)$ approximation[1], where $c_f \triangleq \max_{1 \le n' \le n} \frac{f(n')}{f(n'/3)}$). Now, we analyze the previous RSC algorithm (with no modifications), but in the case of a weighted graph G and when we are trying to optimize the generalized cost function.

We again make the natural assumptions that the function f acting on the number of leaves in subtrees, is defined on the nonnegative reals, is strictly increasing and $f(0) = 0$ (also see Remark 4). We also define: $c_f \triangleq \max_{1 \le n' \le n} \frac{f(n')}{f(\lfloor n'/2 \rfloor) - f(\lfloor n'/4 \rfloor)}$. For what follows, we abuse notation slightly for ease of presentation and write $r/2, r/4$ etc. instead of $\lfloor r/2 \rfloor, \lfloor r/4 \rfloor$ etc. As in the simple unweighted case, we use here the same definitions for OPT and $E_{\text{OPT}}(t)$. Let $w(E_{\text{OPT}}(t))$ denote the total weight of the edges $E_{\text{OPT}}(t)$, i.e. the edges cut by OPT at level t, where we define $w(\emptyset) = 0$ and we also define $g(t) \triangleq f(t+1) - f(t)$. We note that $\sum_{t=0}^{r-1} g(t) = f(r) - f(0) = f(r)$.

Claim 4.1.4 $\sum_{t=0}^{n-1} w(E_{\text{OPT}}(t)) \cdot g(t) = \text{OPT}$

Proof. We will prove that the contributions of an edge $e = (u,v)$ to the LHS and RHS are equal. Let A ($|A| = r_e$) be the minimal cluster in the optimal solution that contains both u, v. The contribution of e to the RHS is: $w_e \cdot f(r_e)$. As for the contribution to the LHS, since A is minimal and $|A| = r_e$, we deduce that $e \in \text{OPT}(t), \forall t < r_e$. Also for levels $t \ge r_e$ we have $e \in A$ or some superset of A and thus $e \notin \text{OPT}(t)$ Hence the contribution to the LHS is: $w_e \cdot \sum_{t=0}^{r_e-1} g(t) = w_e \cdot f(r_e)$. ▮

Focus on a cluster A ($|A| = r$) in the solution produced by the algorithm. Let $cut(A)$ denote the edges in A cut by partitioning A. This contributes $w(cut(A)) \cdot f(r)$ to the objective. We will charge our cost using the following quantity related to the optimum solution: $\sum_{t=r/4}^{r/2-1} w(E_{\text{OPT}}(t) \cap A) \cdot g(t)$.

For that, we look at $\text{OPT}(r/2) \cap A$ and let's say that clusters $A_1, A_2, ..., A_k$ are induced by this partition, each being of size $|A_i| = \gamma_i |A| \le |A|/2 = r/2$ ($\gamma_i \le 1/2$). Then,

$$SC(A) \le \frac{\sum_i w(A_i, A \setminus A_i)}{r^2 \sum_i \gamma_i (1-\gamma_i)} \le \frac{2 \cdot w(E_{\text{OPT}}(r/2) \cap A)}{r^2 \cdot 1/2}$$

where $SC(A)$ is the optimum sparsest cut (value) for A. Since we used an α_n-approximation,

$$\frac{w(cut(A))}{s(r-s)} \le \alpha_n SC(A) \le \frac{4\alpha_n (w(E_{\text{OPT}}(r/2) \cap A)}{r^2} \implies$$
$$w(cut(A)) f(r) \le \frac{4\alpha_n s}{r} w(E_{\text{OPT}}(r/2) \cap A) f(r) \tag{4.4}$$

Since $w(E_{\text{OPT}}(t) \cap A) \geq w(E_{\text{OPT}}(t+1) \cap A)$, we have:

$$\sum_{t=r/4}^{r/2-1} w(E_{\text{OPT}}(t) \cap A) g(t) \geq w(E_{\text{OPT}}(r/2) \cap A) \sum_{t=r/4}^{r/2-1} g(t) = (f(r/2) - f(r/4)) w(E_{\text{OPT}}(r/2) \cap A) \tag{4.5}$$

Using equations (4.4), (4.5), we get that:

$$w(cut(A)) \cdot f(r) \leq 4\alpha_n \cdot \frac{s}{r} \cdot \frac{f(r)}{f(r/2) - f(r/4)} \cdot \sum_{t=r/4}^{r/2-1} w(E_{\text{OPT}}(t) \cap A) \cdot g(t) \tag{4.6}$$

We now sum up the cost contributions of all clusters created in our hierarchical clustering solution. Let $s(A)$ be the size of the smaller piece produced in partitioning A.

$$cost_{RSC} = \sum_A w(cut(A)) \cdot f(|A|) \leq 4\alpha_n \cdot c_f \sum_A \frac{s(A)}{|A|} \sum_{t=|A|/4}^{|A|/2-1} w(E_{\text{OPT}}(t) \cap A) \cdot g(t) \tag{4.7}$$

To complete our argument we need to make the comparison between OPT which is: $\sum_{t=0}^{n-1} w(E_{\text{OPT}}(t)) \cdot g(t)$ and the sum

$$\sum_A \frac{s(A)}{|A|} \sum_{t=|A|/4}^{|A|/2-1} w(E_{\text{OPT}}(t) \cap A) \cdot g(t) \tag{4.8}$$

where the first summation goes over all clusters A in the solution we produce.

Claim 4.1.5 $\sum_A \frac{s(A)}{|A|} \sum_{t=|A|/4}^{|A|/2-1} w(E_{\text{OPT}}(t) \cap A) \cdot g(t) \leq 2 \cdot \sum_{t=0}^{n-1} w(E_{\text{OPT}}(t)) \cdot g(t)$

Proof. Consider some edge $e = (u, v) \in E_{\text{OPT}}(t)$. Focus on sets A in the solution produced such that $e \in E_{\text{OPT}}(t) \cap A$ so that e contributes to the term $\sum_{t=|A|/4}^{|A|/2-1} w(E_{\text{OPT}}(t) \cap A) \cdot g(t)$ in the LHS. For all such clusters A, we need to have: $|A|/4 \leq t < |A|/2 \implies 2t < |A| \leq 4t$.

Let $A_1, A_2, ..., A_{k-1}$ be the sets for which the term $w(E_{\text{OPT}}(t) \cap A)$ appears: A_1 is the largest cluster (satisfying $2t < |A_1| \leq 4t$) that contains the edge $e = (u, v)$ and when split we call its larger piece A_2 (again this set contains e) etc., A_{k-1} is the last set for which the term appears and (u, v) does not appear in A_k (A_k is the larger piece of the two that we got when we partitioned A_{k-1}). We have:

$$\sum_{i=1}^{k-1} \frac{s(A_i)}{|A_i|} = \frac{|A_1| - |A_2|}{|A_1|} + \frac{|A_2| - |A_3|}{|A_2|} + ... + \frac{|A_{k-1}| - |A_k|}{|A_{k-1}|} \leq \frac{\sum_{i=1}^{k-1} |A_i| - |A_{i+1}|}{\min_i |A_i|} \leq \frac{|A_1|}{2t} \leq 2. \tag{4.9}$$

(the constant can be optimized, but it does not change the asymptotic bound). Thus the contribution of every edge $e \in E_{\text{OPT}}(t)$ to the LHS is at most $2w_e g(t)$. Note that this is exactly the contribution to the RHS. This establishes the claim. ∎

Theorem 4.1.6 *RSC achieves an $O(c_f \cdot \alpha_n)$ approximation of the generalized objective function for Hierarchical Clustering.*

Proof. We will prove the theorem by combining (4.7), Claim 4.1.5 and Claim 4.1.4. In particular, from (4.7),

$$cost_{RSC} \leq 4\alpha_n \cdot c_f \sum_{A} \frac{s(A)}{|A|} \sum_{t=|A|/4}^{|A|/2-1} w(E_{\text{OPT}}(t) \cap A) \cdot g(t)$$

Combining the above with Claim 4.1.5, we get that the total cost of the RSC is at most:

$$cost_{RSC} \leq 8c_f\alpha_n \cdot \sum_{t=0}^{n-1} w(E_{\text{OPT}}(t)) \cdot g(t)$$

Finally, using the Claim 4.1.4 we get: $cost_{RSC} \leq (8c_f\alpha_n) \cdot \text{OPT} = O(c_f \cdot \alpha_n) \cdot \text{OPT}$. ∎

Remark 4 *In order for our guarantee to be useful, we need c_f to be a constant (or a slowly growing quantity). This would mean that f is polynomially growing. We observe that in the case where the function f is exponentially growing then our guarantee is not interesting (and in fact we may need to use a different strategy than RSC) and in the case f is logarithmic, then we would get a factor $\approx O(\alpha_n \log n)$ approximation, which is the same guarantee as [43].*

4.1.4 Convex Relaxations for HC

In this section, we present our SDP relaxation for HC based on spreading metrics, we point out its relation with the SDP relaxation of *k-balanced partitioning* in [67] and we prove that it is an $O(\sqrt{\log n})$ approximation for both the simple and the generalized cost function.

Writing the SDP

We view a hierarchical clustering of n data points as a collection of partitions of the data, one for each level $t = n-1, \ldots, 0$ (where level 0 is identical to level 1, for convenience). The partition for a particular level t satisfies the property that every cluster has size at most t; additionally, for every vertex i, the cluster containing vertex i at level t is the maximal cluster in the hierarchy with size at most t. The partition at level $(t-1)$ is a refinement of the partition at level t. Note that the partition corresponding to $t = 1$ must consist of n singleton clusters. We represent the partition at level t by the set of variables x_{ij}^t, $i, j \in V$, where $x_{ij}^t = 1$ if i and j are in different clusters in the partition at level t and $x_{ij}^t = 0$ if i and j are in the same cluster. We point out some properties of these variables x_{ij}^t satisfied by an integer solution corresponding to an actual hierarchical clustering:

1. **refinement:** $x_{ij}^t \leq x_{ij}^{t-1}$. If i and j are separated at level t, then they continue to be separated at level $t-1$.

2. **triangle inequality:** $x_{ij}^t + x_{jk}^t \geq x_{ik}^t$. In the clustering at level t, if i and j are in the same cluster, j and k are in the same cluster, then i and k are in the same cluster.

3. ℓ_2^2 **metric:** The triangle inequality condition implies that x_{ij}^t is a metric. Further, we can associate unit vectors v_i^t with vertices i at level t such that $x_{ij}^t = \frac{1}{2}||v_i^t - v_j^t||_2^2$. In order to do this, all vertices in the same cluster at level t are assigned the same vector, and vertices in different clusters are assigned orthogonal vectors.

4. **spreading:** $\sum_j x_{ij}^t \geq n-t$. For the clustering at level t, there are at most t vertices in the same cluster as i. Hence there are at least $n-t$ vertices in different clusters. For each such vertex j, $x_{ij}^t = 1$ implying the inequality.

5. **cluster size:** The size of the smallest cluster in the hierarchy containing both vertices i and j is given by $1 + \sum_{t=1}^{n-1} x_{ij}^t$. Suppose C is the smallest cluster containing both i and j. Then for $t \geq |C|$, the partition at level t must contain C or some superset of C. Hence $x_{ij}^t = 0$ for $t \geq |C|$. For $t < |C|$, the clustering at level t must have i and j in different clusters, hence $x_{ij}^t = 1$. Hence $\sum_{t=1}^{n-1} x_{ij}^t = |C| - 1$. Finally, we can write the SDP relaxation SDP-HC as follows:

$$\min \sum_{t=0}^{n-1} \sum_{ij \in E} x_{ij}^t w_{ij} = \min \sum_{t=0}^{n-1} \sum_{ij \in E} \frac{1}{2} \|v_i^t - v_j^t\|_2^2 w_{ij},$$

$$\text{such that:} \quad x_{ij}^t \leq x_{ij}^{t-1}, \quad t = n-1, n-2, \ldots 1 \tag{SDP-HC}$$

$$x_{ij}^0 = 1, \forall i,j \in V \text{ and } x_{ij}^t \leq 1, \forall i,j,t$$

$$x_{ij}^t = \frac{1}{2}\|v_i^t - v_j^t\|_2^2 \text{ and } \|v_i^t\|_2^2 = 1, \forall i \in V$$

$$x_{ij}^t \leq x_{jk}^t + x_{ik}^t, \forall i,j,k \in V, \ \forall t \text{ and } \sum_j x_{ij}^t \geq n-t, \forall i,t$$

It is easy to see that an optimal solution to SDP-HC can be computed in polynomial time (see for example the arguments in [67]). By the preceding discussion, we have shown that SDP-HC is a valid relaxation for HC:

Lemma 4.1.7 *The value of an optimal solution to SDP-HC can be computed in polynomial time, and gives a lower bound on the cost of an optimal solution to the hierarchical clustering problem.*

Connections of SDP-HC with Balanced Partitioning

The authors of [67] write an SDP relaxation for the problem of k-Balanced Partitioning (k-BP) which was the following (SDP-k-BP):

$$\min \sum_{ij\in E} w_{ij}\cdot \frac{1}{2}\|v_i - v_j\|_2^2, \tag{SDP-k-BP}$$

$$\text{such that: } \|v_i - v_j\|_2^2 + \|v_j - v_k\|_2^2 \geq \|v_i - v_k\|_2^2,\ \forall i,j,k\in V$$

$$\sum_{j\in S}\frac{1}{2}\|v_i - v_j\|_2^2 \geq |S| - \frac{n}{k},\ \forall S\subseteq V, i\in S$$

Their result was that the above relaxation is an $O(\sqrt{\log k \log n})$ approximation (bi-criteria $\nu = 2$) algorithm for k-BP, that will create pieces of size at most $2n/k$.

Claim 4.1.8 *Let A be a cluster of size r. SDP-HC solution restricted to set A, at level $t = r/4$ is a valid solution for k-balanced partitioning based on the SDP-k-BP relaxation, where $k = 4$.*

Proof. First of all, we have that:

$$\text{SDP}_A(t) \triangleq \sum_{ij\in E, i,j\in A} x_{ij}^t w_{ij} = \sum_{ij\in E,i,j\in A}\frac{1}{2}\|v_i^t - v_j^t\|_2^2 w_{ij}$$

Basically, we take the SDP-HC solution for the entire instance and we focus on some part of it, i.e. we focus on the terms of the objective function that are relevant for the cluster A.

To prove the claim all we need to do is to compare the set of constraints imposed by SDP-HC and SDP-k-BP. In SDP-HC, we have some additional constraints: $x_{ij}^t \leq 1$ and $v_i^t \leq 1$, but that is fine since imposing extra constraints just makes a stricter relaxation. Now let's look at the spreading constraints: In SDP-HC we have $\sum_j x_{ij}^t \geq n - t \implies \sum_{j\in S} x_{ij}^t \geq |S| - t$ which is basically the SDP-k-BP spreading constraints. Thus, by looking at the SDP-HC solution restricted to set A ($|A| = r$), at level $t = r/4$, we can get a valid 4-balanced partitioning solution of A. ▮

In order to produce a hierarchical clustering from the SDP solution, we recursively partition V in a top down fashion: while partitioning a cluster A, we use the SDP-HC solution restricted to set A at level $t = |A|/4$ as a valid solution for 4-balanced partitioning and invoke the algorithm of [67] as a black box. Let E_A be the edges cut by the algorithm when splitting cluster A. From the analysis of [67] , we get that (for us $k = 4$, so $\log k$ is constant):

$$w(E_A) \leq O(\sqrt{\log n})\cdot \text{SDP}_A(r/4) \tag{4.10}$$

and we partition A into pieces of size at most $\leq 2\cdot r/4 = r/2$ (bi-criteria). In the analysis that follows, we will use this result as a black box.

4.1.5 $O(\sqrt{\log n})$ approximation for Hierarchical Clustering

Now we go on to see that the integrality gap of our SDP-HC is $O(\sqrt{\log n})$. Let r_A be the size of a cluster A in the solution produced. For our charging argument, we observe that we pay $r_A \cdot w(E_A)$ where E_A are the edges cut by the KNS [67] algorithm when partitioning A. We will charge this cost to $\sum_{t=r_A/8+1}^{r_A/4} \mathrm{SDP}_A(t) \geq \frac{r_A}{8}\mathrm{SDP}_A(r_A/4)$ (note that as t decreases more edges are cut). Thus, using [67], the total cost of the solution produced (where $cost_{HC}$ is the cost of the rounding algorithm's solution and r_A depends on A):

$$cost_{HC} = \sum_A r_A \cdot w(E_A) \leq O(\sqrt{\log n}) \sum_A \sum_{t=r_A/8+1}^{r_A/4} \mathrm{SDP}_A(t) \tag{4.11}$$

Claim 4.1.9 $\sum_A \sum_{t=|A|/8+1}^{|A|/4} \mathrm{SDP}_A(t) \leq O$*(SDP-HC).*

Proof. The flavor of this analysis is similar to our RSC result from Section 4.1.1. Let's look at an edge $e = (u, v)$ at a fixed level t. For which sets A do we get the term $\mathrm{SDP}_A(t)$ (i.e. both endpoints $u, v \in A$) ? Since $t \in \big(|A|/8, |A|/4\big] \implies 4t \leq |A| < 8t$. There can be at most one such $|A|$ containing both u, v, so LHS is charged only once (of course the RHS is charged $x_{ij}^t w_{ij}$ for that edge). To see why A is unique, suppose we had two such clusters $|A_1|, |A_2|$ that both contained u, v with their sizes $|A_1|, |A_2| \in [4t, 8t)$. Since we have a hierarchical decomposition, one of A_1, A_2 is ancestor of the other. Let's say, wlog, A_1 is ancestor of A_2. But then, all of its descendants are of size below the range $[4t, 8t)$ due to the 4-partition, which is a contradiction. ▮

Remark 5 *In the above analysis, whenever we write $|A|/4$ we mean $\lfloor |A|/4 \rfloor$. However, this will not affect the result. Additionally, we used the bound O(SDP-HC), because some extra constants might be introduced whenever the set A is small ($|A| < 8$).*

Theorem 4.1.10 *The cost of the solution produced by the SDP-HC rounding algorithm is within a factor of $O(\sqrt{\log n})$ from the SDP value.*

Proof. Using Claim 4.1.9 and (4.11) we get that $cost_{HC} \leq O(\sqrt{\log n}) \cdot$SDP-HC. ▮

The case of the generalized cost function

Now, we consider the performance of SDP-HC-gen for the generalized cost function and we show essentially the same approximation guarantees. We note that the SDP-HC-gen is essentially the same as SDP-HC where each term in the objective function is multiplied by $g(t) = f(t+1) - f(t)$. Formally:

$$\min \sum_{t=0}^{n-1} g(t) \sum_{ij \in E} x_{ij}^t w_{ij} = \min \sum_{t=0}^{n-1} g(t) \sum_{ij \in E} \frac{1}{2}\|v_i^t - v_j^t\|_2^2 w_{ij} \tag{SDP-HC-gen}$$

We can easily prove the following claim for the generalized cost:

Claim 4.1.11 $\sum_A \sum_{t=|A|/8+1}^{|A|/4} \mathrm{SDP}_A(t) \cdot g(t) \leq O(\text{SDP-HC-gen})$.

Proof. The proof of the claim is identical to the proof we gave above for Claim 4.1.9 with only difference being that we need to multiply by $g(t)$ each term. ∎

Theorem 4.1.12 *The cost of the solution produced by the SDP-HC-gen rounding algorithm is within a factor of $O(\sqrt{\log n} \cdot c_f)$ from the SDP value where $c_f \triangleq \max_{r\in\{1,\dots,n\}} \frac{f(r)}{f(r/4)-f(r/8)}$.*

Proof. Let A be a cluster of size $|A| = r$ and let $g(t) = f(t+1) - f(t)$. We want to compare the cost of OPT for splitting A with our solution $\mathrm{SDP}_A(t)$ for levels $t = r/8+1, ..., r/4$. Using (4.10) (once again $cost_{HC}(A)$ is just the cost incurred by the rounding algorithm when we partitioned cluster A):

$$cost_{HC}(A) = f(r) \cdot w(E_A) \leq O(\sqrt{\log n}) f(r) \cdot \mathrm{SDP}_A(r/4) \leq$$

$$\leq O(\sqrt{\log n}) \frac{f(r)}{f(r/4) - f(r/8)} \sum_{t=r/8+1}^{r/4} \mathrm{SDP}_A(t) \cdot g(t) \leq O(\sqrt{\log n}) \cdot c_f \sum_{t=r/8+1}^{r/4} \mathrm{SDP}_A(t) \cdot g(t).$$

Using now Claim 4.1.11 and summing over all clusters A in the hierarchical clustering we get (for the sum we should substitute r by r_A): $cost_{HC} \leq O(\sqrt{\log n}) \cdot c_f \cdot \text{SDP-HC-gen}$.∎

Remark 6 *As in Remark 4, here f should be polynomially growing.*

4.1.6 An LP-based $O(\log n)$ approximation via spreading metrics

In this section, we present an approximation algorithm with ratio $O(\log n)$ based on an LP relaxation. The key ingredients for our purposes are the spreading metrics paradigm of [48] and a graph decomposition lemma by Bartal ([18]).

A similar analysis and a $O(\log n)$ approximation guarantee was obtained independently from Roy and Pokutta [85] who derived a similar Linear Programming formulation based on ultrametrics and using the idea of sphere growing ([68, 53]) from graph partitioning for the rounding step. They also performed a series of experiments (on synthetic and real data) which suggest that the hierarchies found by using their LP formulation and their rounding algorithm are close to the true optimum (according to the hierarchical clustering cost function) and often have better projections into flat clusters than the standard linkage based algorithms or the k-means algorithm.

Bartal's decomposition

Bartal presented a graph decomposition lemma and used it in order to prove an $O(\log n)$ approximation guarantee for the spreading metrics paradigm in undirected graphs; thus, he improved the results for many problems considered in [48]. How does the decomposition work? At a high level, it tries to find a low diameter cluster within the graph, such that the weight of the cut created is small with respect to the weight of the cluster. The decomposition is essentially based on a careful implementation of the decomposition of [53]. In what follows, we state Bartal's improved approximation guarantee, summarized in the following theorem, and then briefly highlight the main steps in achieving it; proofs can be found in [18].

Theorem 4.1.13 *There exists an $O(\log n)$ approximation for problems in the spreading metrics paradigm.*

We first need to introduce some notation, before explaining the main steps of the proof. Let $G = (V, E)$ be an undirected graph with two weight functions $w, l : E \to \mathbb{R}^+$. We interpret $l(e)$ to be the length of the edge e, and the distance $d(u, v)$ between pairs of vertices u, v in the graph, is determined by the length of the shortest path between them. Given a subset $S \subseteq V$, $G(S)$ denotes the subgraph of G induced by S. Given partition $(S, \bar{S})$, let $\Gamma(S) = \{(u, v) \in E; u \in S, v \in \bar{S}\}$ and $cut(S) = \sum_{e\in\Gamma(S)} w(e)$. Given a subgraph $H = (V_H, E_H)$ of G, let d_H denote the distance in H, let $\Delta(H)$ denote the diameter of H, and $\Delta = \Delta(G)$. We also define the volume of H, $\phi(H) = \sum_{e\in E_H} w(e)l(e)$.

Informally, the first step needed is to find in G, a partition $(S, \bar{S})$ which is "good", i.e. with low cut value $cut(S)$ with respect to a generalized notion of its volume. The decomposition becomes useful when it is applied recursively. Note, that this is particularly important for our main application which is hierarchical clustering. This gives rise to a recursive approach where we find a good cut, creating two subgraphs and then recuse on each of them. Towards this direction, for the second step needed, we focus on applications which are associated with a cost function *cost* over subgraphs $\widehat{G}$ of G, that is nonnegative, 0 on singletons and obeys the following natural recursion rule:

$$cost(\widehat{G}) \leq cost(\widehat{G}(S)) + cost(\widehat{G}(\bar{S})) + \Delta(\widehat{G}) \cdot cut(S). \tag{4.12}$$

Lemma 4.1.14 *Any cost function defined by the above recursion rule obeys $cost(G) \leq O(\log(\phi/\phi_0)) \cdot \phi(G)$, where $\phi = \phi(G)$ and ϕ_0 is the minimum value of $\phi(\widehat{G})$ on non-singleton subgraphs $\widehat{G}$.*

Finally, we can obtain a $O(\log n)$ approximation bound (depending only on n), by modifying the above lemma slightly, by associating a volume $\phi(G)/n$ with the nodes, like in [53]. This ensures that $\phi_0 \geq \phi(G)/n$ and by substituting we get what we want:

Lemma 4.1.15 *The function defined by the above recursion rule obeys $cost(G) \leq O(\log n) \cdot \phi(G)$.*

Now we turn our attention to the connection with the spreading metrics paradigm. Having the definition of a spreading metric in mind (see Section 4.1.6) and the previous three steps, we may obtain Theorem 4.1.13. For details see [18].

LP relaxation and the Spreading Metrics paradigm

We prove here that the hierarchical clustering objective function defined above falls into the *divide and conquer approximation algorithms via spreading metrics* paradigm of [48].

The spreading metric paradigm applies to minimization problems on undirected graphs $G = (V, E)$ with edge weights $w(e) \geq 1$. We also have an auxiliary graph H and a scaler function on subgraphs of H (e.g. size of the components of H). A decomposition tree T is a tree with nodes corresponding to non-overlapping subsets of V, forming a recursive partition of the nodes V. For a node t of T, we denote by V_t the set of vertices in the subtree rooted at t. Associated are the subgraphs G_t, H_t induced by V_t. Let F_t be the set of edges that connect vertices that belong to different children of t, and $w(F_t) = \sum_{e \in F_t} w(e)$. The cost of T is $cost(T) = \sum_{t \in T} scaler(H_t) \cdot w(F_t)$.

Definition 4.1.1 *A spreading metric is a function on the edges of the graph $l : E \to \mathbb{R}^+$ satisfying the following two properties:*

1. *Lower bound property: The volume of the graph $\sum_{e \in E} w(e)l(e)$ is a lower bound on the optimal cost.*
2. *Diameter property: For any $U \subseteq V$ and H_U, the subgraph of H induced by U, has diameter:*

$$\Delta(U) \geq scaler(H_U)$$

We closely follow the formulation in [48] for the Linear Arrangement problem, which also falls into the spreading metrics paradigm, but we make the necessary semantic changes. We need to show the divide and conquer applicability and the spreading metrics applicability of their result for our problem.

Firstly, to establish the divide and conquer applicability we consider any binary decomposition tree T that fully decomposes the problem (we normalize the edge weights by dividing with the minimum edge weight). Note that there is a $1-1$ correspondence between the leaves of T and the vertices of G. The solution to the hierarchical clustering problem that is represented by T is easily given by the cuts, in G, induced by the internal nodes of T. The cost of the tree T is: $cost_G(T) = \sum_{t \in T} |V_t| w(F_t)$, where V_t and F_t are the set of vertices and cut corresponding to the tree node t and $w(F_t)$ is the total weight of the edges cut at this internal node t. We need to show that this cost bounds the cost of solutions built up from T. For this we prove that for every tree node t the cost of the subtree rooted at t, denoted T_t, bounds the cost of solutions built up from T_t to the hierarchical clustering problem for the subgraph of G induced by the set of vertices V_t. We prove the claim by induction on the level of the tree nodes. The claim clearly holds for all leaves of T. Consider an internal tree node $t \in T$ and denote its two children by t_L and t_R. By induction the claim holds for both t_L and t_R. The solution represented by T_t is given by concatenating the solutions represented by T_{t_L} and T_{t_R}. Note that the additional cost is at most $|V_t|$ times the capacity of the cut F_t that separates V_{t_L} from V_{t_R}. We get

$$cost_G(T_t) \leq cost_G(T_{t_L}) + cost_G(T_{t_R}) + |V_t| w(F_t).$$

The inductive claim follows.

We now show how to compute the spreading metric that assigns length $l(e)$ to an edge $e \in E$ of the graph. Consider the following linear program (LP1):

$$\min \sum_{e \in E} w(e) \cdot l(e) \quad \text{such that:} \tag{4.13}$$

$$\forall U \subseteq V, \forall v \in V : \sum_{u \in U} dist_l(u,v) \geq \frac{1}{2}(|U|^2 - 1) \tag{4.14}$$

$$\forall e \in E : l(e) \geq 0 \tag{4.15}$$

In the linear program, we follow the notation that regards $l(e)$ as edge lengths, and $dist_l(u,v)$ is the length of the shortest path from u to v. We will refer to constraint (4.14) as the spreading constraint. The linear program can be solved in polynomial time since we can construct a separation oracle. In order to verify that the spreading constraint (4.14) is satisfied, for each vertex v, we sort the vertices in V in increasing order of distance $dist_l(u,v)$ and verify the spreading constraint for all prefixes U of this sorted order.

Lemma 4.1.16 *Let $l(e)$ denote a feasible solution of the linear program. For every $U \subseteq V$ with $(|U| > 1)$, and for every vertex $v \in U$ there is a vertex $u \in U$ for which $dist_l(u,v) \geq \frac{1}{2}(|U| - 1)$.*

Proof. The average distance of a node $u \in U - \{v\}$ from v is greater than $\frac{1}{2}(|U|-1)$, because of the constraint corresponding to U and v. Therefore, there exists a vertex $u \in U$ whose distance from v is at least the average distance from v, and the lemma follows, since $dist_l(u,v) \geq \frac{1}{2}(|U|-1)$. ▮

Note that the previous lemma comes short of the diameter guarantee by a factor of 2: while the diameter guarantee requires that the diameter of a subset U be greater than $|U|$, the proven bound is only $|U|/2$. However, this only affects the constant in the approximation factor. In the next lemma, we prove that the volume of an optimal solution of the linear program satisfies the lower bound property.

Lemma 4.1.17 *The cost of an optimal solution of the linear program is a lower bound on the cost of an optimal hierarchical clustering of G.*

Proof. Consider any binary hierarchical clustering given by the sequence of cuts in the decomposition tree T and define $l(e) = |\textbf{leaves}(T[i \vee j])|$ for edge $e = (i,j) \in E$. It is easy to see that this is indeed a metric and it is actually an ultrametric. We show that $l(e)$ is a feasible solution for the linear program above. The cost $\sum_{e \in E} w(e) \cdot l(e)$ equals the cost of the hierarchical clustering induced by the tree T. The feasibility of $l(e)$ is proved as follows: Consider a subset $U \subseteq V$ and a vertex $v \in U$. We observe that the average distance from v of the vertices in U will be minimized when U is "packed around" v, meaning that with each cut we peel off only one vertex at a time. We have that:

$$\sum_{u \in U} dist_l(u,v) = 2 + 3 + ... + |U| \geq \frac{1}{2}(|U|^2 - 2)$$

Hence, $l(\cdot)$ is a feasible solution and the lemma follows. ▮

With the above two lemmas we have proved that our hierarchical clustering objective function falls into the spreading metrics paradigm, because it satisfies the lower bound property and the diameter property. Using Bartal's decomposition and specifically Theorem 4.1.13 from Section 4.1.6 we get an approximation guarantee of $O(\log n)$:

Theorem 4.1.18 *There exists an $O(\log n)$ approximation for the hierarchical clustering objective function defined by (1.1).*

4.1.7 Hardness via Min Linear Arrangement and Small Set Expansion

SSE and hardness amplification

Given a graph $G(V,E)$, define the following quantities for non-empty subsets $S \subset V$: normalized set size $\mu(S) \triangleq |S|/|V|$, and edge expansion $\Phi_G(S) \triangleq \frac{|E(S, V \setminus S)|}{\sum_{i \in S} d_i}$ (here d_i is the degree of i). The *Small Set Expansion* hypothesis was introduced by Raghavendra and Steurer [82].

Problem 4.1.7.1 (Small-Set Expansion(η,δ)) *Given a regular graph $G(V,E)$, distinguish between the following two cases:*
Yes: *There exists a non-expanding set $S \subseteq V$ with $\mu(S) = \delta$ and $\Phi_G(S) \leq \eta$.*
No: *All sets $S \subseteq V$ with $\mu(S) = \delta$ are highly expanding with $\Phi_G(S) \geq 1 - \eta$.*

Hypothesis 4.1.7.1 (Hardness of approximating Small-Set Expansion**)** *For all $\eta > 0$, there exists $\delta > 0$ such that the promise problem* Small-Set Expansion(η,δ) *is NP-hard.*

The authors of [82] showed that the Small Set Expansion Hypothesis implies the Unique Games Conjecture of Khot [65]. A decision problem is said to be SSE-hard if Small-Set Expansion(η,δ) reduces to it by a polynomial time reduction for some constant η and all $\delta > 0$. Raghavendra, Steurer and Tulsiani [83] showed the following hardness amplification result for graph expansion (see Preliminaries for Gaussian Graphs definitions):

Theorem 4.1.19 *For all $q \in \mathbb{N}$ and $\epsilon, \gamma > 0$, it is SSE-hard to distinguish between the following two cases for a given graph $H = (V_H, E_H)$:*
Yes: *There exist q disjoint sets $S_1, ..., S_q \subseteq V_H$ satisfying for all $l \in [q]$: $\mu(S_l) = 1/q$ and $\Phi_H(S_l) \leq$*

$\epsilon + o(\epsilon)$.
No: *For all sets $S \subseteq V_H$: $\Phi_H(S) \geq \Phi_{\mathcal{G}(1-\epsilon/2)}(\mu(S)) - \gamma/\mu(S)$, where $\Phi_{\mathcal{G}(1-\epsilon/2)}(\mu(S))$ is the expansion of sets of volume $\mu(S)$ in the infinite Gaussian graph $\mathcal{G}(1-\epsilon/2)$.*

Hierarchical Clustering Inapproximability based on SSE

Now we are ready to prove our SSE hardness result. Our proof follows the argument of [83] for establishing the hardness of Minimum Linear Arrangement. We prove the following:

Theorem 4.1.20 ***(Hardness of Hierarchical Clustering).*** *For every $\epsilon > 0$, it is SSE-hard to distinguish between the following two cases for a given graph $G = (V, E)$, with $|V| = n$:*
Yes: *There exists a decomposition tree T of the graph such that $cost_G(T) \leq \epsilon n|E|$*
No: *For any decomposition tree T of the graph $cost_G(T) \geq c\sqrt{\epsilon}n|E|$.*

Proof. We apply Theorem 4.1.19 for the following values: $q = \lceil 2/\epsilon \rceil, \epsilon' = \epsilon/3$ and $\gamma = \epsilon$. We need to first handle the **Yes** case. We get that the vertices can be divided into sets $S_1, S_2, ..., S_q$, each having size $n/q = n\epsilon/2$, such that at most $\epsilon' + o(\epsilon')$ fraction of edges leave the sets (i.e. go across sets). Now consider the hierarchical clustering solution that first partitions the vertices into the sets $S_1, S_2, ..., S_q$ and then partitions each S_i arbitrarily. Edges inside the set S_i contribute at most $|S_i|$ to the objective function and this is $|S_i| = n/q = \epsilon n/2$. Moreover, edges whose endpoints are in different sets will have contribution at most n; but there are at most $\epsilon/2$ fraction of such edges and so the overall objective for this hierarchical clustering solution is at most $\epsilon n|E|$.

Now, we handle the **No** case by using the argument of [83] for Minimum Linear Arrangement that follows from an observation of [44] and the fact that the objective function of Minimum Linear Arrangement is always less than the cost of Hierarchical Clustering. To see the latter, observe that when we have a hierarchical clustering tree T, if we consider the ordering of the vertices (leaves) as in the DFS order, then the stretch of an edge (u, v) that is cut, can be at most the size of the subtree which corresponds to that edge and this is exactly the quantity: $|\textbf{leaves}(T[u \vee v])|$. Since we know ([83, 44]) that in the **No** case, for all orderings $\pi : V \to [n], \mathbb{E}_{(u,v)\sim E}[|\pi(u) - \pi(v)|] \geq c\sqrt{\epsilon}n$, it immediately follows that: $cost_G(T) \geq c\sqrt{\epsilon}n|E|$. ∎

4.2 Maximization HC: Variations on a Theme

In this part of the thesis, we focus on the two maximization objectives, introduced recently to give insight into the performance of average-linkage clustering: the similarity based HC objective proposed by [78] and the dissimilarity based HC objective proposed by [39]. In both cases, we present tight counterexamples showing that average-linkage cannot obtain better than $\frac{1}{3}$ and $\frac{2}{3}$ approximations respectively (in the worst-case), settling an open question raised in [78]. This matches the approximation ratio of a random solution, raising a natural question: *can we beat average-linkage for* *these objectives?* We answer this in the affirmative, giving two new algorithms based on semidefinite programming with provably better guarantees.

In particular, two recent (and independent) works took this objective function viewpoint to understand the performance of Average-Linkage. Recall the two maximization objectives, the first with similarity weights

$$T^* = \operatorname*{argmax}_{\text{all trees } T} \sum_{(i,j)\in E} w_{ij} \cdot (n - |T_{ij}|) \tag{HC-OBJ-2}$$

and with dissimilarities:

$$T^* = \operatorname*{argmax}_{\text{all trees } T} \sum_{(i,j)\in E} w_{ij} \cdot |T_{ij}| \tag{HC-OBJ-3}$$

For maximizing the similarity-based objective, the first work showed that Average-Linkage obtains a $\frac{1}{3}$-approximation. Interestingly, for maximizing the dissimilarity-based objective, the second work also showed that Average-Linkage gives a $\frac{2}{3}$-approximation [39]. These two works helped in understanding the performance of Average-Linkage. Here we build along this direction.

Our Contributions. In this paper, we shed further light on these two hierarchical clustering objectives. Since both of the objectives were recently introduced in the context of explaining the success of Average-Linkage, and as these objectives are NP-hard to optimize [43], it is natural to ask how well these objectives can be approximated. Understanding the approximation factors achievable by other algorithms for these objectives is important in evaluating the explanation for the performance of Average-Linkage by these works.

It turns out that a random solution to both these optimization problems achieves an approximation ratio that matches the approximation guarantees established by the above works for Average-Linkage [78, 39]. In particular, [78] suggest that the performance of Average-Linkage may be strictly better than that of a random solution. Our first set of results is negative:

> *In the worst case, the approximation ratio achieved by Average-Linkage is no better than $\frac{1}{3}$ for the maximization objective of Moseley and Wang [78] and no better than $\frac{2}{3}$ for the maximization objective of Cohen-Addad et al. [39].*

This raises a natural question: is it possible to achieve an approximation factor strictly better than that achieved by Average-Linkage (also by a random solution) for these two objectives? Or is it the case that these objectives are approximation resistant (i.e., beating the performance of a random solution is provably hard)? Our main result here is positive:

> *We show simple algorithms that achieve a $\frac{1}{3}+\epsilon$ approximation for the maximization objective of [78] and an algorithm that achieves a $\frac{2}{3}+\delta$ approximation for the maximization*

objective of [39], for constants $\epsilon, \delta > 0$.

Our algorithms are conceptually very simple; in the former case, our algorithm is guided by a semidefinite programming (SDP) solution that has a vector representation for the hierarchical clustering for every level of granularity, and uses spreading metric constraints to strengthen the solution. We use the solution at level $n/2$ to make a judicious choice of initial partition, followed by a random solution to refine each piece produced (see section 4.2.3 for details). In the latter case, our algorithm follows a simple greedy peeling strategy, followed by a max-cut partition (see section 4.2.4 for details).

In addition to shedding light on the two objectives from the point of view of understanding their approximability, an additional lens with which to view our results is a philosophical one: our results raise the question about whether these objectives are indeed the right way to measure the performance of Average-Linkage clustering.

Notations. We abuse notation and use `OPT` to refer to both the optimum solution and its value for the HC problem at hand. Similarly, $Average-Linkage$ denotes both the solution produced by the Average-Linkage algorithm and its objective value. We emphasize that when dealing with similarity weights, the Average-Linkage merging criterion is to *maximize* average similarity across the subclusters available to the algorithm for merging, whereas when dealing with dissimilarity weights, the Average-Linkage merging criterion is to *minimize* average dissimilarity. We term `HC-OBJ-2` as *similarity-HC* and `HC-OBJ-3` as *dissimilarity-HC* in this paper. We use $\mathcal{W}$ to denote the total weight of the edges in the graph, i.e. $\mathcal{W} = \sum_{e \in E} w_e$.

We start by describing the constructions of two families of examples proving that the known performance bounds for Average-Linkage for the two objectives, i.e. *similarity-HC* and *dissimilarity-HC*, are tight.

4.2.1 Average-Linkage for similarity-HC is a tight $\frac{1}{3}$-approximation

Fact 4.2.1 ([43]) *If the graph is a clique with uniform weights for all of the edges, any clustering tree T obtains exactly the same cost/reward. We occasionally use this fact in this section.*

Average-Linkage for similarity-HC is a tight $\frac{1}{3}$-approximation We will provide a construction where the optimum solution `OPT` has value $\approx n\mathcal{W}$, but where Average-Linkage only gets $\frac{1}{3}n\mathcal{W}$ (ignoring lower order terms). The construction does two things: 1) Most of the graph's weight is inside subclusters containing $n^{2/3}$ nodes each. So there exists a solution merging almost all the weight in low levels of the hierarchical decomposition, getting $\approx n\mathcal{W}$ total value. 2) Average-Linkage cuts most of the graph's weight in higher levels of the corresponding tree decomposition so that according to `HC-OBJ-2`, the multiplier of the edge weights is small.

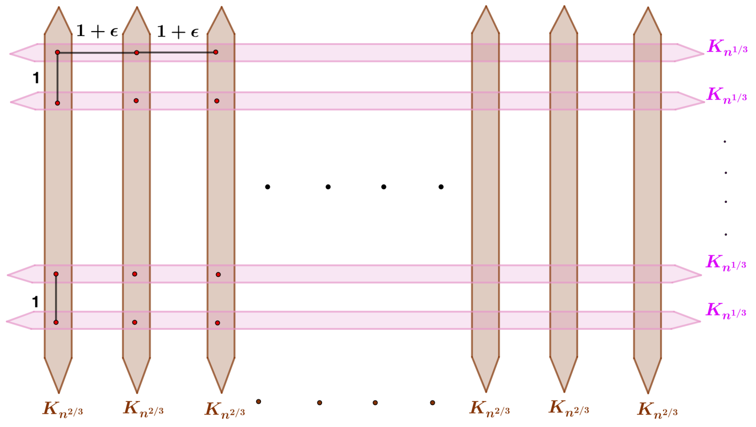

Figure 4.4: The tight instance where Average-Linkage is a tight $\frac{1}{3}$-approximation. The graph has n nodes organized in $n^{1/3}$ vertical groups of $n^{2/3}$ vertices. Each vertical group is a clique $K_{n^{2/3}}$ on $n^{2/3}$ nodes and there are $n^{1/3}$ such groups. Each horizontal group is a clique on $n^{1/3}$ nodes. Every edge in the vertical groups has weight 1, whereas every edge in the horizontal groups has weight $1+\epsilon$.

Construction of the tight instance. To achieve the above, our example (see Figure 4.4) will have n nodes and will contain multiple copies of cliques each of which is either a copy of $K_{n^{1/3}}$ or $K_{n^{2/3}}$. In particular, the tight instance consists of $n^{1/3}$ copies of $K_{n^{2/3}}$ with unit weight. With a slight abuse of notation, we fix an arbitrary ordering $1, 2, \ldots, n^{2/3}$ for the nodes in *each* $K_{n^{2/3}}$ and refer to them by their corresponding order in each clique. Now we augment this construction by adding all pairwise edges connecting nodes of the *same* order across all the $K_{n^{2/3}}$ cliques. This creates $n^{2/3}$ additional $K_{n^{1/3}}$ cliques and we fix the weight of these additional edges to be $1+\epsilon$ (for any small constant $\epsilon > 0$). Note that the total number of nodes is $n^{2/3} \cdot n^{1/3} = n$ and that the total weight of the graph is $\mathcal{W} = \frac{1}{2}n^{2/3} \cdot (n^{2/3}-1) \cdot n^{1/3} \cdot 1 + \frac{1}{2}n^{1/3} \cdot (n^{1/3}-1) \cdot n^{2/3} \cdot (1+\epsilon) = \frac{1}{2}n^{5/3} + O(n^{4/3})$.

The following two lemmas compare `OPT` to Average-Linkage (proofs are deferred to Appendix A.3).

Lemma 4.2.1 *In the above instance, the optimum obtains an objective of at least* $\frac{1}{2}n^{8/3} - O(n^{7/3}) \approx n\mathcal{W}$.

Lemma 4.2.2 *In the above instance, Average-Linkage gets at most* $\frac{1}{6}n^{8/3} + O(n^{7/3}) \approx \frac{1}{3}n\mathcal{W}$.

Combining these lemmas, we settle a open question raised in [78]:

Proposition 4.2.2 *There exists an instance for which Average-Linkage is a* $\frac{1}{3}+o(1)$*-approximation for the similarity-HC objective (`HC-OBJ-2`) introduced in [78].*

4.2.2 Average-Linkage for dissimilarity-HC is a tight $\frac{2}{3}$-approximation

When the pairwise scores denote dissimilarities, we focus on `HC-OBJ-3`. [39] showed that running Average-Linkage gives a $\frac{1}{2}$-approximation, and a slight modification of their proof (see journal version of [39]) gives an improved $\frac{2}{3}$-approximation bound. Here we show that the $\frac{2}{3}$ ratio is actually tight.

Construction of the tight instance. Let the number of nodes n be even. We start with the complete bipartite graph $K_{n/2,n/2}$ with unit weights (let L, R denote the two sides of the graph). We then remove any perfect matching M crossing the (L, R) cut (see Figure 4.5). Note that the total weight of the edges is $\mathcal{W} = \frac{n}{2} \cdot \frac{n}{2} - \frac{n}{2} \approx \frac{1}{4}n^2$.

The following two lemmas compare `OPT` to Average-Linkage (proofs are deferred to Appendix A.3).

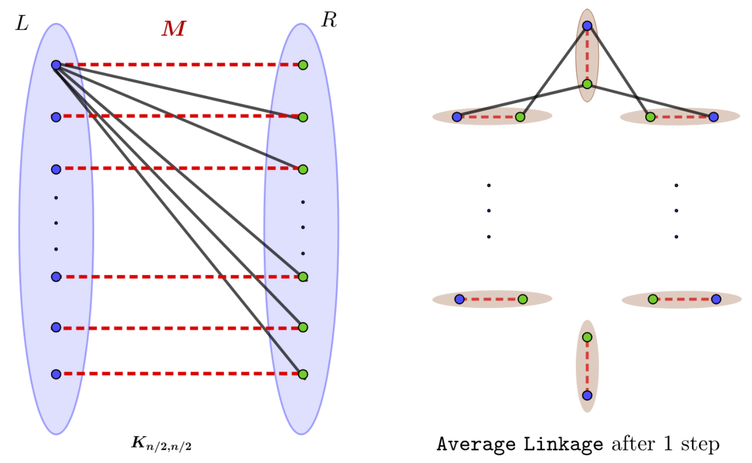

Figure 4.5: The tight instance where $Average - Linkage$ is a tight $\frac{2}{3}$-approximation. The graph is a complete bipartite graph where we removed a perfect matching M denoted with the red dashed edges. After one step of $Average - Linkage$, the instance is a clique on $\frac{n}{2}$ supernodes of size 2 with doubled edge weights.

Lemma 4.2.3 *In the above instance, the optimum HC decomposition obtains an objective value of at least* $\frac{1}{4}n^3 - O(n^2) \approx n\mathcal{W}$.

Lemma 4.2.4 *In the above instance, Average-Linkage gets at most* $\frac{1}{6}n^3 \approx \frac{2}{3}n\mathcal{W}$.

Combining these lemmas, we get the following result:

Proposition 4.2.3 *There exists an instance for which Average-Linkage is a* $\frac{2}{3}+o(1)$*-approximation for the dissimilarity-HC objective (`HC-OBJ-3`), introduced by [39].*

4.2.3 Beating Average Linkage via SDP for the Moseley-Wang HC Objective

In this section, we aim to design approximation algorithms for the similarity-based hierarchical clustering, i.e., creating a hierarchical decomposition that approximately maximizes the objective function HC-OBJ-2. To this end, we formulate a semidefinite programming (SDP) relaxation for the problem. We show that using a well chosen set of vectors returned by this SDP and a simple hyperplane rounding scheme, we can beat the average-linkage which is a tight $\frac{1}{3}$-approximation algorithm. The main challenge in the analysis of the rounding scheme is in lower bounding the probability of certain events related to the triplets of vertices and the order in which they get separated; we do this by exploiting the specific geometry of the vectors in the SDP optimal solution, and with the help of our imposed spreading metric constraints in this SDP relaxation.

SDP relaxation for similarity-HC

Suppose n datapoints with similarity weights $\{w_{ij}\}$ are given. An alternative view of a hierarchical clustering of these points is a collection of partitions of the points at different levels $t = n-1, \dots, 1$, where the partition at level t consists of all the maximal clusters of size at most t. Given this view, we can rewrite the similarity-HC objective function (HC-OBJ-2) as following.

$$\sum_{(i,j)\in E} w_{ij}(n - |T_{ij}|) = \sum_{t=1}^{n-1} \sum_{(i,j)\in E} w_{ij} \cdot \mathbf{1}\{i \text{ and } j \text{ are not separated at level } t\text{'s partitioning}\}$$

Now, given the optimal hierarchical clustering, consider a vector assignment where at every level $t = 1, .., n-1$ the same unit vectors are assigned to all the nodes in the same maximal cluster, while the assigned vectors to different clusters are chosen to be orthogonal. Let $\{\mathbf{v}_i^{(t)}\}$ be the set of assigned vectors. Clearly, the contribution of an edge w_{ij} at level t can be alternatively written as $w_{ij}(\mathbf{v}_i^{(t)} \cdot \mathbf{v}_j^{(t)})$. This observation suggests a relaxation through semidefinite programming/vector programming.

Proposition 4.2.4 *The following SDP is a relaxation for the similarity-HC problem (6.4).*

$$\begin{aligned}
\textit{maximize} \quad & \sum_{t=1}^{n-1} \sum_{(i,j)\in E} w_{ij}(1 - x_{ij}^{(t)}) & & \\
\textit{subject to} \quad & x_{ij}^{(t)} = \frac{1}{2}\|\mathbf{v}_i^{(t)} - \mathbf{v}_j^{(t)}\|_2^2, & \forall (i,j)\in E, t\in[1:n-1] & \\
& \sum_{j\in V: j\neq i} x_{ij}^{(t)} \geq n - t, & \forall i \in V, t\in[1:n-1] & \quad (\textit{spreading constraints}) \\
& x_{ij}^{(t+1)} \leq x_{ij}^{(t)}, \quad x_{ij}^{(1)} = 1 & \forall (i,j)\in E, t\in[1:n-1] & \quad (\textit{monotonicity constraints}) \\
& \mathbf{v}_i^{(t)} \in \mathbb{R}^n, \quad \|\mathbf{v}_i^{(t)}\|_2^2 = 1, & \forall i \in V, t\in[1:n-1] &
\end{aligned} \tag{HC-SDP}$$

In an integral HC decomposition, each node in a maximal cluster of size at most t has been separated from at least $n-t$ vertices at level t (i.e. all of the nodes outside of this cluster). We therefore add the constraints $\sum_j x_{ij}^{(t)} \geq n-t$, termed as the *spreading constraints*. Intuitively, they force the SDP to choose vectors that are somewhat separated, thus preventing it from cheating by assigning identical vectors. Finally, *monotonocity constraints* ensure monotonicty of the separation from top to bottom.

Combining SDP rounding and random to beat average-linkage

Suppose `OPT-SDP` and `OPT` denote the optimum solution of the SDP relaxation (HC-SDP) and the optimum integral solution of the similarity-HC objective (`HC-OBJ-2`) respectively. Our goal is to beat the $\frac{1}{3}$ approximation ratio attained by AVERAGE-LINKAGE . We will consider two simple algorithms for the HC problem. The first algorithm, "*random always*", cuts each cluster recursively and uniformly at random until it reaches to singletons. The second algorithm, "*SDP first, random next*", uses the semidefinite program HC-SDP and hyperplane rounding for determining the first cut, and then it picks a random cut for each of the later clusters until it reaches to singletons.

As it is also known ([78]), recursively performing random cuts will yield an HC solution with expected value exactly equal to $\frac{1}{3}(n-2)\mathcal{W}$, where $\mathcal{W} \triangleq \sum_{(i,j)\in E} w_{ij}$. An initial idea is that when there is a gap between `OPT` and the quantity $(n-2)\mathcal{W}$, i.e. when `OPT` $< (1-\epsilon_1)(n-2)\mathcal{W}$ for some small constant $\epsilon_1 > 0$, then "*random always*" already attains an approximation guarantee of $\frac{1}{3(1-\epsilon_1)} > \frac{1}{3}$. The name of the game is then to come up with a good approximation algorithm in the case where `OPT` is actually pretty large, close to $(n-2)\mathcal{W}$. Interestingly, a suitable initial cut can be found by exploiting the SDP relaxation in this case, which can be then used to guide the "*random always*" algorithm.

Theorem 4.2.5 *The best of the* "SDP first, random next" *(Algorithm 5) and* "random always" *(Algorithm 4) is a randomized α-approximation algorithm for maximizing the similarity-HC objective for hierarchical clustering, where $\alpha = 0.336379 > \frac{1}{3}$.*

Algorithm 4 Random Always

```
1: input: $G = (V, E)$.
2: if $|V| = 1$ then
3:     Return the singleton vertex as the only cluster.
4: else
5:     Randomly partition the set of vertices into $S$ and $\bar{S}$.
6:     Recursively run "random always" on $G_S$ and $G_{\bar{S}}$ to get clusters $\mathcal{C}_S$ and $\mathcal{C}_{\bar{S}}$.
7:     Return the clusters $S$,$\bar{S}$, $\mathcal{C}_S$ and $\mathcal{C}_{\bar{S}}$.
8: end if
```

Algorithm 5 SDP First, Random Next

```
1: input: $G = (V, E)$ and (similarity) weights $\{w_{ij}\}_{(i,j)\in E}$.
2: Solve the SDP relaxation HC-SDP to get an optimum assignment $\{x_{ij}^t\}_{(i,j)\in E,\ t=1,\ldots,n-1}$.
3: Let $x_{ij}^* = x_{ij}^{(\lfloor n/2\rfloor-1)}$ and $\mathbf{v}_i^* = \mathbf{v}_i^{(\lfloor n/2\rfloor-1)}$ be the optimal solution restricted to level $t = \lfloor n/2\rfloor - 1$.
4: Draw $\mathbf{v}_0$ uniformly at random from unit sphere, and let $S = \{i \in V : \mathbf{v}_i^* \cdot \mathbf{v}_0 \geq 0\}$.
5: Partition the vertices into $S$ and $\bar{S} = V \setminus S$.
6: Run "random always" (Algorithm 4) on $S$ and $\bar{S}$ to get clusters $\mathcal{C}_S$ and $\mathcal{C}_{\bar{S}}$.
7: Return the clusters $S$,$\bar{S}$, $\mathcal{C}_S$ and $\mathcal{C}_{\bar{S}}$.
```

Analysis (Proof of Theorem 4.2.5)

We start by decomposing the similarity-HC objective as a summation over contributions of different triplets of vertices i, j and k, where $(i, j) \in E$ and $k \neq i, j$. Accordingly, `HC-OBJ-2` can be rewritten as:

$$\sum_{(i,j)\in E} w_{ij}\,(n - |T_{ij}|) = \sum_{(i,j)\in E} w_{ij}|\texttt{non-leaves}(T_{ij})| = \sum_{(i,j)\in E}\sum_{k\neq i,j} w_{ij}\mathbf{1}\{k \text{ is not a leaf of } T_{ij}\} \tag{4.16}$$

The vertex k does *not* belong to the leaves of T_{ij} if and only if at some point during the execution of the algorithm, k gets separated from i and j, while i and j still remain in the same cluster. Suppose $T^{(1)}$ and $T^{(2)}$ denote the HC tree returned by Algorithm 4 and Algorithm 5. Moreover, let the random variables $\mathcal{Z}_{i,j,k}$ and $\mathcal{Y}_{i,j,k}$ denote the contributions of the edge (i, j) and vertex $k \neq i, j$ to the objective value of Algorithm 4 and Algorithm 5 respectively, i.e.,

$$\mathcal{Z}_{i,j,k} \triangleq w_{ij}\mathbf{1}\{k \text{ is a non-leaf of } T_{ij}^{(1)}\} \quad \text{and} \quad \mathcal{Y}_{i,j,k} \triangleq w_{ij}\mathbf{1}\{k \text{ is a non-leaf of } T_{ij}^{(2)}\}$$

Moreover, let $\mathcal{Y}_{i,j} = \sum_{k\neq i,j}\mathcal{Y}_{i,j,k}$ and $\mathcal{Z}_{i,j} = \sum_{k\neq i,j}\mathcal{Z}_{i,j,k}$. Let `OPT` be the optimal value of the HC objective. Fix $\epsilon_1 > 0$ and consider two cases.

Remark 7 *Already, from the formulation of the Moseley-Wang objective, as triplets we see that the old biology problem of rooted triplets consistency is a generalization of Moseley-Wang optimization. This actually gives a PTAS for dense instances of HC [41].*

Case 1: $\texttt{OPT} < (1-\epsilon_1)(n-2)\sum_{(i,j)\in E} w_{ij}$. By a simple argument, we claim that $\mathbf{E}\left[\mathcal{Z}_{i,j,k}\right] = \frac{1}{3}w_{ij}$. Given this claim, the expected objective value of Algorithm 4 is at least $\frac{n-2}{3}\sum_{(i,j)\in E} w_{ij}$. Moreover, $\texttt{OPT} \le (n-2)\sum_{(i,j)\in E} w_{ij}$, and hence Algorithm 4 obtains $\frac{1}{3(1-\epsilon_1)}$ fraction of $\texttt{OPT}$ in this case. To see why the claim holds, think of each random cut as flipping an independent unbiased coin for each vertex, and then placing the vertex on either sides of the cut based on the outcome of its coin. Now, look at the sequence of the coin flips of i, j and k during the execution of Algorithm 4. We want to find the probability of the event that for the first time the three sequences are not matched, but i's sequence and j's sequence are still matched. Fixing i's sequence, the probability that all three are always matched is $\sum_{i=1}^{\infty}(1/4)^i = 1/3$. Due to the symmetry, the rest of the probability will be divided equally between our target event and the event that for the first time these three sequences are not matched, but still i's sequence and k's sequence are matched. So, k is not a leaf of $T_{ij}^{(1)} = 1/3$, which proves the claim.

Case 2: $\texttt{OPT} \ge (n-2)(1-\epsilon_1)\sum_{(i,j)\in E} w_{ij}$. In this case, we want to find a lower bound on the objective value of Algorithm 5. To this end, we show how to bound $\mathbf{E}\left[\mathcal{Y}_{i,j}\right]$ from below for a large enough fraction of edge weights. Consider the following events:

$$\begin{aligned}\mathcal{E}_{i,j} &\triangleq \{i \text{ and } j \text{ remain together after the first cut}\},\\ \mathcal{E}_{i,j,k} &\triangleq \{i,\ j \text{ and } k \text{ remain together after the first cut}\},\\ \mathcal{E}_{i,j|k} &\triangleq \{i,\ j \text{ remain together and } k \text{ gets separated after the first cut}\}\end{aligned}$$

We can rewrite $\mathbf{E}\left[\mathcal{Y}_{i,j,k}\right]$ as follows.

$$\begin{aligned}\mathbf{E}\left[\mathcal{Y}_{i,j,k}\right] &= \mathbf{E}\left[\mathcal{Y}_{i,j}\mathbf{1}\{\mathcal{E}_{i,j}\}\right] = \mathbf{E}\left[\mathcal{Y}_{i,j,k}\mathbf{1}\{\mathcal{E}_{i,j,k}\}\right] + \mathbf{E}\left[\mathcal{Y}_{i,j,k}\mathbf{1}\{\mathcal{E}_{i,j|k}\}\right]\\ &= \mathbf{E}\left[\mathcal{Y}_{i,j,k}|\mathcal{E}_{i,j,k}\right]\mathcal{E}_{i,j,k} + \mathbf{E}\left[\mathcal{Y}_{i,j,k}|\mathcal{E}_{i,j|k}\right]\mathcal{E}_{i,j|k}\end{aligned}$$

Now, note that $\mathbf{E}\left[\mathcal{Y}_{i,j,k}|\mathcal{E}_{i,j|k}\right] = w_{ij}$, as k has been separated from i and j after the first cut. Moreover, k is a non-leaf of $T_{ij}^{(2)}|\mathcal{E}_{i,j,k} = 1/3$, as Algorithm 5 performs random cuts after the first cut and the previous argument for analyzing $\mathcal{Z}_{i,j,k}$ will be applied. So, $\mathbf{E}\left[\mathcal{Y}_{i,j,k}|\mathcal{E}_{i,j,k}\right] = w_{ij}/3$. Therefore, we have:

$$\mathbf{E}\left[\mathcal{Y}_{i,j,k}\right] = \frac{w_{ij}}{3}\mathcal{E}_{i,j,k} + w_{ij}\mathcal{E}_{i,j|k} = \frac{w_{ij}}{3}\left(\mathcal{E}_{i,j,k} + \mathcal{E}_{i,j|k} + 2\mathcal{E}_{i,j|k}\right) = \frac{w_{ij}}{3}\left(\mathcal{E}_{i,j} + 2\mathcal{E}_{i,j|k}\right),$$

and hence we have:

$$\mathbf{E}\left[\mathcal{Y}_{i,j}\right] = \sum_{k\neq i,j}\mathbf{E}\left[\mathcal{Y}_{i,j,k}\right] = \frac{w_{ij}}{3}\left((n-2)\mathcal{E}_{i,j} + 2\sum_{k\neq i,j}\mathcal{E}_{i,j|k}\right) \tag{4.17}$$

Fix $\epsilon_2 > 0$. Consider all edges for which $x^*_{ij} = x^{(\lfloor n/2 \rfloor -1)}_{ij} \leq \epsilon_2$ (denoted by $\mathcal{H} \subseteq E$). For each $(i,j) \in \mathcal{H}$, by applying the basics of hyperplane rounding, e.g. in [55], we have:

$$\mathcal{E}_{i,j} = (\mathbf{v}^*_i \cdot \mathbf{v}_0)(\mathbf{v}^*_j \cdot \mathbf{v}_0) \geq 0 = 1 - \theta_{ij}/\pi,$$

where θ_{ij} is defined to be the angle between the vectors $\mathbf{v}^*_i$ and $\mathbf{v}^*_j$, i.e. $\theta_{i,j} = \cos^{-1}(\mathbf{v}^*_i \cdot \mathbf{v}^*_j) = \cos^{-1}(1 - x^*_{ij})$. Now, it is clear that $\theta_{ij} \leq \bar{\theta}$ for edges in $\mathcal{H}$, where $1 - \cos(\bar{\theta}) = \epsilon_2$. Therefore,

$$\mathcal{E}_{i,j} \geq 1 - \bar{\theta}/\pi, \quad \forall (i,j) \in \mathcal{H} \tag{4.18}$$

To bound $\sum_{k \neq i,j} \mathcal{E}_{i,j|k}$ for every edge $(i,j) \in \mathcal{H}$, we first find an explicit closed-form for each probability term. Interestingly, despite the complicated nature of this calculation, our method is simple and is not relaying on any three-dimensional geometry. Hence, it might be of independent interest.

Remark 8 *To calculate an explicit closed-form for the probability of the event $\mathcal{E}_{i,j|k}$, three involved correlated random variables $\mathbf{v}^*_i \cdot \mathbf{v}_0$, $\mathbf{v}^*_j \cdot \mathbf{v}_0$ and $\mathbf{v}^*_k \cdot \mathbf{v}_0$ need to be considered. In the direct approach, e.g. à la [55], we need to look at the unit projections of these three vectors and $\mathbf{v}_0$ onto the span of $\mathbf{v}^*_i$, $\mathbf{v}^*_j$, and $\mathbf{v}^*_k$ (hence a three-dimensional representation for each). Suppose $\tilde{\mathbf{v}}_0$ be the projection of $\mathbf{v}_0$ onto the mentioned three-dimensional space. As the entries of $\mathbf{v}_0$ are jointly Gaussian, $\tilde{\mathbf{v}}_0$ is indeed a uniformly random point from the three-dimensional sphere. Now, finding the probability of the event that i and j are on one side and k is on the other side of the hyperplane with normal vector $\tilde{\mathbf{v}}_0$ involves a complicated calculation in this three dimensional geometry.*

Lemma 4.2.5 *For every triplet of vertices i,j and k, $\mathcal{E}_{i,j|k} = \frac{\theta_{ik}+\theta_{jk}-\theta_{ij}}{2\pi}$*

Proof. We start by the following key observation, which relates the quantities $\mathcal{E}_{i,j|k}$, $\mathcal{E}_{i,k|j}$ and $\mathcal{E}_{k,j|i}$ to the original separation probabilities of a hyperplane rounding scheme:

1. $\mathcal{E}_{i,k|j} + \mathcal{E}_{j,k|i} = 1 - \mathcal{E}_{i,j} = \frac{\theta_{ij}}{\pi}$
2. $\mathcal{E}_{i,j|k} + \mathcal{E}_{i,k|j} = 1 - \mathcal{E}_{j,k} = \frac{\theta_{jk}}{\pi}$
3. $\mathcal{E}_{j,k|i} + \mathcal{E}_{i,j|k} = 1 - \mathcal{E}_{i,k} = \frac{\theta_{ik}}{\pi}$

Solving the above 3×3 system, we can obtain the desired closed-form expressions for $\mathcal{E}_{i,j|k}$, $\mathcal{E}_{i,k|j}$ and $\mathcal{E}_{k,j|i}$ in terms of the angles between the vectors:

$$\mathcal{E}_{i,j|k} = \frac{\theta_{ik}+\theta_{jk}-\theta_{ij}}{2\pi} \quad , \quad \mathcal{E}_{i,k|j} = \frac{\theta_{ij}+\theta_{jk}-\theta_{ik}}{2\pi} \quad , \quad \mathcal{E}_{k,j|i} = \frac{\theta_{ik}+\theta_{ij}-\theta_{jk}}{2\pi}$$

This concludes the proof. ∎

In the next step of the analysis, we find a worst-case lower-bound for $\sum_{k\neq i,j}\mathcal{E}_{i,j|k}$ through a *factor revealing program*. More accurately, we set up a minimization problem where the objective function is equal to $\sum_{k\neq i,j}\mathcal{E}_{i,j|k}$. For the constraints, note that due to the spreading constraints of the similarity-HC SDP relaxation (HC-SDP) for vertices i and j, we have

$$\sum_{k\neq i} x^*_{ik} \geq n - \lfloor n/2\rfloor + 1 \quad , \quad \sum_{k\neq j} x^*_{jk} \geq n - \lfloor n/2\rfloor + 1$$

and therefore $\sum_{k\neq i}\cos(\theta_{ik}) \leq n/2-1$ and $\sum_{k\neq j}\cos(\theta_{jk}) \leq n/2-1$. Now, by applying Theorem 4.2.5, we can lower bound $\sum_{k\neq i,j}\mathcal{E}_{i,j|k}$ by the optimal solution of the following optimization problem:

$$\begin{aligned}
\textit{minimize} \quad & \frac{1}{2\pi}\sum_{k\neq i,j}\left(\theta_{ik}+\theta_{jk}-\bar{\theta}\right) \\
\textit{subject to} \quad & \sum_{k\neq i}\cos(\theta_{ik}) \leq n/2-1, \\
& \sum_{k\neq j}\cos(\theta_{jk}) \leq n/2-1, \\
& 0\leq \theta_{ik}\leq \frac{\pi}{2} \quad , \quad 0\leq\theta_{jk}\leq\frac{\pi}{2} \qquad \forall k
\end{aligned} \tag{$\mathcal{P}_{\text{lower-bound}}$}$$

Note that we restrict our attention to $0\leq\theta_{ik},\theta_{jk}\leq\pi/2$, simply because in HC-SDP we force $x^{(t)}_{ij}\leq 1$, and hence $\mathbf{v}^*_i\cdot\mathbf{v}^*_j\geq 0$ for all $i,j\in V$. We now have this lemma, whose proof is deferred to Appendix A.4.

Lemma 4.2.6 *The optimal solution of the factor revealing program $\mathcal{P}_{\textit{lower-bound}}$ is lower-bounded by*

$$(n-2)\left(\tfrac{1}{4}-\tfrac{\bar{\theta}}{2\pi}\right)$$

By combining eq. (4.17) and eq. (4.18) with Theorem 4.2.6, we have:

$$\begin{aligned}
\mathbf{E}\left[\mathtt{OBJ}_{\mathrm{ALG}_2}\right] &\geq \sum_{(i,j)\in\mathcal{H}}\mathbf{E}\left[\mathcal{Y}_{i,j}\right] \geq (n-2)\left(\sum_{(i,j)\in\mathcal{H}} w_{ij}\right)\left(\frac{1}{3}\cdot\left(1-\frac{\bar{\theta}}{\pi}\right)+\frac{2}{3}\cdot\left(\frac{1}{4}-\frac{\bar{\theta}}{2\pi}\right)\right) \\
&\geq (n-2)\left(\sum_{(i,j)\in\mathcal{H}} w_{ij}\right)\left(\frac{1}{2}-\frac{2\bar{\theta}}{3\pi}\right)
\end{aligned} \tag{4.19}$$

We finally bound the total weight of edges in $\mathcal{H}$. Note that for an edge $(i,j)\notin\mathcal{H}$, $x^*_{ij}=x^{(\lfloor n/2\rfloor-1)}_{ij}>\epsilon_2$. Therefore, due to the monotonicity constraint in HC-SDP, $x^{(t)}_{ij}>\epsilon_2, \forall 1\leq t\leq\lfloor n/2\rfloor-1$. Now we have:

$$\texttt{OPT-SDP} = \sum_{t=1}^{n-1} \sum_{(i,j)\in E} w_{ij}(1 - x_{ij}^{(t)}) \le (n-2)\sum_{(i,j)\in E} w_{ij} - \sum_{t=1}^{\lfloor n/2 \rfloor - 1} \sum_{(i,j)\in E} w_{ij}x_{ij}^{(t)}$$
$$\le (n-2)\sum_{(i,j)\in E} w_{ij} - \frac{\epsilon_2(n-2)}{2} \cdot \sum_{(i,j)\in E\setminus\mathcal{H}} w_{ij}$$

On the other hand, based on the assumption of Case 2, we know

$$\texttt{OPT-SDP} \ge \texttt{OPT} \ge (n-2)(1-\epsilon_1)\sum_{(i,j)\in E} w_{ij} \tag{4.20}$$

By rearranging the terms we have $\sum_{(i,j)\in\mathcal{H}} w_{ij} \ge (1 - \frac{2\epsilon_1}{\epsilon_2})\sum_{(i,j)\in E} w_{ij}$. Now, combined with Equation (4.19), we can show Algorithm 5 obtains $(1-\frac{2\epsilon_1}{\epsilon_2})(\frac{1}{2} - \frac{2\cos^{-1}(1-\epsilon_2)}{3\pi})$ fraction of $\texttt{OPT}$.

By balancing out the two cases, finding the optimal ϵ_1 as a function of ϵ_2, and finally by setting $\epsilon_2 \approx 0.139$ we get the desired approximation factor of ≈ 0.336379. For more details, refer to Appendix A.4.

4.2.4 Beating Average Linkage via MaxCut for Dissimilarity HC

In this section we focus on the dissimilarity-HC objective (HC-OBJ-3). As demonstrated in Section 4.2.2, Average-Linkage fails to obtain better than $\frac{2}{3}$ fraction of the optimum in the worst-case. Similarly, "*random always*" fails to beat this approximation ratio, simply because its objective value on any instance is exactly equal to $\frac{2}{3}n\mathcal{W}$, while in a bipartite graph the optimum dissimilarity-HC objective is equal to $n\mathcal{W}$. Therefore, one natural question to ask is if there exists a polynomial time algorithm that can beat the $\frac{2}{3}$ approximation factor. We answer this question in the affirmative by providing a simple algorithm.

The "Peel-off First, Max-cut Next" Algorithm

By looking at the structure of the dissimilarity-HC objective function in eq. (HC-OBJ-3), it is clear that the top-level cuts of the tree, i.e. those corresponding to clusters of larger sizes, have considerable contributions to the objective function. For example, consider a simple algorithm that starts with a random cut and then forms the rest of the tree arbitrarily. This algorithm can still obtain an objective value of $\frac{1}{2}n\mathcal{W}$. Inspired by this observation, a tempting idea to beat the approximation factor of $\frac{2}{3}$ is to start with an approximation algorithm for the max-cut, e.g. [55], and then construct the rest of the hierarchical tree (probably by random cutting or by continuing with the same max-cut algorithm).

However, this naive approach fails because of the following instance. Suppose we have a graph with an embedded clique of size ϵn (for an arbitrarily small $\epsilon > 0$) and the rest of the weights are zero. The optimum dissimilarity-HC solution clearly peels off vertices of the clique one by one, and obtains an objective value of at least $n(1-\epsilon)\mathcal{W}$. However, the "recursive max-cut" or the "max-cut first, random next" both cut the clique into two (almost) symmetric halves at each iteration, and obtain an objective value of at most

$$\text{objective-value} \leq \left(n\frac{\mathcal{W}}{2} + \epsilon n\frac{\mathcal{W}}{4}\right) + \left(n\frac{\mathcal{W}}{8} + \epsilon n\frac{\mathcal{W}}{16}\right) + \left(n\frac{\mathcal{W}}{32} + \epsilon n\frac{\mathcal{W}}{64}\right) + \ldots \leq \frac{2+\epsilon}{3}n\mathcal{W}$$

The above example suggests a natural modification to our idea, i.e. to first peel off *high weighted degree* vertices, and then use a max-cut algorithm. Intuitively, if in such a pre-processed instance the optimum objective value of the dissimilarity-HC is close to $n\mathcal{W}$, then there should exist a considerably large cut. This large cut can be detected by an approximate max-cut algorithm, and will provide a large enough objective value for the dissimilarity-HC if used as a top-level cut in the final hierarchical clustering tree. If there is a constant gap between the optimum and $n\mathcal{W}$, one can run "*random always*" and already get an approximation factor strictly better than $\frac{2}{3}$. Formally, we propose *"peel-off first, max-cut next"* (Algorithm 6) and show how the better of this algorithm and the "*random always*" (Algorithm 4) beats the $\frac{2}{3}$-approximation factor by a small constant.

Algorithm 6 Peel-off First, Max-cut Next

```
1: **input:** $G = (V, E)$, (dissimilarity) weights $\{w_{ij}\}_{(i,j)\in E}$, and parameter $\gamma > 0$.
2: Initialize hierarchical clustering tree $T \leftarrow \emptyset$.
3: Set the *peeling-off threshold* to be $\tau = \frac{2\mathcal{W}}{n} \cdot \gamma$.
4: $\tilde{V} \leftarrow V$ and $\tilde{E} \leftarrow E$.
5: **while** $\exists$ a vertex $v \in \tilde{V}$ such that $\sum_{u \in V : (v,u) \in E} w_{vu} > \tau$ **do**
6:     Update the HC binary tree $T$ by adding the cut $(\{v\}, \tilde{V} \setminus \{v\})$ to the tree.
7:     $\tilde{V} \leftarrow \tilde{V} \setminus \{v\}$ and $\tilde{E} \leftarrow \tilde{E} \setminus \{e \in E : e$ incident to $v\}$.
8: **end while**
9: Run [55] for max-cut on $\tilde{G} = (\tilde{V}, \tilde{E})$.          {→ max-cutting phase.}
10: Let the resulting cut be $(S, \tilde{V} \setminus S)$, and update the HC tree $T$ by adding this cut to the tree.
11: Run "*random always*" (Algorithm 4) on $S$ and $\tilde{V} \setminus S$. Add the resulting binary trees to $T$.
12: Return the tree $T$.
```

Theorem 4.2.6 *There exists a choice of $\gamma > 0$ so that the best of* "peel-off first, max-cut next" *with parameter γ (Algorithm 6) and* "random always" *(Algorithm 4) is an α-approximation algorithm for maximizing the dissimilarity-HC objective, where $\alpha = 0.667078 > \frac{2}{3}$.*

Analysis (Proof of Theorem 4.2.6)

Fix a parameter ϵ. Let OPT be the optimal objective value of the dissimilarity-HC. Similar to the proof of Theorem 4.2.5, consider two cases:

Case 1: $\texttt{OPT} < (1-\epsilon)n\mathcal{W}$. A simple argument (refer to the proof of Theorem 4.2.5) shows that the expected objective value of Algorithm 4 is exactly equal to $\frac{2}{3}n\mathcal{W}$ in this case, and therefore it obtains $\frac{2}{3(1-\epsilon)}$ fraction of $\texttt{OPT}$ (for an exposition of this proof, we refer the reader to 35)

Case 2: $\texttt{OPT} \geq (1-\epsilon)n\mathcal{W}$. In this case, let the optimum (binary) HC tree be T^*. Fix another parameter $\delta < \frac{1}{2}$. The collection of all maximal clusters of size at most $n(1-\delta)$ forms a partition of the vertices. Here is a recursive way of looking at this partition: Imagine we start from the root of T^*. Each time the optimum tree performs a binary cut, we consider the two produced clusters (see Figure 4.6). If any of these clusters has size at most $n(1-\delta)$, then it is a maximal cluster of size at most $n(1-\delta)$ and, by definition, it will be added to the partition. As $\delta < \frac{1}{2}$, either both of these clusters must be of size at most $n(1-\delta)$, or exactly one of them is smaller than $n(1-\delta)$ while the other is strictly larger than $n(1-\delta)$. If the latter is true, we recursively follow the tree along the bigger cluster, i.e. we make the bigger cluster the new root and we iterate. Otherwise, if the former is true, we stop following the tree as we would have already produced two smaller than $n(1-\delta)$ pieces. We denote the produced sequence of clusters by $(L_1, R_1), \ldots, (L_k, R_k)$, where (L_i, R_i) are the two clusters produced by T^* at the i^{th} split. Without loss of generality we set:

$$|L_i| \geq n(1-\delta) > |R_i|, \; i = 1, \ldots, k-1 \qquad \text{and} \qquad \max\left(|L_k|, |R_k|\right) < n(1-\delta)$$

Based on the above construction, the resulting partition consists of the sets $R_1, R_2, \ldots, R_k$ and L_k. Note that $|L_k| + |R_k| = |L_{k-1}| \geq n(1-\delta)$, and therefore the rest of the graph contains $|V \setminus (L_k \cup R_k)| = n - |L_{k-1}| \leq \delta n$ vertices.

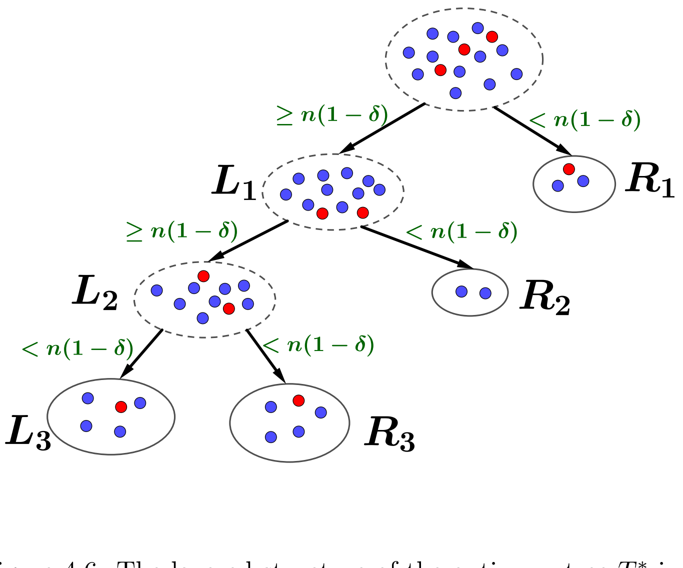

Figure 4.6: The layered structure of the optimum tree T^* in Case 2.

$\mathtt{OPT}$	value of HC for the optimum solution T^*
$\mathtt{OPT}_{\text{red}}$	contribution of edges with at least one red endpoint in optimum
$\mathtt{OPT}_{\text{blue}}$	contribution of blue edges in optimum
$\mathtt{OPT}_{\text{blue-chain}}$	contribution of blue edges $(u, \cdot), u \in V \setminus (L_k \cup R_k)$ in optimum
$\mathtt{OPT}_{\text{blue-cut}}$	contribution of blue edges $(u, v), u, v \in L_k \cup R_k$ in optimum
$\mathtt{ALG}_{\text{peel}}$	objective value gained by our algorithm during peeling-off phase 1
$\mathtt{ALG}_{\text{cut}}$	objective value gained by our algorithm during max-cutting phase 2
$MaxCut_{blue}$	max-cut value only among blue vertices available to our algorithm in phase 2

Table 4.1: A guide through the different variable names used in the proof.

Before delving into the proof details for Theorem 4.2.6, we first provide an overview of the proof highlighting the main ideas.

Proof Sketch: Our algorithm runs in two phases, namely the *peeling-off phase* and the *max-cutting phase* and a list of the symbols involved in the proof is provided in Table 4.1.

Step 1: Even though our algorithm removes vertices one by one during the peeling-off phase while the optimum tree T^* removes chunks of nodes (i.e. the R_i's), we will be flexible to ignore their contributions to $\mathtt{OPT}$ by only losing a small factor in the approximation, because these pieces are small.

Step 2: We want to devise a charging scheme between $\mathtt{OPT}$ and $\mathtt{ALG}$. To achieve this, we further divide $\mathtt{ALG} = \mathtt{ALG}_{\text{peel}} + \mathtt{ALG}_{\text{cut}}$ (these are the contributions to the HC objective during the peeling-off and max-cutting phase respectively) and $\mathtt{OPT} = \mathtt{OPT}_{\text{red}} + \mathtt{OPT}_{\text{blue}}$. Suppose we mark the vertices that the algorithm peels off during the first phase as “red” and the rest of the vertices are marked as “blue”.

Step 3: To take care of $\mathtt{OPT}_{\text{red}}$ (this is the total contribution of edges with at least one red endpoint in the objective value of T^*), we only use $\mathtt{ALG}_{\text{peel}}$. Note that there can't be many high weighted degree vertices, so every vertex removed by $\mathtt{ALG}_{\text{peel}}$ had a significant multiplier in the HC objective. Since, $\mathtt{OPT}_{\text{red}}$ could only have a multiplier of n we get that $\mathtt{ALG}_{\text{peel}} \geq (1 - \frac{2\mathcal{W}}{n\tau})\mathtt{OPT}_{\text{red}}$ (see Theorem 4.2.7). From this point on, we can completely ignore red vertices in the analysis.

Step 4: The remaining edges have blue both of their end-points (referred to as blue edges from now on). Let $\mathtt{OPT}_{\text{blue}}$ be the total contribution of blue edges in the optimum T^*. Dealing with $\mathtt{OPT}_{\text{blue}}$ requires more work. We need to further break $\mathtt{OPT}_{\text{blue}}$ into: $\mathtt{OPT}_{\text{blue}} = \mathtt{OPT}_{\text{blue-chain}} + \mathtt{OPT}_{\text{blue-cut}}$ (the total contribution of all blue edges with at least one end-point in $V \setminus (L_k \cup R_k)$ and the total contribution of all blue edges with both end-points in $L_k \cup R_k$ respectively). Note that $\mathtt{OPT}_{\text{blue-chain}}$ is negligible because it refers to low weighted degree vertices in small pieces, so $\mathtt{OPT}_{\text{blue-chain}}$ is a small fraction of $n\mathcal{W}$ (and hence of $\mathtt{OPT}$ which is close to $n\mathcal{W}$). See Theorem 4.2.8.

Step 5: Finally, we will use $\mathtt{ALG}_{\text{cut}}$ to take care of the $\mathtt{OPT}_{\text{blue-cut}}$ entirely. Actually, $\mathtt{ALG}_{\text{cut}}$ will take care not only for the $\texttt{OPT}_{\text{blue-cut}}$ (for now ignore some contribution from $w(L_k), w(R_k)$ because it is really small), but also at least half of the $\texttt{OPT}_{\text{blue-chain}}$. See Theorem 4.2.10.

The above steps lead us to Theorem 4.2.13 which finishes the proof of Theorem 4.2.6.

Lemma 4.2.7 *Let ℓ denote the number of vertices peeled off during the first phase. Then $\ell \leq \frac{2\mathcal{W}}{\tau}$ and also $\texttt{ALG}_{peel} \geq (1 - \frac{2\mathcal{W}}{n\tau})\texttt{OPT}_{red}$.*

Proof. Every vertex that is peeled off during the first phase has weighted degree at least τ. Observe that ℓ cannot be larger than $\frac{2\mathcal{W}}{\tau}$, simply because the total sum of the weighted degrees is at most $2\mathcal{W}$. Moreover, every peeled-off vertex u (these are exactly the red vertices) belongs to a cluster of size at least $(n - \ell) \geq (n - \frac{2\mathcal{W}}{\tau})$, hence u's contribution to $\texttt{ALG}_{\text{peel}}$ is at least $(n - \frac{2\mathcal{W}}{\tau}) \sum_{v \in V} w_{uv}$. Note that by the definition of $\texttt{OPT}_{\text{red}}$ we have:

$$\texttt{OPT}_{\text{red}} \leq n \cdot \sum_{u \text{ is red}} \sum_{v \in V} w_{uv}$$

Summing up the contributions of red vertices to $\texttt{ALG}_{\text{peel}}$:

$$\texttt{ALG}_{\text{peel}} \geq (n - \tfrac{2\mathcal{W}}{\tau}) \sum_{u \text{ is red}} \sum_{v \in V} w_{uv} \geq (1 - \tfrac{2\mathcal{W}}{n\tau})\texttt{OPT}_{\text{red}}$$

This concludes the second part of the claim. ∎

We just obtained an upper bound for $\texttt{OPT}_{\text{red}}$ in terms of our algorithm's $\texttt{ALG}_{\text{peel}}$ so we can ignore from now on the red vertices and turn our attention to $\texttt{OPT}_{\text{blue}} = \texttt{OPT}_{\text{blue-chain}} + \texttt{OPT}_{\text{blue-cut}}$. The first step is to upper bound $\texttt{OPT}_{\text{blue-chain}}$:

Lemma 4.2.8 *$\texttt{OPT}_{blue\text{-}chain} \leq \delta\tau n^2 \leq \frac{2\delta\gamma}{1-\epsilon}\texttt{OPT}$.*

Proof. As noted previously, $|V \setminus (L_k \cup R_k)| \leq \delta n$, hence there are not that many vertices in $|V \setminus (L_k \cup R_k)|$. Since any edge that contributes to $\texttt{OPT}_{\text{blue-chain}}$, must have, by definition, at least one endpoint in $|V \setminus (L_k \cup R_k)|$, there are at most δn such edges and because they are blue, again by definition, their weighted degree is smaller than τ. Noting that the maximum cluster size is at most n, we conclude that $\texttt{OPT}_{\text{blue-chain}} \leq (\delta n) \cdot \tau \cdot n = \delta\tau n^2$ and substituting τ in terms of γ, we get the lemma. ∎

Let $w(L_k, R_k)$ be the the total weight of blue edges crossing the cut (L_k, R_k) and let $w(R_k)$ and $w(L_k)$ be the total weight of the edges with both end-points in R_k and L_k respectively. An obvious upper bound that can be derived for $\texttt{OPT}_{\text{blue-cut}}$ (recall that this refers only to blue edges), by focusing on the graph induced by the blue vertices in L_k, R_k, is $\texttt{OPT}_{\text{blue-cut}} \leq n(w(L_k, R_k) + w(L_k) + w(R_k))$.

After the max-cutting phase is over, we have no further control over the contribution of edges with both end-points in L_k or both end-points in R_k, so we should better have an upper bound for their total weights. Informally, since $\texttt{OPT}$ is large (**Case 2**) and both L_k, R_k have small size, it can't be the case that significant portion of the weight lies inside L_k, R_k, as otherwise $\texttt{OPT}$ would have to be small (the formal proof is deferred to the Appendix A.5).

Claim 4.2.9 $nw(L_k) + nw(R_k) \leq \frac{\epsilon}{\delta} n\mathcal{W} \leq \frac{\epsilon}{(1-\epsilon)\delta} \mathit{OPT}$.

The next lemma starts by a lower bound for the $MaxCut_{blue}$ value, which is the value of the maximum cut in the graph induced only from the blue vertices available to our algorithm during its max-cutting phase, i.e. after we have removed the red vertices. Our algorithm will of course get only a ρ_{GW}-approximation ($\rho_{\text{GW}} \approx 0.878$) to $MaxCut_{blue}$, since it uses Goemans-Williamson for max-cut.

Lemma 4.2.10 $\mathit{ALG}_{cut} \geq \rho_{GW}\left(1 - \frac{2\mathcal{W}}{n\tau}\right)\left(\mathit{OPT}_{blue\text{-}cut} - \frac{\epsilon}{(1-\epsilon)\delta}\mathit{OPT} + \frac{\mathit{OPT}_{blue\text{-}chain}}{2}\right)$

Proof. As mentioned, we know that $nw(L_k, R_k) \geq \texttt{OPT}_{\text{blue-cut}} - nw(L_k) - nw(R_k) \geq \texttt{OPT}_{\text{blue-cut}} - \frac{\epsilon}{(1-\epsilon)\delta}\texttt{OPT}$. During the max-cutting phase, the vertices available to our algorithm are all the blue vertices. These can be divided into two categories relative to the $\texttt{OPT}$ solution. The first category are blue vertices $u \in L_k \cup R_k$. The second category are blue vertices $v \in V \setminus (L_k \cup R_k)$. Imagine the following cut $(C, \bar{C})$ with weight $w(C, \bar{C})$: focus only on the vertices u of the first category and split them into two pieces optimally to obtain the maximum cut. Now randomly assign the vertices v of the second category to the two pieces. This cut $(C, \bar{C})$ would obtain, by definition, an HC objective value of:

$$n \cdot w(C, \bar{C}) \geq n \cdot w(L_k, R_k) + \frac{\texttt{OPT}_{\text{blue-chain}}}{2} \geq \texttt{OPT}_{\text{blue-cut}} - \frac{\epsilon}{(1-\epsilon)\delta}\texttt{OPT} + \frac{\texttt{OPT}_{\text{blue-chain}}}{2}$$

Since $MaxCut_{blue}$ is the optimal cut, it can only be better than $w(C, \bar{C})$ and hence:

$$n \cdot MaxCut_{blue} \geq \texttt{OPT}_{\text{blue-cut}} - \frac{\epsilon}{(1-\epsilon)\delta}\texttt{OPT} + \frac{\texttt{OPT}_{\text{blue-chain}}}{2}$$

We know from Theorem 4.2.7, that at the beginning of the max-cutting phase, our algorithm has removed at most $\frac{2\mathcal{W}}{\tau}$ vertices and hence, the cluster size at the point where our algorithm uses the Goemans-Williamson algorithm is at least $(n - \frac{2\mathcal{W}}{\tau})$. The lemma follows since we can only get a ρ_{GW} approximation to $MaxCut_{blue}$. ∎

Finally, we are able to combine all the above together into the final comparison between our algorithm's objective value $\texttt{ALG}$ and the optimum $\texttt{OPT}$:

Lemma 4.2.11 *Let* $\tau = \gamma\frac{2\mathcal{W}}{n}, \delta = \frac{\sqrt{\epsilon}}{\sqrt{\gamma}}$. *By optimizing for the parameters* γ, ϵ, *we get an* α-*approximation to the dissimilarity-HC objective, where* $\alpha = 0.667078 > \frac{2}{3}$.

Proof. The proof involves optimizing for the parameters ϵ, γ, δ and balancing out the two factors obtained from **Case 1** and **Case 2**. The final equation is:

$$\rho_{\text{GW}}\left(1 - \frac{1}{\gamma}\right)\left(1 - \frac{\epsilon/\delta}{1-\epsilon} - \frac{\delta\gamma}{1-\epsilon}\right) = \frac{2}{3(1-\epsilon)}$$

We defer the details of this proof to the Appendix A.5. This finishes the proof of Theorem 4.2.6. ∎

4.2.5 An improved 0.42-approximation for Moseley-Wang

Here we present polynomial-time 0.4246-approximation algorithms that use Max-Uncut Bisection as a subroutine for the maximization dual of Dasgupta's objective. The previous section presented a 0.336-approximation showing that one can beat the performance of Average-Linkage (also of a random tree) which achieves 1/3.

Max-Uncut Bisection: This is the complement problem to Min-Cut Bisection (which is perhaps more standard in the literature), and the goal here is to split the vertices of a weighted graph into two sets $(S, \bar{S})$, such that the weight of uncut edges $\sum_{ij \in E} w_{ij} - \sum_{i \in S, j \in \bar{S}} w_{ij}$ is maximized. It is known that one can achieve at least .8776 of the optimum value in polynomial time [10, 99].

Algorithm 7 HC via Max-Uncut Bisection

```
1: input: Similarity matrix $w \in \mathbb{R}^{n \times n}_{\geq 0}$.
2: Partition the underlying graph on $n$ vertices with edges weighted by $w$ into two parts $S$ and
   $\bar{S}$ using Max-Uncut Bisection as a black box. This creates the top split in the hierarchical
   clustering tree.
3: Run Average-Linkage on $S$ and on $\bar{S}$ to get trees $\mathcal{T}_S$ and $\mathcal{T}_{\bar{S}}$.
4: Construct the resulting HC tree by first splitting into $(S, \bar{S})$, then building trees $\mathcal{T}_S$ and $\mathcal{T}_{\bar{S}}$ on
   the respective sets.
```

Our main result is:

Theorem 4.2.7 *Given an instance of hierarchical clustering, our Algorithm 7 outputs a tree achieving* $\frac{4\rho}{3(2\rho+1)} - o(1) \geq .4246$ *(for* $\rho = 0.8776$*) of the optimum according to the Moseley-Wang objective, if a* ρ*-approximation for the* Max-Uncut Bisection *problem is used as a black-box.*

Remark: The current best approximation factor achievable for Max-Uncut Bisection in polynomial time is $\rho = 0.8776$. This makes our analysis almost tight, since one can't get better than .444 even by using an exact Max-Uncut Bisection algorithm (with $\rho = 1$).

Remark: While writing this thesis, a follow-up work by Alon et al. [5] showed that the same algorithm we presented here actually attains a $\frac{2}{3}\rho = 0.585$ approximation to the Moseley-Wang objective.

High-level Overview of Proof

Before delving into the technical details of the main proof, we present our high-level strategy through a series of 4 main steps:

Step 1: Consider a binary[2] tree $\mathcal{T}^*$ corresponding to the optimal solution for the hierarchical clustering problem and let $\texttt{OPT} = \mathcal{F}^+(\mathcal{T}^*)$ be the value of the objective function for this tree. Note that there exists a subtree $\widehat{\mathcal{T}^*}$ in this tree which contains more than $n/2$ leaves while its two children contain at most $n/2$ leaves each (see Figure 4.7). Given this decomposition of the optimum tree into three size restricted sets A, B, C, we provide an upper bound for $\texttt{OPT}$ as a function of the weight inside and across these sets (see Proposition 4.2.8). We then need to do a case analysis based on whether the weight across or inside these sets is larger.

Step 2: In the former case, things are easy as one can show that $\texttt{OPT}$ is *small* and that even the contribution from the Average-Linkage part of our algorithm alone yields a $\frac{4}{9}$-approximation. This is carried out in Proposition 4.2.10 based on the Fact 4.2.9.

Step 3: In the latter case, we show that there exists a split of the graph into two exactly equal pieces, so that the weight of the uncut edges is relatively *large*. This is crucial in the analysis as having a good solution to the Max-Uncut Bisection directly translates into a high value returned by the ρ-approximate black box algorithm (see Lemma 4.2.12, Proposition 4.2.11 and Proposition 4.2.12).

Step 4: Finally, from the previous step we know that the returned value of the black box is large, hence taking into account the form of the HC objective, we can derive a lower bound for the value our Algorithm 7. The proof of the main theorem is then completed by Proposition 4.2.13 and Lemma 4.2.13.

Proof of Theorem 4.2.7

For ease of presentation, we assume n is even, to avoid the floor/ceiling notation and we omit the $o(1)$ terms.

Proposition 4.2.8 *Let A be the set of leaves in the left subtree of $\widehat{\mathcal{T}^*}$, let B the set of leaves in the right subtree of $\widehat{\mathcal{T}^*}$ and $C = V \setminus (A \cup B)$ be the set of leaves outside of $\widehat{\mathcal{T}^*}$. Then*[3]*:*

$$\texttt{OPT} \leq (w(A) + w(B) + w(C)) \cdot (n-2) +$$

$$+ (w(A,B) + w(B,C) + w(A,C)) \cdot |C|$$

Proof. For an edge (i,j) whose endpoints lie in the same cluster (i.e., A, B or C), its contribution to the objective is at most $w_{ij}(n-2)$, using the trivial upper bound of $(n-2)$. Consider any pair of leaves $(i,j) \in A \times B$ in $\mathcal{T}^*$. The least common ancestor for this pair is the root of $\widehat{\mathcal{T}^*}$ and hence the contribution of this pair to the objective is equal to $w_{ij}(n - |\widehat{\mathcal{T}^*}|) = w_{ij}|C|$. Similarly, for any pair of leaves $(i,j) \in A \times C$ (or in $B \times C$), their least common ancestor is a predecessor of the root of

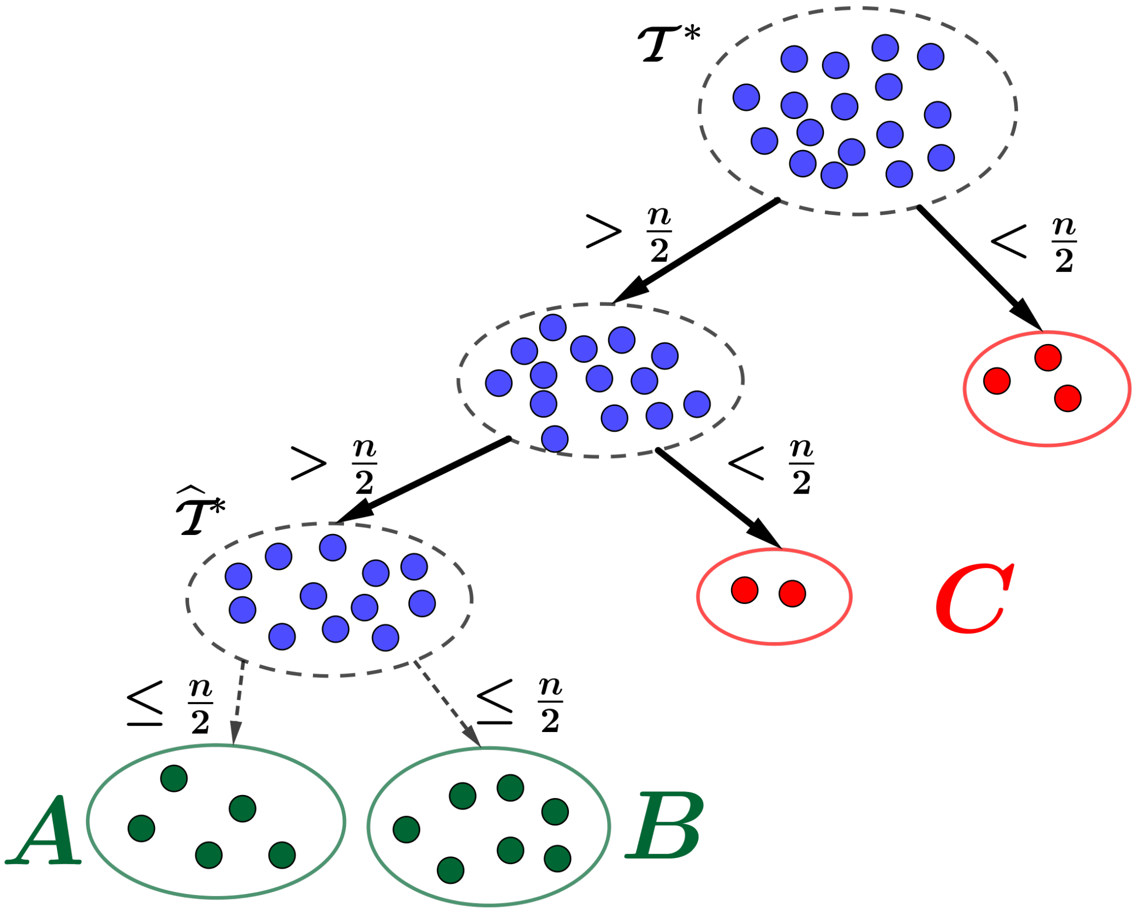

Figure 4.7: Splitting $\mathcal{T}^*$ to size restricted sets A, B, C.

$\widehat{\mathcal{T}^*}$ and hence the contribution of this pair to the objective is at most $w_{ij}(n - |\widehat{\mathcal{T}^*}|) = w_{ij}|C|$. The desired bound now follows by summing up all the contributions of all distinct pairs of leaves. ∎

From now on, let $\alpha \coloneqq w(A) + w(B) + w(C)$ and let $\beta \coloneqq w(A, B) + w(B, C) + w(A, C)$ denote the total weights of similarities inside the three sets and crossing a pair of these sets respectively. Note the total weight of all similarities is $W = \alpha + \beta$.

Fact 4.2.9 (Average-Linkage [78]) *The* Average-Linkage *algorithm gives a solution whose* $\mathcal{F}^+$ *objective is at least* $\frac{1}{3}W(n-2) = \frac{1}{3}(\alpha+\beta)(n-2)$.

Proposition 4.2.10 *If* $\alpha \leq \beta$, *our Algorithm 7 outputs a solution of value at least* $4/9\texttt{OPT} \geq 0.44\texttt{OPT}$, *where* $\texttt{OPT}$ *denotes the HC value of any optimum solution.*

Proof. Recall that by definition of C it holds that $|C| < \frac{n}{2} \leq \frac{n}{2} - 1 \leq \frac{n-2}{2}$. Hence by Proposition 4.2.8 we have $\texttt{OPT} \leq \alpha(n-2) + |C| \cdot \beta \leq \alpha(n-2) + \frac{n-2}{2}\beta$. On the other hand, by Fact 4.2.9, Average-Linkage outputs a solution whose expected value is $\frac{1}{3}(\alpha+\beta)(n-2)$. We have $\frac{1}{3}(\alpha+\beta)(n-2) - \frac{4}{9}(\alpha(n-2) + \frac{n-2}{2}\beta) = \frac{1}{9}(\beta - \alpha) \geq 0$. Hence, the Average-Linkage part of the algorithm alone gives a $\frac{4}{9}$-approximation in this case. ∎

Lemma 4.2.12 *Suppose* $\alpha \geq \beta$. *Then, there exists a balanced cut* (L, R) *of the nodes in* G, *such that the weight of the uncut edges is at least* $\alpha - (\alpha - \beta)\delta_{max}(c)$, *where* $c = |C|/n$ *and* $\delta_{max}(c) = \frac{c(1-2c)}{(1-3c^2)}$.

Proof. For the partition (A, B, C) we will refer to edges whose both endpoints are inside one of the three sets as *red* edges (i.e., $(i, j) \in (A \times A) \cup (B \times B) \cup (C \times C)$). We refer to the edges whose two endpoints are contained in two different sets as *blue* edges (i.e., $(i, j) \in (A \times B) \cup (A \times C) \cup (B \times C)$). Our goal here is to give a randomized partitioning scheme that produces the bisection (L, R) with high value of uncut weight lying inside L, R.

For simplicity, recall that n is even. The case of odd n is handled similarly. Denote $a = |A|/n$ and $b = |B|/n$. Let $\widetilde{a} = 1/2 - a$, $\widetilde{b} = 1/2 - b$, and $\widetilde{c} = 1/2 - c$. Note that $\widetilde{a}$, $\widetilde{b}$ are non-negative, and $\widetilde{c}$ is strictly positive due to the size restrictions. Define:

$$q_A = \frac{2\widetilde{b}\widetilde{c}}{(\widetilde{b}+\widetilde{c})^2}, \quad q_B = \frac{2\widetilde{a}\widetilde{c}}{(\widetilde{a}+\widetilde{c})^2}, \quad q_C = \frac{2\widetilde{a}\widetilde{b}}{(\widetilde{a}+\widetilde{b})^2},$$

and

$$p_A = \frac{q_B q_C}{q_A q_B + q_B q_C + q_A q_C},$$
$$p_B = \frac{q_A q_C}{q_A q_B + q_B q_C + q_A q_C},$$
$$p_C = \frac{q_A q_B}{q_A q_B + q_B q_C + q_A q_C}.$$

We also denote the following expression by δ:

$$\delta = \frac{q_A\, q_B\, q_C}{q_A q_B + q_B q_C + q_A q_C}.$$

Consider the following partitioning procedure:

- Pick one of the sets A, B, or C with probability p_A, p_B, and p_C, respectively (note that $p_A + p_B + p_C = 1$).
- If the chosen set is A, partition it into two random sets S_B and S_C of size $\widetilde{b}|A|/(\widetilde{b}+\widetilde{c})$ and $\widetilde{c}|A|/(\widetilde{b}+\widetilde{c})$ and output the cut $L = B \cup S_B$, $R = C \cup S_C$.
- Similarly, if the chosen set is B, we partition it into two random sets S_A and S_C of size $\widetilde{a}|B|/(\widetilde{a}+\widetilde{c})$ and $\widetilde{c}|B|/(\widetilde{a}+\widetilde{c})$ and output the cut $L = C \cup S_C$, $R = A \cup S_A$.
- If the chosen set is C, we partition it into two random sets S_A and S_B of size $\widetilde{a}|C|/(\widetilde{a}+\widetilde{b})$ and $\widetilde{b}|C|/(\widetilde{a}+\widetilde{b})$ and output the cut $L = A \cup S_A$, $R = B \cup S_B$.

We first observe that each of the output sets L and R has $n/2$ vertices, i.e., (L, R) is a bisection of the graph. If for instance, the algorithm picks set A at the first step, then the set L contains $|B| + \widetilde{b}|A|/(\widetilde{b}+\widetilde{c})$ vertices. We have

$$|L| = |B| + \frac{\widetilde{b}}{\widetilde{b}+\widetilde{c}}|A| = bn + \frac{1/2 - b}{1-b-c} \cdot an =$$

$$= bn + \frac{1/2-b}{a} \cdot an = \frac{n}{2}.$$

The set R is the complement to L, thus, it also contains $n/2$ vertices. The cases when the algorithm picks the set B or C are identical.

We now compute the expected weight of red edges in the bisection (L, R).

Proposition 4.2.11 *The expected weight of uncut red edges is* $(1-\delta)\alpha$.

Proof. Again, assume that the algorithm picks the set A at the first step. Then, the sets B and C are contained in the sets L and R, respectively. Consequently, no edges in B and C are in the cut between L and R. Every edge in A is cut with probability $2\widetilde{b}\widetilde{c}/(\widetilde{b}+\widetilde{c})^2$. Thus, the weight of red edges in the cut between L and R (denoted as $E^{red}(L, R)$) given that the algorithm picks set A equals

$$\mathbb{E}|E^{red}(L,R)| \text{ algorithm picks } A \text{ at first step} =$$

$$= \frac{2\widetilde{b}\widetilde{c}}{(\widetilde{b}+\widetilde{c})^2} w(A) = q_A w(A).$$

Similarly, if the algorithm picks the set B or C, the expected sizes of the cuts equal $q_B w(B)$ and $q_C w(C)$, respectively. Hence, the expected weight of the red edges between L and R (when we do not condition on the first step of the algorithm) equals

$$\mathbb{E}|E^{red}(L,R)| = p_A q_A w(A) + p_B q_B w(B) + p_C\, q_C w(C)$$

Observe that

$$p_A q_A = p_B q_B = p_C q_C = \frac{q_A\, q_B\, q_C}{q_A q_B + q_B q_C + q_A q_C} = \delta.$$

Then, the expected weight of red edges between L and R equals:

$$\mathbb{E}|E^{red}(L,R)| = \delta(w(A) + w(B) + w(C)) = \delta\alpha.$$

Here, we used that $w(A) + w(B) + w(C) = \alpha$. The expected weight of uncut red edges equals $(1-\delta)\alpha$. ▮

We now lower bound the weight of uncut blue edges.

Proposition 4.2.12 *The expected weight of uncut blue edges is at least $\delta\beta$.*

Proof. We separately consider edges between sets A and B, B and C, A and C. Consider an edge $(u,v) \in A \times B$. This edge is not in the cut (L,R) if both endpoints u and v belong to L or both endpoints belong to R. The former event – $\{u,v \in L\}$ – occurs if the set B is chosen in the first step of the algorithm and the set S_A contains vertex v; the latter event – $\{u,v \in R\}$ – occurs if A is chosen in the first step of the algorithm and the set S_B contains vertex u. The probability of the union of these events is[4]

$$[(u,v) \notin (L,R)] = p_B \cdot \frac{\widetilde{a}}{\widetilde{a}+\widetilde{c}} + p_A \cdot \frac{\widetilde{b}}{\widetilde{b}+\widetilde{c}} =$$

$$= p_B q_B \cdot \frac{(\widetilde{a}+\widetilde{c})}{2\widetilde{c}} + p_A q_A \cdot \frac{(\widetilde{b}+\widetilde{c})}{2\widetilde{c}}.$$

Since $p_A q_A = p_B q_B = \delta$, we have

$$[(u,v) \notin (L,R)] = \delta \cdot \frac{\widetilde{a}+\widetilde{b}+2\widetilde{c}}{2\widetilde{c}} = \delta \cdot \frac{1/2+\widetilde{c}}{2\widetilde{c}} \geq \delta.$$

The last inequality holds because $(1/2+\widetilde{c})/\widetilde{c} \geq 2$ for all $\widetilde{c} \in (0,1/2]$. The same bound holds for edges between sets B and C and sets A and C. Therefore, the expected weight of uncut blue edges is at least $\delta\beta$. ▮

By the above two propositions (Proposition 4.2.11 and Proposition 4.2.12) the expected total weight of uncut edges is at least:

$$\mathbb{E}w(L) + w(R) \geq (1-\delta)\alpha + \delta\beta = \alpha - (\alpha - \beta)\delta.$$

Note that we are in the case with $\alpha - \beta \geq 0$. Thus to establish a lower bound on the expectation, we need to show an upper bound on δ. Write

$$\delta = \frac{q_A\, q_B\, q_C}{q_A q_B + q_B q_C + q_A q_C} = \frac{1}{\frac{1}{q_A} + \frac{1}{q_B} + \frac{1}{q_C}}.$$

After plugging in the values of q_A, q_B, and q_C, we obtain the following expression for δ.

$$\delta = \frac{1}{\frac{(\widetilde{b}+\widetilde{c})^2}{2\widetilde{b}\widetilde{c}} + \frac{(\widetilde{a}+\widetilde{c})^2}{2\widetilde{a}\widetilde{c}} + \frac{(\widetilde{a}+\widetilde{b})^2}{2\widetilde{a}\widetilde{b}}} =$$

$$= \frac{1}{3 + \frac{1}{2}\left(\frac{\widetilde{b}+\widetilde{c}}{\widetilde{a}} + \frac{\widetilde{a}+\widetilde{c}}{\widetilde{b}} + \frac{\widetilde{a}+\widetilde{b}}{\widetilde{c}}\right)}$$

Observe that $a + b + c = 1$ and $\widetilde{a} + \widetilde{b} + \widetilde{c} = 1/2$. Thus, $\widetilde{b} + \widetilde{c} = 1/2 - \widetilde{a}$, $\widetilde{a} + \widetilde{c} = 1/2 - \widetilde{b}$, and $\widetilde{a} + \widetilde{b} = 1/2 - \widetilde{c}$. Hence,

$$\delta = \frac{1}{3 + \frac{1}{2}\left(\frac{1/2-\widetilde{a}}{\widetilde{a}} + \frac{1/2-\widetilde{b}}{\widetilde{b}} + \frac{1/2-\widetilde{c}}{\widetilde{c}}\right)} =$$

$$= \frac{1}{\frac{3}{2} + \frac{1}{4}\left(\frac{1}{\widetilde{a}} + \frac{1}{\widetilde{b}} + \frac{1}{\widetilde{c}}\right)}.$$

Note that since the function $t \mapsto 1/t$ is convex for $t > 0$, we have

$$\frac{1}{2}\left(\frac{1}{\widetilde{a}} + \frac{1}{\widetilde{b}}\right) \geq \frac{2}{\widetilde{a} + \widetilde{b}} = \frac{2}{1/2 - \widetilde{c}} = \frac{2}{c}$$

Therefore,

$$\delta \leq \frac{1}{\frac{3}{2} + \frac{1}{c} + \frac{1}{4\widetilde{c}}} = \frac{c(1-2c)}{1-3c^2}.$$

We conclude that the expected weight of uncut edges is at least $\alpha - (\alpha - \beta)\delta_{max}(c)$, where $\delta_{max}(c) = c(1-2c)/(1-3c^2)$. ∎

Proposition 4.2.13 *Let $\rho = 0.8776$ be the approximation factor of the* Max-Uncut Bisection *algorithm [10, 99]. Then if $\beta \leq \alpha$, our Algorithm 7 outputs a solution of value at least $\frac{2\rho}{3}(n-1)((1-\delta_{max}(c))\alpha + \delta_{max}(c)\beta)$.*

Proof. Let (L, R) be the bisection produced by the ρ-approximate Max-Uncut Bisection algorithm. This partition satisfies:

$$w(L) + w(R) \geq \rho \texttt{OPT}_{\text{Max-Uncut Bisection}}$$

Our Algorithm 7 produces a tree where at the top level the left subtree is L, the right subtree is R and both of these subtrees are then generated by AVERAGE-LINKAGE. Hence each edge $(i,j) \in L \times L$ (and similarly for edges in $R \times R$) contributes:

$$w_{ij}\left(\tfrac{n}{2} + \tfrac{1}{3}(\tfrac{n}{2} - 2)\right) = \tfrac{2}{3} w_{ij}(n-1)$$

to the objective. Thus the overall value of our solution is at least:

$$\tfrac{2}{3}(n-1)(w(L)+w(R)) \geq$$

$$\geq \frac{2\rho}{3}(n-1) \cdot \texttt{OPT}_{\textsc{Max-Uncut Bisection}}$$

If $\beta \leq \alpha$ then by Lemma 4.2.12 we have that $\texttt{OPT}_{\textsc{Max-Uncut Bisection}} \geq \alpha - (\alpha - \beta)\delta_{max}(c)$ and the proof follows by rearranging the terms. ▮

Lemma 4.2.13 *The approximation factor ξ of our Algorithm 7 is at least* $\frac{4\rho}{3(2\rho+1)} \geq 0.42469$.

Proof. First, note that if $\beta \geq \alpha$ then by Proposition 4.2.10 the approximation is at least 0.44. Hence it suffices to only consider the case when $\beta \leq \alpha$. Recall that by Fact 4.2.9, AVERAGE-LINKAGE outputs a solution of value $\frac{1}{3}(\alpha+\beta)(n-2)$ and by Proposition 4.2.8, we have $\texttt{OPT} \leq \alpha(n-2) + |C|\beta$. Hence if $\frac{1}{3}(\alpha+\beta)(n-2) \geq \xi\left(\alpha(n-2)+|C|\beta\right)$ then the desired approximation holds.

Thus we only need to consider the case when $\frac{1}{3}(\alpha+\beta)(n-2) \leq \xi\left(\alpha(n-2)+|C|\beta\right)$ or equivalently:

$$\frac{1}{3}(\alpha+\beta) \leq \xi\left(\alpha + \tfrac{|C|}{n-2}\beta\right) \iff$$

$$\iff \beta \leq \frac{3\xi - 1}{1 - 3\frac{|C|}{n-2}\xi}\alpha.$$

Let $c_1 = \frac{2\rho}{3}(1 - \delta_{max}(c))$ and $c_2 = \frac{2\rho}{3}\delta_{max}(c)$. In this case by Proposition 4.2.13, our Algorithm 7 gives value at least $c_1(n-1)\alpha + c_2(n-1)\beta$. Hence it suffices to show that $c_1(n-2)\alpha + c_2(n-2)\beta \geq \xi(\alpha(n-2)+|C|\beta)$. Or equivalently that:

$$\beta(\tfrac{|C|}{n-2}\xi - c_2) \leq \alpha(c_1 - \xi)$$

Using the bound on β above it suffices to show that:

$$\frac{3\xi - 1}{1 - 3\frac{|C|}{n-2}\xi}(\tfrac{|C|}{n-2}\xi - c_2) \leq c_1 - \xi.$$

After simplifying this expression the above bound holds for:

$$\xi \leq \frac{c_1 - c_2}{1 + 3\frac{cn}{n-2}c_1 - \frac{cn}{n-2} - 3c_2}.$$

Hence it suffices to find the minimum of the RHS over $c \in [0, \frac{1}{2} - \frac{1}{n}]$. Plugging in the expressions for c_1 and c_2 after simplification the RHS is equal to:

$$\frac{2}{3} \frac{\rho(1-c)}{2c^2\rho + (1-3c^2)}$$

Differentiating over c one can show that the minimum of this expression is attained for $c = \frac{1}{2} - \frac{1}{n}$. Indeed, the numerator of the derivative is quadratic function with negative leading coefficient whose roots are $1 \pm \frac{\sqrt{t(t+1)}}{t}$ for $t = 2\rho - 3$. The left root is approximately 0.545 and hence the derivative is negative on $[0, \frac{1}{2} - \frac{1}{n}]$. The value at the minimum $c = \frac{1}{2} - \frac{1}{n}$ is thus equal[5] to ($\rho = 0.8776$):

$$\frac{4\rho}{3(2\rho+1)} \geq 0.42469$$

▮

4.2.6 Hardness for Moseley-Wang via Small Set Expansion

We complement our positive results in the previous section by providing APX-hardness for the Moseley-Wang objective (even for 0-1 similarities), under the Small Set Expansion hypothesis.

Theorem 4.2.14 *Under the* Small Set Expansion *(*SSE*) hypothesis, there exists $\epsilon > 0$, such that it is NP-hard to approximate the Moseley-Wang HC objective function within a factor $(1-\epsilon)$.*

Initially introduced by Raghavendra and Steurer [82], SSE has been used to prove improved hardness results for optimization problems including Balanced Separator and Minimum Linear Arrangement [83].

Given a d-regular, unweighted graph $G = (V, E)$ and $S \subseteq V$, let $\mu(S) := |S|/|V|$ and $\Phi(S) := |E(S, V \setminus S)|/d|S|$. Raghavendra et al.[83] prove the following strong hardness result. (While it is not explicitly stated that the result holds for regular graphs, it can be checked that their reduction produces a regular graph [93].)

Theorem 4.2.15 (Theorem 3.6 of [83]) *Assuming the* SSE*, for any $q \in \mathbb{N}$ and $\epsilon, \gamma > 0$, given a regular graph $G = (V, E)$, it is NP-hard to distinguish the following two cases.*

- *YES: There exist q disjoint sets $S_1, \ldots, S_q \subseteq V$ such that for all $\ell \in [q]$,*

$$\mu(S_\ell) = 1/q \qquad \text{and} \qquad \Phi(S_\ell) \leq \epsilon + o(\epsilon).$$

- *NO: For all sets $S \subseteq V$,*

$$\Phi(S) \geq \phi_{1-\epsilon/2}(\mu(S)) - \gamma/\mu(S)$$

where $\phi_{1-\epsilon/2}(\mu(S))$ is the expansion of the sets of volume $\mu(S)$ in the infinite Gaussian graph with correlation $1-\epsilon/2$.

Proof. [Proof of Theorem 4.2.14] Let us consider the instance of Hierarchical Clustering defined by the same graph where each pair has weight 1 if there is an edge, and 0 otherwise. Then $W = |E|$ is the total weight.

- YES: The fraction of edges crossing between different S_i's is at most $\epsilon + o(\epsilon)$, and all edges inside some S_i are multiplied by at least $n(1-1/q)$ in the objective function. So the objective function for Hierarchical clustering is at least

$$(1-\epsilon-o(\epsilon))W\cdot(1-\tfrac{1}{q})n \geq nW(1-\tfrac{1}{q}-\epsilon-o(\epsilon)).$$

- NO: Consider an arbitrary binary tree $\mathcal{T}$ that maximizes the Moseley-Wang objective function. For a tree node $a \in \mathcal{T}$, let $\mathcal{T}_a$ be the subtree of $\mathcal{T}$ rooted at a, and $V_a \subseteq V$ be the set of graph vertices corresponding the leaves of T_a.

  Let $b \in \mathcal{T}$ be a highest node such that $n/3 \leq |V_b| \leq 2n/3$ (such a node always exists in a binary tree). By Theorem 4.2.15, we have

$$\Phi(V_b) \geq \phi_{1-\epsilon/2}(\mu(V_b)) - \gamma/\mu(V_b) \geq C\sqrt{\epsilon}$$

  for some absolute constant C. Here we use the fact that

$$\phi_{1-\epsilon/2}(\mu(V_b)) \geq \Omega(\sqrt{\epsilon}) \text{ for } \mu(V_b) \in [1/3, 2/3]$$

  and take γ small enough depending on ϵ.

  So the total fraction of edges in $E(V_b, V \setminus V_b)$ is at least

$$\mu(V_b)\cdot\Phi(V_b) \geq \tfrac{C\sqrt{\epsilon}}{3}.$$

  Note that edges in $E(T_b, V \setminus T_b)$ will multiplied by at most $n/3$ in the objective function. (Let a be the parent of b. Then $|V_a| > 2n/3$ by the choice of b and for any edge crossing V_b, the the least common ancestors of the two endpoints will be a or one if its ancestors.) Therefore, the objective function is at most

$$nW - \tfrac{C\sqrt{\epsilon}}{3}W\cdot\tfrac{2n}{3} = nW(1-\tfrac{2C\sqrt{\epsilon}}{9}).$$

Therefore, the value is at least $nW(1-1/q-\epsilon-o(\epsilon))$ in the YES case and $nW(1-(2C\sqrt{\epsilon}/9))$ in the NO case. By taking $\epsilon > 0$ sufficiently small and q arbitrarily large, there is a constant gap between the YES case value and the NO case value. ∎

Chapter 5

Hierarchical Clustering for Euclidean Data

What's the most you ever lost on a coin toss?

-No Country for Old Men (2007)

In previous chapters, we were given pairwise similarity information in the input; however, in practice we may have access to informative features about our data points. For example, a new YouTube video is uploaded and we try to gather in an automatic way if this video talks about cats or not, if it is long or short, funny or serious, offensive or not etc. All these are features and the goal then is to find a good hierarchical clustering based on the features.

Contributions. We will start with the simplest case of 1-dimensional Euclidean data. Even in this seemingly simple setting, obtaining an efficient algorithm that produces an exactly optimum solution seems non-trivial, motivating the study of approximation algorithms. We will prove that two algorithms – Random Cut (RC) and Average-Linkage (AL) – obtain $\frac{1}{2}$-approximation of the optimal solution as measured by the Moseley-Wang objective (HC-OBJ-2). Average-Linkage achieves this deterministically, while RC only in expectation. This beats the best known approximation for the general case, which is 0.336 [35] as we saw in the previous chapter. Here RC is substantially faster than Average-Linkage with a running time of $O(n \log n)$ vs. $O(n^2 \log n)$.

We next consider the high-dimensional case with the Gaussian Kernel and show that Average-Linkage cannot beat the factor $\frac{1}{3}$ even in poly-logarithmic dimensions. We propose the Projected Random Cut (PRC) algorithm that gets a constant improvement over $\frac{1}{3}$, irrespective of the dimension d (the improvement is a function of the ratio of the diameter to σ^2, and drops as this ratio gets large). Furthermore, a simple implementation of PRC runs in $O(n(d + \log n))$ time while Average-Linkage runs in $O(dn^2 \log n)$ time. Even single-linkage (equivalent to finding a minimum spanning tree) runs in almost-linear time only for constant d and has exponential dependence on the dimension (see e.g., [100]) and it is open whether it can be scaled to large d when n is large.

Experiments. Many existing algorithms have time efficiency shortcomings, and none of them can be used for really large datasets. On the contrary, our Projected Random Cut (see Section 5.4) is a fast HC algorithm that scales to the largest ML datasets. The running time scales almost linearly and the algorithm can be implemented in one pass without needing to store similarities, so the memory is $O(n)$. We also evaluate its quality on a small dataset (Zoo).

5.1 Setting: Feature Vectors with Gaussian Kernel

The main protagonist here will be the Moseley-Wang objective (HC-OBJ-2), which we will denote as $\mathcal{F}^+$ in our equations. As we know,Average-Linkage obtains a $\frac{1}{3}$-approximation for maximizing this objective function, and we showed in the previous chapter, there is an SDP-based algorithm that achieves a $(\frac{1}{3} + \epsilon)$-approximation for the problem, for small constant ϵ.

One drawback of the prior work on these hierarchical clustering objectives is the fact that they all consider arbitrary similarity scores (specified as an $n \times n$ matrix); however, there is much more structure to such similarity scores in practice and here we study the commonly encountered case of *Euclidean data*, where the similarity score w_{ij} is computed by applying a monotone decreasing function to the Euclidean distance between i and j. Roughly speaking, we show how to exploit this structure to design improved approximation/faster algorithms for hierarchical clustering, and how to re-analyze algorithms commonly used in practice.

Arguably the most common distance-based similarity measure used for Euclidean data is the Gaussian kernel. Here we use the spherical version $w_{ij} \sim \exp(-\|v_i - v_j\|/2\sigma^2)$. The parameter σ^2, referred to as the *bandwidth*, plays an important role for applications and a large body of literature exists on selection of this parameter [101].

While it might seem that the case of Euclidean data with the Gaussian kernel is a very restricted class of inputs to the HC problem, we show that for suitably high dimensions and suitable small choice of bandwidth σ^2 it can simulate arbitrary similarity scores (scaled appropriately). Thus any improvements to approximation guarantees for the Euclidean case (that do not apply to general similar scores) must necessarily involve assumptions about the dimension of the data (not very high) or on σ^2 (not pathologically small). Such assumptions on σ^2 are consistent with common methods for computing the bandwidth parameter based on data (e.g., [101]).

Euclidean data. We consider data sets represented as sets of d-dimensional *feature* vectors. Suppose these vectors are $v_1, \ldots, v_n \in \mathbb{R}^d$. We focus on similarity measures between pairs of data points, denoted by $[w_{ij}]_{i,j\in[n]}$, where the similarities only depend on the underlying vectors, i.e.,

$w_{ij} = f(v_i, v_j)$ for some function $f : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ and furthermore are fully determined by monotone functions of distances between them.

Definition 5.1.1 (Distance-based similarity measure) *A similarity measure $w_{ij} = f(v_i, v_j)$ is "distance-based" if $f(v_i, v_j) = g(\|v_i - v_j\|_2)$ for some function $g \colon \mathbb{R} \to [0, 1]$, and is "monotone distance-based" if furthermore $g \colon \mathbb{R} \to [0, 1]$ is a monotone non-increasing function.*

As an example of the monotone similarity measure it is natural to consider the *Gaussian kernel similarity*, i.e.,

$$w_{ij} = (\sqrt{2\pi}\sigma)^{-n} e^{-\frac{\|v_i - v_j\|_2^2}{2\sigma^2}}, \qquad \text{(\textit{Gaussian Kernel})}$$

where σ is a normalization factor determining the *bandwidth* of the Gaussian kernel [54]. For simplicity, we ignore the multiplicative factor $(\sqrt{2\pi}\sigma)^{-n}$ (unless noted otherwise), as our focus is on multiplicative approximations and scaling has no effects.

General upper bounds. In order to analyze the linkage-based clustering algorithms and our proposed algorithms, we propose a natural upper bound on the value of the $\mathcal{F}^+$ objective function. The idea is to decompose the objective function $\mathcal{F}^+$ into contributions of *triple* of vertices:

$$\mathcal{F}^+(\mathcal{T}) = \sum_{i<j<k} \left(w_{ij} \mathbb{1}\left[\mathcal{E}_{ij}^k\right] + w_{jk} \mathbb{1}\left[\mathcal{E}_{jk}^i\right] + w_{ik} \mathbb{1}\left[\mathcal{E}_{ik}^j\right] \right)$$

where $\mathcal{E}_{yz}^x$ denotes the event that x is separated first among the vertices of triple $\{x, y, z\}$ in tree $\mathcal{T}$. Note that the final tree scores only one of the similarity weights between the triple $\{i, j, k\}$. Given this observation, we define the following benchmark:

$$\texttt{MAX-upper} \triangleq \sum_{i<j<k} \max(w_{ij}, w_{jk}, w_{ik}),$$

Clearly, for all trees $\mathcal{T}$, $\mathcal{F}^+(\mathcal{T}) \leq \texttt{MAX-upper}$.

One-dimensional benchmarks. Consider the special case of 1D data points (with any monotone distance-based similarity measure as in Definition 5.1.1), and suppose $v_1 \leq v_2 \leq \ldots \leq v_n \in \mathbb{R}$. Now, for any triple $i < j < k$, as a simple observation we have $w_{ik} \leq \min(w_{ij}, w_{jk})$. Hence we can modify the above benchmark to obtain two refined new benchmarks:

$$\texttt{1D-MAX-upper} \triangleq \sum_{i<j<k} \max(w_{ij}, w_{jk}),$$

$$\texttt{1D-SUM-upper} \triangleq \sum_{i<j<k} (w_{ij} + w_{jk}).$$

Again, clearly for all trees $\mathcal{T}$ we have:

$$\mathcal{F}^+(\mathcal{T}) \leq \texttt{1D-MAX-upper} \leq \texttt{1D-SUM-upper}$$

5.2 Performance of Average Linkage

In this section we look at the extreme case where the feature vectors have $d = 1$, and we try to analyze the popular/natural algorithms existing in this domain by evaluating how well they approximate the objective function $\mathcal{F}^+$. We focus on average-linkage and Random Cut (will be formally defined later). In particular, Random Cut is a building block of our algorithm for high-dimensional data given in Section 5.4.

We show that Average-Linkage gives a 1/2-approximation, and can obtain no better than 3/4 fraction of the optimal objective value in the worst-case. We then show that Random Cut also is a 1/2-approximation (in expectation) and this factor is tight. In Section 5.3, we discuss other simple algorithms: single-linkage and greedy cutting (will be defined formally later). We start by the observation that greedy cutting and single-linkage output the same tree (and so are equivalent). We finish by showing that there is an instance where single-linkage attains only $\frac{1}{2}$ of the optimum objective value. We further show on this instance both average-linkage and random cutting are almost optimal.

For the rest of this section, suppose we have 1D points $x_1 \leq \ldots, \leq x_n \in \mathbb{R}$, where $w_{ij} = g(\|x_i - x_j\|)$ for some monotone non-increasing function $g : \mathbb{R} \to [0, 1]$.

Average-linkage. In order to simplify the notation, we define w_{AB} to be $\sum_{i\in A, j\in B} w_{ij}$ for any two sets A and B. We first prove a simple structural property of average-linkage in 1D.

Lemma 5.2.1 *For $d = 1$, under any monotone distance-based similarity measure $w_{ij} = g(\|x_i - x_j\|)$, average-linkage can always merge neighbouring clusters.*

Proof. We do the proof by induction. This property holds at the first step of average-linkage (base of induction). Now suppose up to a particular step of average-linkage, the algorithm has always merged neighbours. At this step, we have an ordered collection of super nodes $(C_1, \ldots, C_m)$, where for every $i < j$ and for all $(x, y) \in C_i \times C_j$, $x \leq y$. If at this step average-linkage doesn't merge two neighbouring super nodes, then there exists $i < j < k$, where

$$\frac{w_{C_iC_j}}{|C_i||C_j|} < \frac{w_{C_iC_k}}{|C_i||C_k|} \Rightarrow \frac{\sum_{x\in C_i} w_{xC_j}}{|C_j|} < \frac{\sum_{x\in C_i} w_{xC_k}}{|C_k|}$$

But note that because of the monotone non-increasing distance-based similarity weights, for any triple $(x, y, z) \in C_i \times C_j \times C_k$ we have $w_{xy} \geq w_{xz}$, This is a contradiction to the above inequality, which finishes the inductive step. ▮

Theorem 5.2.1 *For $d = 1$, under any monotone distance-based similarity measure $w_{ij} = g(\|x_i - x_j\|)$, average-linkage obtains at least $\frac{1}{2}$ of the* `1D-SUM-upper`*, and hence is a $\frac{1}{2}$-approximation for the objective $\mathcal{F}^+$.*

Proof. The proof uses a potential function argument. Given a partitioning of points $x_1 \leq x_2 \leq \ldots \leq x_n$ into sets $S_1, \ldots, S_m$, a triple of points $i < j < k$ is called *separated* if no pair of these three points belong to the same set. Now, the potential function Φ gets $\{S_1, \ldots, S_m\}$ as input, and maps it to a summation over all separated triples by $\{S_i\}_{i \in [m]}$ as below:

$$\Phi(S_1, \ldots, S_m) = \sum_{i<j<k:\ (i,j,k) \text{ is separated}} (w_{ij} + w_{jk})$$

Note that $\Phi(\{x_1\}, \ldots, \{x_n\}) = \texttt{1D-SUM-upper}$, and $\Phi(\{x_1, \ldots, x_n\}) = 0$.

We now run average-linkage. Based on the definition of $\mathcal{F}^+$, every time that average-linkage merges two super nodes A and B it scores $w_{AB} \cdot (n - |A| - |B|)$, and sum over of all these per-step scores is equal to its final objective value. Let `Score-AL` denotes the variable that stores of the score of average-linkage over time. At every step we keep track of (1) the change in the potential function, denoted by $\Delta\Phi$ and (2) how much progress average-linkage had towards the final objective value, denoted by Δ`Score-AL`. In order to prove 1/2-approximation, it suffices to show that at every step of average-linkage we have:

$$\Delta\texttt{Score-AL} + \frac{1}{2}\Delta\Phi \geq 0. \tag{$\star$}$$

To see this note that average-linkage starts with all points separated, and ends with one cluster/super node with all the points. Therefore, by summing eq. ($\star$) over all merging steps of average-linkage and canceling the terms in the telescopic sum, we have:

$$\begin{aligned}&(\mathcal{F}^+(\mathcal{T}_{\text{AL}}) - 0) + \\ &\frac{1}{2}\left(\Phi(\{x_1, \ldots, x_n\}) - \Phi(\{x_1\}, \ldots, \{x_n\})\right) \geq 0.\end{aligned}$$

Plugging values of Φ at the start and the end, we get:

$$\mathcal{F}^+(\mathcal{T}_{\text{AL}}) \geq \frac{1}{2}(\texttt{1D-SUM-upper}),$$

which implies the 1/2-approximation factor.

To prove eq. ($\star$), we focus on a single step of average-linkage where by Theorem 5.2.1 some two neighboring clusters denoted as A and B get merged. Let C denote the nodes on the left of A and let D denote the nodes on the right of B (Figure 5.1).[1] By merging two clusters A, B, the change in the score of average-linkage is:

$$\Delta\texttt{Score-AL} = w_{AB}(|C| + |D|)$$

Moreover, any separated triple $i < j < k$ such that either $i \in C, j \in A, k \in B$ or $i \in A, j \in B, k \in D$ will not be separated anymore after this merge. For each such triple, the potential function drops by $w_{ij} + w_{jk}$. Therefore:

$$-\Delta\Phi = (w_{AB}|C| + w_{AC}|B|) + (w_{AB}|D| + w_{BD}|A|)$$

To compare the two, we show that $w_{AB}(|C| + |D|) \geq w_{AC}|B| + w_{BD}|A|$, and hence:

$$\begin{aligned}\Delta\texttt{Score-AL} &= w_{AB}(|C| + |D|) \geq w_{AC}|B| + w_{BD}|A| \\ &= -\Delta\Phi - \Delta\texttt{Score-AL}\ ,\end{aligned}$$

which implies eq. ($\star$) as desired. To prove the last claim, note that average-linkage picks the pair (A, B) over both (A, C) and (B, D). Therefore, by definition of average-linkage:

$$\frac{w_{AB}}{|A||B|} \geq \frac{w_{AC}}{|A||C|} \implies w_{AB}|C| \geq w_{AC}|B|$$

$$\frac{w_{AB}}{|A||B|} \geq \frac{w_{BD}}{|B||D|} \implies w_{AB}|D| \geq w_{BD}|A|$$

By summing above inequalities, we get $w_{AB}(|C| + |D|) \geq w_{AC}|B| + w_{BD}|A|$, which finishes the proof. ▮

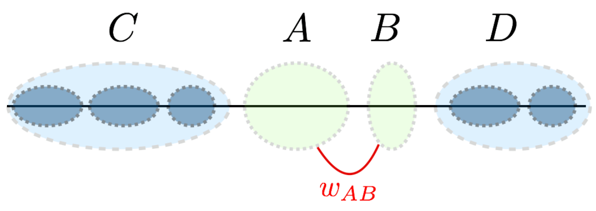

Figure 5.1: Illustration of the merging process in 1D.

As a final note, in Section 5.5 we discuss hard instances for average-linkage under Gaussian kernels when $d = 1$, where we essentially show the result of Theorem 5.2.1 is tight when comparing against `1D-SUM-upper`, and there is no hope to get an approximation factor better than $\frac{3}{4}$ for average-linkage in general for $d = 1$.

Random Cut. The following algorithm (termed as RANDOM CUT) picks a uniformly random point r in the range $[x_1, x_n]$ and divides the set of points into left and right using r as the splitter. The same process is applied recursively until the leaves are reached.

Lemma 5.2.2 *For $d = 1$ under any monotone distance-based similarity measure $w_{ij} = g(x_i, x_j)$ the algorithm* RANDOM CUT *obtains at least $1/2$ fraction of the* ***`1D-MAX-upper`****, and hence gives a*

Algorithm 8 Random Cut

```
  Input:  Integer $n$, points $x_1 \le \dots \le x_n$.
  Output: Binary tree with leaves $(x_1, \dots, x_n)$

  if $n == 1$ then
    return New leaf containing $x_1$.
  end if
  Pick $r \sim U([x_1, x_n])$
  Let $m$ be the largest integer such that $x_m \le r$.
  Create new internal tree node $x$.
  $x.left$ = Random Cut($m, x_1, \dots, x_m$)
  $x.right$ = Random Cut($n - m, x_{m+1}, \dots, x_n$)
  return $x$
```

$\frac{1}{2}$-approximation for the objective $\mathcal{F}^+$ in expectation.

Proof. For every triple $i < j < k$ conditioned on partitioning the interval $[i, k]$ for the first time the longer edge amongst (x_i, x_j) and (x_j, x_k) gets cut with probability $p_1 \ge 1/2$ and the shorter with probability $p_2 \le 1/2$ so that $p_1 + p_2 = 1$. W.l.o.g let's assume that (x_i, x_j) is the longer edge. Then the algorithm Random Cut scores $w_{jk}p_1 + w_{ij}(1 - p_1)$ in expectation for the (i, j, k) triple. Note that:

$$w_{jk}p_1 + w_{ij}(1-p_1) = (w_{jk} - w_{ij})\left(p_1 - \frac{1}{2}\right) + \frac{1}{2}(w_{ij} + w_{jk}) \ge \frac{1}{2}(w_{ij} + w_{jk})$$

By the linearity of expectation taking the sum over all triples $i < j < k$ Random Cut scores at least 1/2 of `1D-MAX-upper` in expectation and hence gives $\frac{1}{2}$-approximation for the objective $\mathcal{F}^+$. ∎

5.3 Greedy Cutting and Single-linkage.

Consider a simple algorithm, denoted by Greedy Cut, that picks the interval with maximum length among $\{[x_i, x_i + 1]\}_{i=1:n-1}$ (lets say $[x_m, x_{m+1}]$), and repeats the same operation recursively on $(x_1, \dots, x_m)$ and $(x_{m+1}, \dots, x_n)$ until the leaves are reached.

Lemma 5.3.1 *For $d = 1$ and under any monotone distance-based similarity measure g* Greedy Cut *and single-linkage return the same HC tree. Moreover, the edges picked by* Greedy Cut *are exactly the same edges picked by single-linkage, picked in reverse order.*

Proof. It is known that single-linkage is essentially the Kruskal algorithm (and hence edges picked by single-linkage form a Maximum Spanning Tree (MST)). Clearly, for any monotone distance-based measure the line connecting x_1 to x_n is the unique MST (as any tree can be shortcut-ed with this line), and hence single-linkage picks intervals $\{[x_i, x_{i+1}]\}_{i=1:n-1}$ in increasing order of their lengths. At the same time, Greedy Cut picks also the same intervals, but in decreasing order of their length. Moreover, because single-linkage merges the edges picked by GREEDY CUT in the reverse ordering (and it creates the HC tree from bottom to top), it returns the same HC tree as GREEDY CUT. ▌

Remark 9 *As a simple observation,* GREEDY CUT *is equivalent to* reverse-Kruskal *in 1D; It starts from all edges, goes over them in increasing order of weights (here, in decreasing order of lengths), and only keeps an edge when its removal makes the graph disconnected. We claim the first edge picked by reverse-kurskal corresponds to the interval $[x_m, x_{m+1}]$ with the maximum length, and hence the equivalence between the two algorithms by induction. To prove the claim, the line between x_1 and x_n keeps the graph connected, so the first picked edge corresponds to an interval (x_i, x_{i+1}). Also, it should be the interval $[x_m, x_{m+1}]$ with maximum length. Moreover, removing this edge makes the graph disconnected, because otherwise there is was cross edge between the left-side $(x_1, ..., x_m)$ and the right-side $(x_{m+1}, \ldots, x_n)$. The length of such an edge is more than the length of (x_m, x_{m+1}), so it should have been removed before, a contradiction.*

We finish the section by demonstrating the lack of performance of single-linkage (and hence GREEDY CUT) through an example, which justifies using both average-linkage and RANDOM CUT as *smooth* versions of single-linkage for 1D data points.

Lemma 5.3.2 *For $d = 1$, single-linkage can obtain at most $\frac{1}{2}$ of the optimum objective, and $\frac{1}{2}$ of the objective values of average-linkage and* RANDOM CUT*, under Gaussian kernels.*

Proof. Consider an example with n equally spaced points on a line, where the distance between any two adjacent node is Δ. We now slightly move the points so that weight of (x_i, x_{i+1}) is $(\sqrt{2\pi}\sigma)^{-n}(1-(i-1)\epsilon)e^{-\Delta^2/2\sigma^2}$ for $i = 1 : n-1$. Now, single-linkage peels off points in the order $x_1, \ldots, x_n$. Roughly speaking, we let Δ/σ^2 to be large enough so we can ignore the similarity weights between any two non-adjacent points. Hence single-linkage gets an objective value of $\approx \left(\sum_{i=1}^{n-1}(n-i)\right)(\sqrt{2\pi}\sigma)^{-n}e^{-\Delta^2/2\sigma^2}$, which evaluates to $\left(n^2/2 + o(n^2)\right)(\sqrt{2\pi}\sigma)^{-n}e^{-\Delta^2/2\sigma^2}$. Average-linkage returns the symmetric binary tree (and hence the objective value is

$$((n-2)(n-1) - n\log(n))\,(\sqrt{2\pi}\sigma)^{-n}e^{-\Delta^2/2\sigma^2} = \left(n^2 + o(n^2)\right)(\sqrt{2\pi}\sigma)^{-n}e^{-\Delta^2/2\sigma^2}$$

. Random returns a random binary tree, with (roughly speaking) a similar objective value of

$$\left(n^2 + o(n^2)\right)(\sqrt{2\pi}\sigma)^{-n}e^{-\Delta^2/2\sigma^2}$$

in expectation. The fact that optimum objective value is as large as these two quantities finishes the proof. ▌

5.4 A Fast Algorithm based on Random Projections

We now describe an algorithm Projected Random Cut which we use for high-dimensional data. This algorithm is given as Algorithm 9. It first projects on a random spherical Gaussian vector and then clusters the resulting projections using Random Cut.

Algorithm 9 Projected Random Cut

```
Input: Integer $n$, vectors $v_1, \dots, v_n \in \mathbb{R}^d$.
Output: Binary tree with leaves $(v_1, \dots, v_n)$

Pick a random Gaussian vector $\mathbf{g} \sim N_d(0,1)$
Compute dot products $x_i = \langle v_i, \mathbf{g} \rangle$
$x_{i_1}, \dots x_{i_n} = \textsc{Sort}(x_1, \dots, x_n)$
return Random Cut$(n, x_{i_1}, \dots, x_{i_n})$
```

Theorem 5.4.1 *For any input set of vectors $v_1, \dots, v_n \in \mathbb{R}^d$ the algorithm* Projected Random Cut *gives an α-approximation (in expectation) for the objective $\mathcal{F}^+$ under the Gaussian kernel similarity measure $w_{ij} \sim e^{-\|v_i - v_j\|_2^2/2\sigma^2}$ where $\alpha = (1+\delta)/3$ for $\delta = \min_{i,j} \exp(-\frac{\|v_i - v_j\|_2^2}{2\sigma^2})$.*

Proof. Recall an upper bound on the optimum:

$$OPT \le \texttt{MAX-upper} = \sum_{i<j<k} \max(w_{ij}, w_{ik}, w_{jk}).$$

Fix any triple (i,j,k) where $i < j < k$. Note that the objective value achieved by the algorithm Projected Random Cut can also be expressed as $ALG = \sum_{i<j<k} ALG_{i,j,k}$ where $ALG_{i,j,k}$ is the contribution to the objective from the triple (i,j,k) defined as follows. Consider the tree constructed by the algorithm. If v_k is the first vector in the triple (v_i, v_j, v_k) to be separated from the other two in the hierarchical partition (starting from the root) then $A_{i,j,k}$ is defined to be w_{ij} (in the other two cases when i or j are separated first the definition is analogous). Note that since Projected Random Cut is a randomized algorithm $ALG_{i,j,k}$ is a random variable. By the linearity of expectation we have $\mathbb{E}[ALG] = \sum_{i<j<k} \mathbb{E}[ALG_{i,j,k}]$. Thus in order to complete the proof it suffices to show that for every $i < j < k$ it holds that:

$$\mathbb{E}[ALG_{i,j,k}] \ge \alpha \max(w_{ij}, w_{ik}, w_{jk}).$$

Fix any triple (v_i, v_j, v_k) which forms a triangle in $\mathbb{R}^d$. Conditioned on cutting this triangle for the first time let (p_{ij}, p_{ik}, p_{jk}) be the vector of probabilities corresponding to the events that the corresponding edge is not cut. I.e. this is the probability that we score the contribution of this edge in the objective. Note that $p_{ij} + p_{ik} + p_{jk} = 1$.

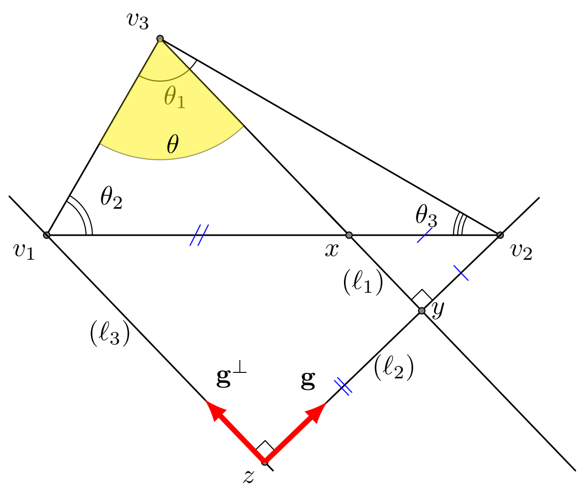

Figure 5.2: Projecting the triangle (v_1, v_2, v_3) on $\mathbf{g}$.

Consider any triangle whose vertices are v_i, v_j, v_k. To simplify presentation we set $i = 1, j = 2, k = 3$. We can assume that $\|v_2 - v_1\| \geq \|v_2 - v_3\| \geq \|v_1 - v_3\|$. Let $\theta_1 = \angle(v_1 - v_3, v_2 - v_3)$, $\theta_2 = \angle(v_2 - v_1, v_3 - v_1)$ and $\theta_3 = \angle(v_1 - v_2, v_3 - v_2)$ so that $\theta_1 \geq \theta_2 \geq \theta_3$. See Figure 5.2. Note that the probability that the i-th longest side of the triangle has the longest projection is then θ_i/π.

Lemma 5.4.1 *If (v_1, v_3) is the shortest edge in the triangle (v_1, v_2, v_3) then it holds that $p_{13} \geq \frac{1}{3}$*

Proof. Suppose that $\mathbf{g}$ forms an angle $\pi/2 - \theta$ with $(v_3 - v_1)$, i.e. the vector $\mathbf{g}^\perp$ orthogonal to $\mathbf{g}$ forms angle θ with $v_3 - v_1$. We define three auxiliary points x, y, z as follows (see also Figure 5.2.). Let ℓ_1 be a line parallel to $\mathbf{g}^\perp$ going through v_3, let ℓ_2 to be a line parallel to $\mathbf{g}$ going through v_2 and let ℓ_3 be the line parallel to ℓ_1 going through v_1. We then let x be the intersection of ℓ_1 and (v_1, v_2), y be the intersection of ℓ_1 and ℓ_2 and z be the intersection of ℓ_2 and ℓ_3 (see Fig 5.2).

Thus the projections of $v_3 - v_1$, $v_2 - v_1$ and $v_2 - v_3$ on $\mathbf{g}$ are $y - z$, $v_2 - z$ and $v_2 - y$. Hence conditioned on (v_1, v_2) having the longest projection the probability of scoring the contribution of (v_2, v_3) is given as $p_{23}^{12}(\theta) = \frac{\|y-z\|}{\|y-v_2\|}$ since we are applying the Random Cut algorithm after the projection. Note that by Thales's theorem we have $p_{23}^{12}(\theta) = \frac{\|y-z\|}{\|y-v_2\|} = \frac{\|x-v_1\|}{\|v_2-v_1\|}$. Applying the law of sines to the triangles (v_1, v_2, v_3) and (x, v_1, v_3) we have:

$$\frac{\sin\theta_1}{\|v_1 - v_2\|} = \frac{\sin(\theta_1 + \theta_2)}{\|v_1 - v_3\|}, \quad \frac{\sin\theta}{\|v_1 - v_3\|} = \frac{\sin(\theta + \theta_2)}{\|v_1 - x\|},$$

where we used the fact that $\sin\theta_3 = \sin(\pi - \theta_1 - \theta_2) = \sin(\theta_1 + \theta_2)$. Similarly, we use the fact that

$\sin(\angle(v_3 - x, v_1 - x)) = \sin(\theta + \theta_2)$. Using the above we can express $p_{23}^{12}(\theta)$ as:

$$p_{23}^{12}(\theta) = \frac{\sin\theta}{\sin(\theta+\theta_2)}\frac{\sin(\theta_1+\theta_2)}{\sin\theta_1}$$

Thus the overall probability of scoring the contribution of the edge (v_1, v_3) conditioned on (v_1, v_2) having the longest projection which we denote as p_{13}^{12} is given as:

$$p_{23}^{12} = \frac{1}{\theta_1}\int_0^{\theta_1} p_{23}^{12} d\theta$$

Similarly, consider the probability p_{13}^{12} of scoring the contribution of (v_1, v_3) conditioned on (v_1, v_2) having the longest projection. We can express it as:

$$p_{13}^{12} = \frac{1}{\theta_1}\int_0^{\theta_1} p_{13}^{12}(\theta) d\theta,$$

where

$$p_{13}^{12}(\theta) = \frac{\sin\theta}{\sin(\theta+\theta_3)}\frac{\sin(\theta_1+\theta_3)}{\sin\theta_1}.$$

Below we will show that $p_{13}^{12} \geq p_{23}^{12}$. In fact, we will show that for any fixed $\theta \in [0, \theta_1]$ it holds that $p_{13}^{12}(\theta) \geq p_{23}^{12}(\theta)$. Comparing the expressions for both it suffices to show that $\frac{\sin(\theta_1+\theta_3)}{\sin(\theta+\theta_3)} \geq \frac{\sin(\theta_1+\theta_2)}{\sin(\theta+\theta_2)}$ for all $\theta \in [0, \theta_1]$. Since $\theta_1 = \pi - \theta_2 - \theta_3$ this is equivalent to:

$$\frac{\sin\theta_3}{\sin(\theta+\theta_2)} \leq \frac{\sin\theta_2}{\sin(\theta+\theta_3)}$$

It suffices to show that:

$$\sin\theta_3 \sin(\theta+\theta_3) \leq \sin\theta_2 \sin(\theta+\theta_2)$$

Using the formula $\sin\alpha\sin\beta = \frac{1}{2}(\cos(\alpha-\beta) - \cos(\alpha+\beta))$ it suffices to show that:

$$\cos(\theta + 2\theta_3) \geq \cos(\theta + 2\theta_2).$$

The above inequality follows for all $\theta \in [0, \pi - \theta_2 - \theta_3]$ since $\theta_3 \leq \theta_2$. This shows that $p_{13}^{12} \geq p_{23}^{12}$. Since the probability that (v_1, v_2) has the longest projection is θ_1/π we have that the probability of scoring (v_1, v_3) and (v_1, v_2) having the longest projection is at least $\frac{\theta_1}{2\pi}$. An analogous argument shows that the probability of scoring (v_1, v_3) and (v_2, v_3) having the longest projection is at least $\frac{\theta_2}{2\pi}$.

Putting things together:

$$p_{13} \geq \frac{1}{2}\frac{\theta_1+\theta_2}{\pi} = \frac{1}{2}\frac{\pi-\theta_3}{\pi} \geq \frac{1}{3},$$

where we used that $\theta_3 \leq \pi/3$ since $\theta_3 \leq \theta_2 \leq \theta_1$. ▮

We are now ready to complete the proof of Theorem 5.4.1. Let $\gamma = \frac{\frac{2}{3}\delta}{(1+\delta)}$ and note that $\gamma \geq \delta/3$ since $\delta \leq 1$. If $p_{13} \geq 1/3 + \gamma$ then the desired guarantee follows immediately. Otherwise, if $p_{13} \leq 1/3 + \gamma$ then we have:

$$\begin{aligned}\frac{\mathbb{E}[ALG_{1,2,3}]}{OPT_{1,2,3}} &= \frac{p_{13}w_{13} + p_{12}w_{12} + p_{23}w_{23}}{w_{13}} \\ &\geq \frac{1}{3} + \left(\frac{2}{3} - \gamma\right)\frac{w_{12}}{w_{13}} \geq \frac{1}{3} + \left(\frac{2}{3} - \gamma\right)\delta \\ &= \frac{1}{3} + \frac{2}{3}\frac{\delta}{1+\delta} \geq \frac{1+\delta}{3},\end{aligned}$$

where we used the fact that

$$\frac{w_{12}}{w_{13}} = e^{\frac{\|v_1-v_3\|_2^2 - \|v_1-v_2\|_2^2}{2\sigma^2}} \geq e^{\frac{-\|v_1-v_2\|_2^2}{2\sigma^2}} \geq \delta.$$

∎

5.4.1 Gaussian Kernels with small δ

Theorem 5.4.1 only provides an improved approximation guarantee for Projected Random Cut compared to the factor 1/3 (i.e., the tight approximation guarantees of average-linkage in high dimensions; see Section 5.5) if δ is not too small, where $\delta = \min_{i,j}\exp(-\frac{\|v_i-v_j\|_2^2}{2\sigma^2})$. In particular, we get constant improvement if $\delta = \Omega(1)$. Is this a reasonable assumption? Interestingly, we answer this question in the affirmative by showing that if we have $\delta = \exp(-\tilde{\Omega}(n))$, then the Gaussian kernel can encode *arbitrary* similarity weights (up to scaling, which has no effects on multiplicative approximations). For simplicity, we only prove this result for $\{\epsilon, 1\}$ weights here, while it can be generalized to arbitrary weights.

Theorem 5.4.2 *Given any undirected graph $G = (V, E)$ on n nodes and $\epsilon > 0$, there exist unit vectors $\{k_v\}_{v\in V}$ in $\mathbb{R}^d$ and bandwidth parameter $\sigma \in \mathbb{R}_+$, such that $d = O(n^2)$, $\frac{1}{\sigma^2} = \Omega(n\log(1/\epsilon))$, and for some $\alpha > 0$ we have:*

$$\forall (u,v) \in E : e^{-\frac{\|k_u-k_v\|_2^2}{2\sigma^2}} = \alpha,$$
$$\forall (u,v) \notin E : e^{-\frac{\|k_u-k_v\|_2^2}{2\sigma^2}} = \alpha\epsilon.$$

Proof. Our proof is constructive. Let $d = \binom{n}{2}$. Pick orthogonal vectors $\{x_e\}_{e\in E}$ in $\mathbb{R}^d$ such that $\|x_e\|_2 = \frac{1}{d_u d_v}$, where d_u is the degree of node u in graph G. For each $v \in V$, define $y_v \in \mathbb{R}^d$ as follows:

$$y_v \triangleq \sum_{(u,v)\in E} (d_u d_v)x_{uv}.$$

Note that $\|y_v\|_2^2 = \sum_{(u,v)\in E} d_u^2 d_v^2 \frac{1}{d_u^2 d_v^2} = d_v$ As the next step, pick a set of n orthonormal vectors $\{z_v\}$ in the null space of $\{y_v\}_{v\in V}$. Finally, for each $v \in V$, define the final vector $k_v \in \mathbb{R}^d$ as follows:

$$k_v \triangleq \sqrt{1-\frac{d_v}{n}} z_v + \sqrt{\frac{1}{n}} y_v$$

First, note that these vectors have unit length:

$$\|k_v\|_2^2 = 1 - \frac{d_v}{n} + \frac{\|y_v\|_2^2}{n} = 1 - \frac{d_v}{n} + \frac{d_v}{n} = 1$$

Now, pick any two vertices u and v. If $(u,v) \notin E$, then:

$$\begin{aligned}\langle k_u, k_v\rangle &= \langle \sqrt{1-\frac{d_v}{n}} z_u + \sqrt{\frac{1}{n}} y_u, \sqrt{1-\frac{d_v}{n}} z_v + \sqrt{\frac{1}{n}} y_v\rangle \\ &= \frac{1}{n}\langle y_u, y_v\rangle = 0 \ ,\end{aligned}$$

where we used the fact that $\langle z_u, z_v\rangle = 0$, $\langle z_u, y_v\rangle = \langle z_v, y_u\rangle = 0$ and the fact that $\langle x_e, x_{e'}\rangle = 0$ for every edge e incident to u and every edge e' incident to v, as $e \neq e'$ when $(u,v) \notin E$. Similarly, when $(u,v) \in E$:

$$\langle k_u, k_v\rangle = \frac{1}{n}\langle y_u, y_v\rangle = \frac{1}{n}(d_u^2 d_v^2 \|x_{uv}\|_2^2) = \frac{1}{n}$$

Now, consider a Gaussian kernel with bandwidth $\sigma = (n\log(1/\epsilon))^{-\frac{1}{2}}$ and vectors $\{k_v\}_{v\in V}$. Since $\|k_u - k_v\|_2^2 = 2(1 - \langle k_u, k_v\rangle)$ from the above calculations of the inner products it follows that:

$$\begin{aligned}\forall (u,v) \in E &: e^{-\frac{\|k_u-k_v\|_2^2}{2\sigma^2}} = e^{\frac{1-1/n}{\sigma^2}}, \\ \forall (u,v) \notin E &: e^{-\frac{\|k_u-k_v\|_2^2}{2\sigma^2}} = e^{\frac{1}{\sigma^2}}.\end{aligned}$$

Thus the ratio is $e^{-\frac{1}{n\sigma^2}} = \epsilon$, as desired. ▮

5.5 Hard Instances with Gaussian Kernel

High-dimensional case. We embed the construction of [35] shown in Figure 5.3 into vectors with similarities given by the Gaussian kernel.

Theorem 5.5.1 *There exists a set of vectors $v_1, \dots, v_n \in \mathbb{R}^d$ for $d = poly(\log n)$ for which the average-linkage clustering algorithm achieves an approximation at most $\frac{1}{3} + o(1)$ for $\mathcal{F}^+$ under the Gaussian kernel similarity measure.*

Proof. [Proof of Theorem 5.5] We start by this theorem:

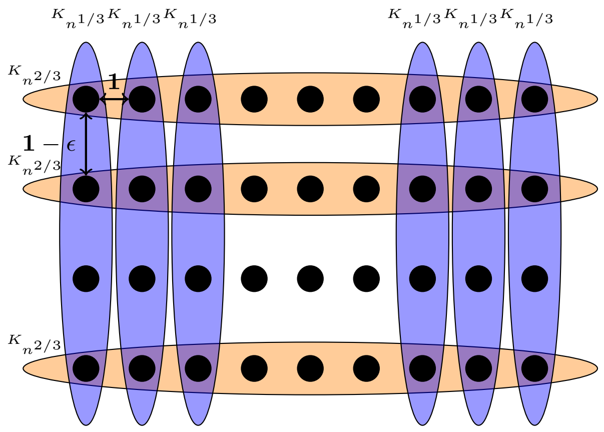

Figure 5.3: Hard instance $\mathcal{I}$ from [28]. Vertices in orange blocks form cliques of size $n^{2/3}$ connected by edges of similarity $1-\epsilon$, vertices in blue blocks form cliques of size $n^{1/3}$ connected by edges of similarity 1, all other pairs have similarity 0.

Theorem 5.5.2 ([35]) *For any constant $\epsilon \in (0,1)$ the instance $\mathcal{I}$ average-linkage clustering achieves the value of $\mathcal{F}^+$ at most $\frac{1}{6}n^{8/3} + O(n^{7/3})$ while the optimum is at least $\frac{1}{2}n^{8/3} - O(n^{7/3})$.*

Let $\Delta, \tau > 0$ be real-valued parameters to be chosen later. We use indices i and j to index our set of vectors. For $i \in \{1, 2, \ldots, n^{1/3}\}$, $j \in \{1, 2, \ldots, n^{2/3}\}$ let

$$v_{i,j} = \Delta(e_i + (1+\tau)e_{k+j}),$$

where $k = n^{1/3}$ and e_i is the t-th standard unit vector $e_t = (0\ldots, 1, \ldots, 0)$ with the 1 in the t-th entry. Then it is easy to see that for any fixed $i \in [n^{1/3}]$ and $j_1 \neq j_2 \in [n^{2/3}]$ it holds that:

$$\|v_{i,j_1} - v_{i,j_2}\|_2^2 = 2(1+\tau)^2\Delta^2$$

Similarly, for any fixed $j \in [n^{2/3}]$ and $i_1 \neq i_2 \in [n^{1/3}]$ it holds that:

$$\|v_{i_1,j} - v_{i_2,j}\|_2^2 = 2\Delta^2.$$

Otherwise if $i_1 \neq i_2 \in [n^{1/3}]$ and $j_1 \neq j_2 \in [n^{2/3}]$ then:

$$\|v_{i_1,j_1} - v_{i_2,j_2}\|_2^2 = 2\Delta^2 + 2(1+\tau)^2\Delta^2 \geq 4\Delta^2.$$

By setting $\Delta^2 = 2\sigma^2 c \log n$ for a sufficiently large constant c the contribution of pairs of vectors with $i_1 \neq i_2$ and $j_1 \neq j_2$ can be made negligible. Let $2(1+\tau)^2\Delta^2 = \alpha$ and $2\Delta^2 = \beta$. The rest of the pairs correspond to an hard instance $\mathcal{I}$ for which average-linkage only achieves a $\frac{1}{3} + o(1)$-approximation compared to the optimum. By setting $\tau = 1/poly(\log(n))$ we have $e^{(\beta^2-\alpha^2)/2\sigma^2} =$

$\Omega(1)$ and hence by Theorem 5.5.2 it follows that average-linkage clustering can't achieve better than $1/3 + o(1)$ approximation for this instance.

Finally, note that by applying the Johnson-Lindenstrauss transform we can reduce the dimension required for the above reduction to $d = poly(\log n)$. Indeed, projecting on a random subspace of dimension $O(\frac{\log n}{\xi^2})$ would preserve ℓ_2^2-distances between all pairs of vectors up to a multiplicative factor of $(1 \pm \xi)$ setting $\xi = \frac{\tau}{10} = 1/poly(\log n)$ it follows that our hard instance can be embedded in dimension $d = poly(\log n)$. ▮

Low-dimensional case. For $d = 1$ a hard instance for average-linkage clustering can also be constructed.

Lemma 5.5.1 *For points $x_1, \dots, x_n \in \mathbb{R}$ average-linkage clustering achieves approximation at most $3/4$ for $\mathcal{F}^+$ under the Gaussian kernel similarity measure.*

Proof. Consider the instance consisting of four equally spaced points on a line, i.e. $0, \Delta, 2\Delta, 3\Delta$. Then (after shifting the two middle points slightly) the average-linkage clustering algorithm might first connect the two middle points and then connect the two other points in arbitrary order. We denote the cost of this solution as AVG. An alternative solution would be to create two groups $(0, \Delta)$ and $(2\Delta, 3\Delta)$ and then merge them together. We denote the cost of this solution as OPT. By making Δ sufficiently large the contribution of pairs at distance more than Δ from each other can be ignored. We thus have $AVG \le (\sqrt{2\pi}\sigma)^{-n}(3e^{-\Delta^2/2\sigma^2} + o(e^{-\Delta^2/2\sigma^2}))$ and $OPT \ge (\sqrt{2\pi}\sigma)^{-n} 4e^{-\Delta^2/2\sigma^2}$, which gives the ratio of $3/4$ for sufficiently large Δ. ▮

Corollary 5.5.3 *In the four-point instance of the proof of Theorem 5.5.1, the `1D-SUM-upper` evaluates to*
$(\sqrt{2\pi}\sigma)^{-n} 6e^{-\Delta^2/2\sigma^2}$, and hence average-linkage cannot obtain more than $1/2$ of `1D-SUM-upper`.

In this section we demonstrate the quality of the solution returned by Projected Random Cut (PRC) on a small dataset, and highlight that running time only scales linearly on a large dataset. PRC does not compute the similarity weights (i.e., it only takes one pass over the feature vectors with dimension-free memory requirements) and then sorts the projected points in time $O(n \log n)$. Hence, it is fast even in use-cases with over millions of datapoints (in contrast to average-linkage, spectral clustering, or even single-linkage which all have superlinear running times in high dimensions).

We run PRC algorithm on two real datasets from the UCI ML repository [71]: (i) the Zoo dataset [71, 96] (the *small* dataset) that contains 100 animals given as 16D feature vectors (this dataset comes from applications in biology), and (ii) the SIFT10M dataset [52] (the *large* dataset) that contains around 10M datapoints, where each datapoint is a 128D Scale Invariant Feature Transform (SIFT) vector (this dataset comes from applications in computer vision). For more details, refer to subsection 5.5.1.

σ	PRC	SPECTRAL	AL	`MAX-upper`	$\frac{\text{PRC}}{\texttt{MAX-upper}}$
1.5	48	61	28	64	0.75
2	64	83	47	87	0.74
2.5	83	100	66	105	0.79
3	100	112	82	117	0.85
3.5	111	121	95	126	0.87
4	117	128	105	132	0.88
4.5	123	133	114	137	0.91
5	129	137	120	140	0.92

Table 5.1: Values of the objective (times 10^{-3}) on the Zoo dataset (averaged over 10 runs).

Small dataset (Zoo). We compare PRC to (i) the recursive spectral clustering using the second eigenvector of the normalized Laplacian of the weight matrix (SPECTRAL) [35], (ii) average-linkage (AL), and (iii) `MAX-upper`, an upper-bound on the objective value of the optimum tree; see Chapter 3. In contrast to PRC, these three benchmarks need to compute the weights and are slow on large datasets.

Table 5.1 summarizes the results of this experiment for various choices of the parameter σ (first column). The second, third and fourth columns report the objective values (i.e., $\mathcal{F}^+$) of the trees returned by PRC, SPECTRAL and AL respectively, and the fifth column gives an empirically observed approximation guarantee for PRC by comparing it against the upper bound on optimum `MAX-upper`. We observe that PRC attains high approximation factors of $\approx [0.73, \ldots, 0.92]$ compared to `MAX-upper` . Moreover, PRC obtains competitive objective values compared to SPECTRAL and AL, although it runs faster (in almost linear-time).

On the choice of bandwidth parameter σ, as a rule of thumb, we find an interval $[\alpha, \beta]$ such that (i) if $\sigma > \beta$, a considerable portion of pairs of datapoints with very different weights under the cosine similarities [90] have almost equal weights under the Gaussian kernel, and (ii) if $\sigma < \alpha$, a considerable portion of pairs of datapoints with almost equal weights under the cosine similarities have very different weights under the Gaussian kernel. We end up with $\alpha = 1.5, \beta = 5$.

Large dataset (SIFT10M) The focus of this experiment is measuring the running time of PRC, and showing that it only scales linearly with the dataset size. Note that evaluating the performance of any other algorithm or upper bound (or even one pass over the similarity matrix) would be prohibitive.

We run PRC on truncated versions of SIFT10M of sizes 10K, 100K, 500K, 1M and 10M. σ is set to 450 [2]. We emphasize that our PRC algorithm runs extremely fast on a 2014 MacBook[3]. The running times are summarized in Table 5.2. Observe that PRC scales almost linearly with the data and has almost the same running time as just a single pass over the datapoints.

Size	PRC (seconds)	1 Data Pass (seconds)
10K	1.7	1.5
100K	13	9.6
500K	67	46.4
1M	135	99.7
10M	1592	1144

Table 5.2: Running times of PRC and one pass.

5.5.1 Datasets from Section 5.5

We used real data in our experiments. To be able to measure the performance of produced Hierarchical Clusterings, we need to compute `MAX-upper`, and hence we use a small dataset. To be able to show linear scaling of running time even for tens of millions of datapoints, we will use large datasets of sizes 10K to 10M.

The small dataset: The Zoo dataset contains 100 animals given as 16-dimensional vectors, forming 7 different classes (e.g., mammals, amphibians, etc.). The features contain information about the animal's characteristics (e.g., if it has tail, hair, fins, etc.). Here, we want to showcase the quality of the solution produced by PRC; in the case of the small dataset, we can afford keeping track of two benchmarks: the performance of the widely-used SPECTRAL algorithm and the `MAX-upper`[4].

The large dataset: In the SIFT10M dataset, each data point is a SIFT feature, extracted from Caltech-256 [56] by the open source VLFeat library [95]. Caltech-256 is used as a computer vision benchmark image data set, that features a total of 256 different classes with high intra-class variations for each category. Each datapoint is a 128-dimensional vector and similar to the Zoo dataset, we use this information as features to perform fast Hierarchical Clustering. Here, we want to test the scalability of our algorithm so we use successively larger datasets from the SIFT10M dataset of 10K, 100K, 500K, 1M and finally 10M datapoints by truncating the data file as necessary.

Chapter 6

Hierarchical Clustering with Structural Constraints

By 'life' we mean a thing that can nourish itself and grow and decay.

-Aristotle

This chapter is devoted to the case where both quantitative and qualitative information is available to our algorithms. The former takes the form of similarity weights as we previously had, whereas the latter come as *structural constraints* to be followed by the output hierarchy. For many real-world applications, we would like to exploit prior information about the data that imposes constraints on the clustering hierarchy, and is not captured by the set of features available to the algorithm.

A simple example of such constraints is a *triplet* constraint (or "must-link-before") constraints $ab|c$: these are the analogue of "must-link/cannot-link" from standard clustering. Notice that hierarchies on data imply that all datapoints are linked at the highest level and all are separated at the lowest level, hence "cannot-link" and "must-link" constraints are not directly meaningful. A triplet $ab|c$ requires that valid solutions contain a sub-cluster with a, b together and c previously separated from them. For a concrete example from taxonomy of species, a triplet constraint may look like $(\textsc{Tuna}, \textsc{Salmon}|\textsc{Lion})$.

Structural constraints pose major challenges for bottom-up approaches like average/single linkage and even though they can be naturally incorporated into top-down divisive algorithms, no formal guarantees exist on the quality of their output as measured by Dasgupta's cost and its variants. In this chapter, we provide provable approximation guarantees for two simple top-down algorithms. We show how to find good solutions even in the presence of conflicting prior information, by formulating a constraint-based regularization of the objective. Finally, we demonstrate our approach on a real dataset for the taxonomy application.

6.1 Motivation and Chapter Overview

Hierarchical clustering was originally motivated by gene expression data analysis [46] and applications in bioinformatics [45]. In such applications, the user (i.e., biologist) often has *background knowledge* about the data that may not be captured by the input to the clustering algorithm which is just automatically generated pairwise similarities. This background knowledge comes in the form of constraints on the hierarchy to be found. There is a rich body of work on constrained flat clustering formulations that take into account such user input in the form of "cannot link" and "must link" constraints [97, 98, 20]. Very recently, "semi-supervised" versions of HC that incorporate additional constraints have been studied [96], where the natural form of such constraints is triplet (or "must link before") constraints $ab|c$. Such triplet constraints, as we formally show later, can encode more general structural constraints in the form of rooted subtrees. Surprisingly, such simple triplet constraints already pose significant challenges for bottom-up linkage methods. (Figure 6.1).

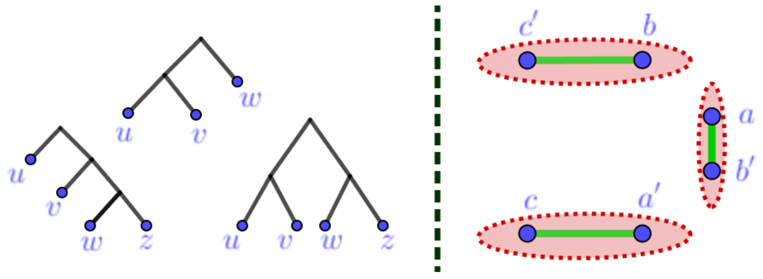

Figure 6.1: *(Left)* Example of a triplet constraint $uv|w$ and more general rooted tree constraints on 4 points u, v, w, z. *(Right)* Example with only two constraints $ab|c, a'b'|c'$ demonstrating that popular distance-based linkage algorithms may fail to produce valid HC. Here they get stuck after 3 merging steps (green edges).

Our work in this chapter is motivated by applying the optimization lens to study the interaction of hierarchical clustering algorithms with structural constraints. Constraints can be fairly naturally incorporated into top-down (i.e. divisive) algorithms for hierarchical clustering; but can we establish guarantees on the quality of the solution they produce? Another issue is that incorporating constraints from multiple experts may lead to a conflicting set of constraints; can the optimization viewpoint of hierarchical clustering still help us obtain good solutions even in the presence of infeasible constraints? Finally, different objective functions for HC have been studied in the literature; do algorithms designed for these objectives behave similarly in the presence of constraints? To the best of our knowledge, the work presented in this chapter is the first to propose a unified approach for constrained HC through the lens of optimization and to give provable approximation guarantees for a collection of fast and simple top-down algorithms that have been used for unconstrained HC in practice (e.g. community detection in social networks [75]).

Two HC Objectives. Recall Dasgupta's objective from previous chapters, whose goal is to find a tree T^* such that:

$$T^* = \underset{\text{all trees } T}{\arg\min} \sum_{(i,j)\in E} w_{ij} \cdot |T_{ij}| \tag{6.1}$$

We denote (6.1) as *similarity-HC*. For applications where the geometry of the data is given by dissimilarities, again denoted by $\{w_{ij}\}_{(i,j)\in E}$, recall Cohen-Addad et al. objective [39]:

$$T^* = \underset{\text{all trees } T}{\arg\max} \sum_{(i,j)\in E} w_{ij} \cdot |T_{ij}| \tag{6.2}$$

We denote (6.2) as *dissimilarity-HC*. A comprehensive list of desirable properties of the aforementioned objectives can be found in previous chapters.

Our Results. *i)* We design algorithms that take into account both the geometry of the data, in the form of similarities, and the structural constraints imposed by the users. Our algorithms emerge as the natural extensions of Dasgupta's original recursive sparsest cut algorithm and the recursive balanced cut suggested in [27]. We generalize previous analyses to handle constraints and we prove an $O(k\alpha_n)$-approximation guarantee, thus surprisingly matching the best approximation guarantee of the *unconstrained* HC problem for constantly many constraints.

ii) In the case of infeasible constraints, we extend the similarity-HC optimization framework, and we measure the quality of a possible tree T by a *constraint-based regularized* objective. The regularization naturally favors solutions with as few constraint violations as possible and as far down the tree as possible (similar to the motivation behind the similarity-HC objective). For this problem, we provide a top-down $O(k\alpha_n)$-approximation algorithm by drawing an interesting connection to an instance of the hypergraph sparsest cut problem.

iii) We then change gears and study the dissimilarity-HC objective. Surprisingly, we show that known top-down techniques do not cope well with constraints, drawing a contrast with the situation for similarity-HC. Specifically, the *(locally) densest cut heuristic* suggested in [39] performs poorly even if there is only one triplet constraint, blowing up its approximation factor to $O(n)$. Moreover, we improve upon the state-of-the-art in [39], by showing a simple randomized partitioning is a $\frac{2}{3}$-approximation algorithm. We also give a deterministic *local-search* algorithm with the same worst-case guarantee. Furthermore, we show that our randomized algorithm is robust under constraints, mainly because of its "exploration" behavior. In fact, besides the number of constraints, we propose an inherent notion of *dependency measure* among constraints to capture this behavior quantitatively. This helps us not only to explain why "non-exploring" algorithms may perform poorly, but also gives tight guarantees for our randomized algorithm.

Experimental Results. We run simple experiments on the Zoo dataset [71] to demonstrate our approach and the performance of our algorithms for a taxonomy application. We consider a setup where there is a ground-truth tree and extra information regarding this tree is provided for the algorithm in the form of triplet constraints. The upshot is that specific variations of our algorithms can exploit this information. In this practical application, our algorithms have around %9 improvements in the objective compared to the naive recursive sparsest cut that does not use this information. See Appendix A.2 for more details on the setup and precise conclusions of our experiments.

Constrained HC work-flow in Practice. Throughout the chapter, we develop different tools to handle user-defined structural constraints for hierarchical clustering. Here we describe a recipe on how to use our framework in practice.

(1) Preprocessing constraints to form triplets. User-defined structural constraints as rooted binary subtrees are convenient for the user and hence for the usability of our algorithm. The following proposition (whose proof is in the Appendix 6.2.1) allows us to focus on studying HC with just triplet constraints.

Proposition 6.1.1 *Given constraints as a rooted binary subtree T on k data points ($k \geq 3$), there is linear time algorithm that returns an equivalent set of at most k triplet constraints.*

(2) Detecting feasibility. The next step is to see if the set of triplet constraints is consistent, i.e., whether there exists a HC satisfying all the constraints. For this, we use a simple linear time algorithm known as `BUILD` [3].

(3) Hard constraints vs. regularization. `BUILD` can create a hierarchical decomposition that satisfies triplet constraints, but ignores the *geometry* of the data, whereas our goal here is to consider both simultaneously. Moreover, in the case that the constraints are infeasible, we aim to output a clustering that minimizes the cost of violating constraints combined with the cost of the clustering itself.

• *Feasible instance:* To output a feasible HC, we propose using *Constrained Recursive Sparsest Cut* (`CRSC`) or *Constrained Recursive Balanced Cut* (`CRBC`): two simple top-down algorithms which are natural adaptations of recursive sparsest cut [75, 43] or recursive balanced cut [27] to respect constraints (Section 6.2.1).

• *Infeasible instance:* In this case, we turn our attention to a regularized version of HC, where the cost of violating constraints is added to the tree cost. We then propose an adaptation of `CRSC`, namely *Hypergraph Recursive Sparsest Cut* (`HRSC`) for the regularized problem (Section 6.2.2).

Real-world application example. In phylogenetics, which is the study of the evolutionary history and relationships among species, an end-user usually has access to whole genomes data of a group of organisms. There are established methods in phylogeny to infer similarity scores between pairs of datapoints, which give the user the similarity weights w_{ij}. Often the user also has access to rare structural footprints of a common ancestry tree (e.g. through gene rearrangement data, gene inversions/transpositions etc.). These rare, yet informative, footprints play the role of the structural constraints. The user can follow our pre-processing step to get triplet constraints from the given rare footprints, and then use Aho's BUILD algorithm to choose between regularized or hard version of the HC problem. The above illustrates how to use our workflow and why using our algorithms facilitates HC when expert domain knowledge is available.

Further related work. Similar to [96], constraints in the form of triplet queries have been used in an (adaptive) active learning framework by [92, 47], showing that approximately $O(n \log n)$ triplet queries are enough to learn an underlying HC. Other forms of user interaction in order to improve the quality of the produced clusterings have been used in [13] where they prove that interactive feedback in the form of cluster split/merge requests can lead to significant improvements. Robust algorithms for HC in the presence of noise were studied in [16, 12] and a variety of sufficient conditions on the similarity function that would allow linkage-style methods to produce good clusters was explored in [14]. On a different setting, the notion of triplets has been used as a measure of *distance* between hierarchical decomposition trees on the same data points [22]. More technically distant analogs of how to use relations among triplets points have recently been proposed in [66] for defining kernel functions corresponding to high-dimensional embeddings.

6.2 Minimizing Dasgupta's Objective with Triplets

6.2.1 Modified Sparsest or Balanced Cut Analysis

Given an instance of the constrained hierarchical clustering, our proposed `CRSC` algorithm uses a blackbox α_n-approximation algorithm for the sparsest cut problem (the best-known approximation factor for this problem is $O(\sqrt{\log n})$ due to [8]). Moreover, it also maintains the feasibility of the solution in a top-down approach by recursive partitioning of what we call the *supergraph* G'. Informally speaking, the supergraph is a simple data structure to track the progress of the algorithm and the resolved constraints.

More formally, for every constraint $ab|c$ we merge the nodes a and b into a *supernode* $\{a, b\}$ while maintaining the edges in G (now connecting to their corresponding supernodes). Note that G' may have parallel edges, but this can easily be handled by grouping edges together and replacing them with the sum of their weights. We repeatedly continue this *merging* procedure until there are no more constraints. Observe that any feasible solution needs to start splitting the original graph G by using a cut that is also present in G'. When cutting the graph $G' = (G_1, G_2)$, if a constraint $ab|c$ is *resolved*,[1] then we can safely *unpack* the supernode $\{a, b\}$ into two nodes again (unless there is another constraint $ab|c'$ in which case we should keep the supernode). By continuing and recursively finding approximate sparsest cuts on the supergraph G_1 and G_2, we can find a feasible hierarchical decomposition of G respecting all triplet constraints. Next, we show the approximation guarantees for our algorithm.

Algorithm 10 `CRSC`

```
1: Given $G$ and the triplet constraints $ab|c$, run BUILD to create the supergraph $G'$.
2: Use a blackbox access to an $\alpha_n$-approximation oracle for the sparsest cut problem, i.e.
   $\arg\min_{S\subseteq V} \frac{w_{G'}(S,\bar{S})}{|S|\cdot|\bar{S}|}$.
3: Given the output cut $(S,\bar{S})$, separate the graph $G'$ into two pieces $G_1(S,E_1)$ and $G_2(V\setminus S,E_2)$.
4: Recursively compute a HC $T_1$ for $G_1$ using only $G_1$'s active constraints. Similarly compute $T_2$
   for $G_2$.
5: Output $T=(T_1,T_2)$.
```

Analysis of `CRSC` Algorithm. The main result of this section is the following theorem:

Theorem 6.2.1 *Given a weighted graph $G(V,E,w)$ with k triplet constraints $ab|c$ for $a,b,c\in V$, the `CRSC` algorithm outputs a HC respecting all triplet constraints and achieves an $O(k\alpha_n)$-approximation for the HC-similarity objective as in (6.1).*

Notations and Definitions. We slightly abuse notation by having `OPT` denote the optimum hierarchical decomposition or its optimum value as measured by (6.1). Similarly for `CRSC`. For $t\in[n]$, $\texttt{OPT}(t)$ denotes the maximal clusters in `OPT` of size at most t. Note that $\texttt{OPT}(t)$ induces a partitioning of V. We use $\texttt{OPT}(t)$ to denote edges cut by $\texttt{OPT}(t)$ (i.e. edges with endpoints in different clusters in $\texttt{OPT}(t)$) or their total weight; the meaning will be clear from context. For convenience, we define $\texttt{OPT}(0)=\sum_{(i,j)\in E} w_{ij}$. For a cluster A created by `CRSC`, a constraint $ab|c$ is *active* if $a,b,c\in A$, otherwise $ab|c$ is resolved and can be discarded.

Overview of the Analysis. There are three main ingredients: The first is to view a HC of n datapoints as a collection of partitions, one for each level $t=n-1,\dots,1$, as in [27]. For a level t, the partition consists of maximal clusters of size at most t. The total cost incurred by `OPT` is then a combination of costs incurred at each level of this partition. This is useful for comparing our `CRSC` cost with `OPT`. The second idea is in handling constraints and it is the main obstacle where previous analyses [27, 39] break down: constraints inevitably limit the possible cuts that are feasible at any level, and since the set of active constraints[2] differ for `CRSC` and `OPT`, a direct comparison between them is impossible. If we have no constraints, we can charge the cost of partitioning a cluster A to lower levels of the `OPT` decomposition. However, when we have triplet constraints, the partition induced by the lower levels of `OPT` in a cluster A *will not be feasible in general* (Figure 6.2). The natural way to overcome this obstacle is merging pieces of this partition so as to respect constraints and using higher levels of `OPT`, but it still may be impossible to compare `CRSC` with `OPT` if all pieces are merged. We overcome this difficulty by an indirect comparison between the CRSC cost and lower levels $\frac{r}{6k_A}$ of OPT, where k_A is the number of active constraints in A. Finally, after a cluster-by-cluster analysis bounding the CRSC cost for each cluster, we exploit disjointness of clusters of the same level in the CRSC partition allowing us to combine their costs. *Proof.* [Proof of Theorem 6.2.1] We start by borrowing the following facts from [28], modified slightly for the purpose of our analysis (proofs are provided in the supplementary materials).

Fact 6.2.2 (Decomposition of OPT) *The total cost paid by OPT can be decomposed into costs of the different levels in the OPT partition, i.e.* $OPT = \sum_{t=0}^{n} w(OPT(t))$.

Fact 6.2.3 (OPT at scaled levels) *Let* $k \leq \frac{n}{6}$ *be the number of constraints. Then,* $OPT \geq \frac{1}{6k} \cdot \sum_{t=0}^{n} w(OPT(\lfloor \frac{t}{6k} \rfloor))$.

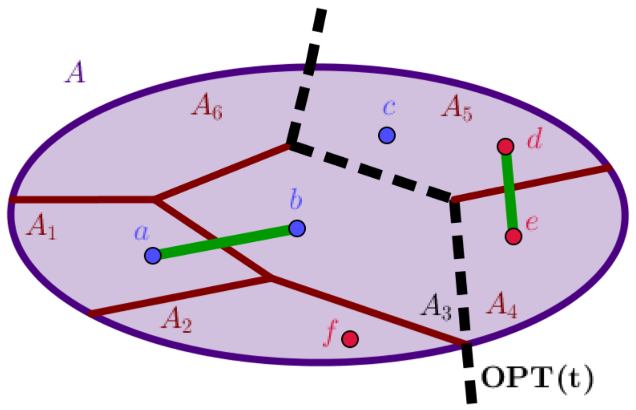

Figure 6.2: The main obstacle in the constrained HC problem is that our algorithm has different *active constraints* compared to OPT. Both $ab|c, de|f$ constraints are resolved by the cut OPT(t).

Given the above facts, we look at any cluster A of size r produced by the algorithm. Here is the main technical lemma that allows us to bound the cost of CRSC for partitioning A.

Lemma 6.2.1 *Suppose CRSC partitions a cluster A ($|A| = r$) in two clusters (B_1, B_2) (w.l.o.g. $|B_1| = s, |B_2| = r - s, s \leq \lfloor \frac{r}{2} \rfloor \leq r - s$). Let the size $r \geq 6k$ and let $l = 6k_A$, where k_A denotes the number of active constraints for A. Then:* $r \cdot w(B_1, B_2) \leq 4\alpha_n \cdot s \cdot w(OPT(\lfloor \frac{r}{l} \rfloor) \cap A)$.

Proof. The cost incurred by CRSC for partitioning A is $r \cdot w(B_1, B_2)$. Now consider OPT($\lfloor \frac{r}{l} \rfloor$). This induces a partitioning of A into pieces $\{A_i\}_{i \in [m]}$, where by design $|A_i| = \gamma_i |A|$, $\gamma_i \leq \frac{1}{l}, \forall i \in [m]$. Now, consider the cuts $\{(A_i, A \setminus A_i)\}$. Even though all m cuts are allowed for OPT, for CRSC some of them are forbidden: for example, in Figure 6.2, the constraints $ab|c, de|f$ render 4 out of the 6 cuts infeasible. But how many of them can become infeasible with k_A active constraints? Since every constraint is involved in at most 2 cuts, we may have at most $2k_A$ infeasible cuts. Let $F \subseteq [m]$

denote the index set of feasible cuts, i.e. if $i \in F$, the cut $(A_i, A \setminus A_i)$ is feasible for `CRSC`. To cut A, we use an α_n-approximation of sparsest cut, whose sparsity is upper bounded by *any feasible* cut:

$$\frac{w(B_1,B_2)}{s(r-s)} \leq \alpha_n \cdot \text{SP.CUT}(A) \leq \alpha_n \min_{i\in F} \frac{w(A_i, A\setminus A_i)}{|A_i||A\setminus A_i|} \leq \alpha_n \frac{\sum_{i\in F} w(A_i, A\setminus A_i)}{\sum_{i\in F}|A_i||A\setminus A_i|}$$

where for the last inequality we used the standard fact that $\min_i \frac{\mu_i}{\nu_i} \leq \frac{\sum_i \mu_i}{\sum_i \nu_i}$ for $\mu_i \geq 0$ and $\nu_i > 0$. We also have the following series of inequalities:

$$\alpha_n \frac{\sum_{i\in F} w(A_i, A\setminus A_i)}{\sum_{i\in F}|A_i||A\setminus A_i|} \leq \alpha_n \frac{2w(\texttt{OPT}(\lfloor\frac{r}{l}\rfloor)\cap A)}{r^2\sum_{i\in F}\gamma_i(1-\gamma_i)} \leq 4\alpha_n \frac{w(\texttt{OPT}(\lfloor\frac{r}{l}\rfloor)\cap A)}{r^2}$$

where the first inequality holds because we double count some (potentially all) edges of $\texttt{OPT}(\lfloor\frac{r}{l}\rfloor)\cap A$ (these are the edges cut by $\texttt{OPT}(\lfloor\frac{r}{l}\rfloor)$ that are also present in cluster A, i.e. they have both endpoints in A) and the second inequality holds because $\gamma_i \leq \frac{1}{6k} \implies 1-\gamma_i \geq \frac{6k-1}{6k}$ and

$$\sum_{i\in F}\gamma_i(1-\gamma_i) \geq \sum_{i=1}^{m}\gamma_i(1-\gamma_i) - 2\sum_{i\in[m]\setminus F}\frac{1}{6k} \geq \frac{6k-1}{6k}\sum_{i=1}^{m}\gamma_i - \frac{2k}{6k} = \frac{4k-1}{6k} \geq 1/2$$

Finally, we are ready to prove the lemma by combining the above inequalities ($\frac{r-s}{r} \leq 1$):

$$\begin{aligned} r\cdot w(B_1,B_2) &= r\cdot s(r-s)\cdot\frac{w(B_1,B_2)}{s(r-s)} \\ &\leq r\cdot s(r-s)\cdot 4\alpha_n\frac{w(\texttt{OPT}(\lfloor\frac{r}{l}\rfloor)\cap A)}{r^2} \leq 4\alpha_n\cdot s\cdot w(\texttt{OPT}(\lfloor\frac{r}{l}\rfloor)\cap A). \end{aligned}$$

This finishes the lemma. ▮

It is clear that we exploited the charging to lower levels of `OPT`, since otherwise if all pieces in A were merged, the denominator with the $|A_i|$'s would become 0. The next lemma lets us combine the costs incurred by `CRSC` for different clusters A (proof is in the supplementary materials)

Lemma 6.2.2 (Combining the costs of clusters in `CRSC`) *The total CRSC cost for partitioning all clusters A into (B_1, B_2) (with $|A| = r_A, |B_1| = s_A$) is bounded by:*

$$(1) \quad \sum_{A:|A|\geq 6k} r_A\cdot w(B_1,B_2) \leq O(\alpha_n)\cdot\sum_{t=0}^{n} w(\mathit{OPT}(\lfloor\tfrac{t}{6k}\rfloor))$$

$$(2) \quad \sum_{A:|A|<6k} r_A w(B_1,B_2) \leq 6k\cdot \mathit{OPT}$$

Combining Fact 6.2.3 and Lemma 6.2.2 finishes the proof. ▮

Remark 10 *In the supplementary material, we prove how one can use* balanced cut, *i.e. finding a* *cut S such that*

$$\underset{S \subseteq V: |S| \geq n/3, |\bar{S}| \geq n/3}{\arg\min} \quad w_{G'}(S, \bar{S}) \tag{6.3}$$

instead of sparsest cut, and using approximation algorithms for this problem achieves the same approximation factor as in Theorem 6.2.1, but with better running time.

Remark 11 *Optimality of the CRSC algorithm: Note that complexity theoretic lower-bounds for the unconstrained version of HC from [27] also apply to our setting; more specifically, they show that no constant factor approximation exists for HC assuming the Small-Set Expansion Hypothesis.*

Theorem 6.2.4 (The divisive algorithm using balanced cut) *Given a weighted graph $G(V, E, w)$ with k triplet constraints $ab|c$ for $a, b, c \in V$, the constrained recursive balanced cut algorithm* `CRBC` *(same as* `CRSC`*, but using balanced cut instead of sparsest cut) outputs a HC respecting all triplet constraints and achieves an $O(k\alpha_n)$-approximation for Dasgupta's HC objective. Moreover, the running time is almost linear time.*

6.2.2 Soft Triplets and Regularization

Previously, we assumed that constraints were feasible. However, in many practical applications, users/experts may disagree, hence our algorithm may receive conflicting constraints as input. Here we want to explore how to still output a satisfying HC that is a good in terms of objective (6.1) (similarity-HC) and also respects the constraints as much as possible. To this end, we propose a *regularized* version of Dasgupta's objective, where the regularizer measures quantitatively the degree by which constraints get violated.

Informally, the idea is to penalize a constraint more if it is violated at top levels of the decomposition compared to lower levels. We also allow having different violation *weights* for different constraints (potentially depending on the expertise of the users providing the constraints). More concretely, inspired by the Dasgupta's original objective function, we consider the following optimization problem:

$$\min_{T \in \mathcal{T}} \left(\sum_{(i,j) \in E} w_{ij} |T_{ij}| + \lambda \cdot \sum_{ab|c \in \mathcal{K}} c_{ab|c} |T_{ab}| \cdot \mathbf{1}\{ab|c \text{ is violated}\} \right), \tag{6.4}$$

where $\mathcal{T}$ is the set of all possible binary HC trees for the given data points, $\mathcal{K}$ is the set of the k triplet constraints, T_{ab} is the size of the subtree rooted at the least common ancestor of a, b, and $c_{ab|c}$ is defined as the *base cost* of violating triplet constraint $ab|c$. Note that the regularization parameter $\lambda \geq 0$ allows us to interpolate between satisfying the constraints or respecting the geometry of the data.

Hypergraph Recursive Sparsest Cut In order to design approximation algorithms for the regularized objective, we draw an interesting connection to a different problem, which we call *3-Hypergraph Hierarchical Clustering (3HHC)*. An instance of this problem consists of a hypergraph $G^{\mathcal{H}} = (V, E, E^{\mathcal{H}})$ with edges E, and hyperedges of size 3, $E^{\mathcal{H}}$, together with similarity weights for edges, $\{w_{ij}\}_{(i,j)\in E}$, and similarity weights for 3-hyperedges,[3] $\{w_{ij|k}\}_{(i,j,k)\in E^{\mathcal{H}}}$. We now think of HC on the hypergraph $G^{\mathcal{H}}$, where for every binary tree T we define the cost to be the natural extension of Dasgupta's objective:

$$\sum_{(i,j)\in E} w_{ij}|T_{ij}| + \sum_{(i,j,k)\in E^{\mathcal{H}}} w^{T}_{ijk}|T_{ijk}| \tag{6.5}$$

where w^T_{ijk} is either equal to $w_{ij|k}, w_{jk|i}$ or $w_{ki|j}$, and T_{ijk} is either the subtree rooted at $\mathrm{LCA}(i,j)$,[4] $\mathrm{LCA}(i,k)$ or $\mathrm{LCA}(k,j)$, all depending on how T cuts the 3-hyperedge $\{i,j,k\}$. The goal is to find a hierarchical clustering of this hypergraph, so as to minimize the cost (6.5) of the tree.

Reduction from Regularization to 3HHC. Given an instance of HC with constraints (with their costs of violations) and a parameter λ, we create a hypergraph $G^{\mathcal{H}}$ so that the total cost of any binary clustering tree in the 3HHC problem (6.5) corresponds to the regularized objective of the same tree as in (6.4). $G^{\mathcal{H}}$ has exactly the same set of vertices, (normal) edges and (normal) edge weights as in the original instance of the HC problem. Moreover, for every constraint $ab|c$ (with cost $c_{ab|c}$) it has a hyperedge $\{a,b,c\}$, to which we assign three weights $w_{ab|c} = 0, w_{ac|b} = w_{bc|a} = \lambda \cdot c_{ab|c}$. Therefore, we ensure that any divisive algorithm for the 3HHC problem avoids the cost $|T_{abc}| \cdot \lambda \cdot c_{ab|c}$ only if it chops $\{a,b,c\}$ into $\{a,b\}$ and $\{c\}$ at some level, which matches the regularized objective.

Reduction from 3HHC to Hypergraph Sparsest Cut. A natural generalization of the sparsest cut problem for our hypergraphs, which we call *Hyper Sparsest Cut (HSC)*, is the following problem:

$$\arg\min_{S\subseteq V} \left(\frac{w(S,\bar{S}) + \sum_{(i,j,k)\in E^{\mathcal{H}}} w^S_{ijk}}{|S||\bar{S}|} \right) ,$$

where $w(S,\bar{S})$ is the weight of the cut $(S,\bar{S})$ and w^S_{ijk} is either equal to $w_{ij|k}, w_{jk|i}$ or $w_{ki|j}$, depending on how $(S,\bar{S})$ chops the hyperedge $\{i,j,k\}$. Now, similar to [27, 43], we can recursively run a blackbox approximation algorithm for HSC to solve 3HHC. The main result of this section is the following technical proposition, whose proof is analogous to that of Theorem 6.2.1 (provided in the supplementary materials).

Proposition 6.2.5 *Given the hypergraph $G^{\mathcal{H}}$ with k hyperedges, and given access to an algorithm which is α_n-approximation for HSC, the Recursive Hypergraph Sparsest Cut (`R-HSC`) algorithm achieves an $O(k\alpha_n)$-approximation.*

Reduction from HSC back to Sparsest Cut. We now show how to get an α_n-approximation oracle for our instance of the HSC problem by a general reduction to sparsest cut. Our reduction is simple: given a hypergraph $G^{\mathcal{H}}$ and all the weights, create an instance of sparsest cut with the same vertices, (normal) edges and (normal) edge weights. Moreover, for every 3-hyperedge $\{a, b, c\}$ consider adding a *triangle* to the graph, i.e. three weighted edges connecting $\{a, b, c\}$, where:

$$\begin{aligned} w'_{ab} &= \frac{w_{bc|a} + w_{ac|b} - w_{ab|c}}{2} = \lambda \cdot c_{ab|c}, \\ w'_{ac} &= \frac{w_{bc|a} + w_{ab|c} - w_{ac|b}}{2} = 0, \\ w'_{bc} &= \frac{w_{ac|b} + w_{ab|c} - w_{bc|a}}{2} = 0. \end{aligned}$$

This construction can be seen in Figure 6.3. The important observation is that $w'_{ab} + w'_{ac} = w_{bc|a}$, $w'_{ab} + w'_{bc} = w_{ac|b}$ and $w'_{bc} + w'_{ac} = w_{ab|c}$, which are exactly the weights associated with the corresponding splits of the 3-hyperedge $\{a, b, c\}$. So, correctness of the reduction[5] follows as the weight of each cut is preserved between the hypergraph and the graph after adding the triangles. For a discussion on extending this gadget more generally, see the supplement.

Remark 12 *Reduction to hypergraphs: we would like to emphasize the necessity of the hypergraph version in order for the reduction to work. One might think that just adding extra heavy edges would be sufficient, but there is a technical difficulty with this approach. Consider a triplet constraint $ab|c$; once c is separated from a and b at some level, there is no extra tendency anymore to keep a and b together (i.e. only the similarity weight should play role after this point). This behavior cannot be captured by only adding heavy-weight edges. Instead, one needs to add a heavy edge between a and b that disappears once c is separated, and this is exactly why we need the hyperedge gadget. One can replace the reduction for a one-shot proof, but we believe it will be less modular and less transparent.*

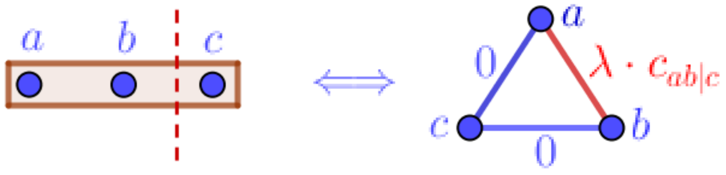

Figure 6.3: Transforming a 3-hyperedge to a triangle.

6.2.3 Dissimilarity HC and Constraint Dependencies

In this section we study dissimilarity-HC, and we look into the problem of designing approximation algorithms for both unconstrained and constrained hierarchical clustering. In [38], they show that average linkage is a $\frac{1}{2}$-approximation for this problem and they propose a top-down approach based on locally densest cut achieving a $(\frac{2}{3} - \epsilon)$-approximation in time $\tilde{O}\left(\frac{n^2(n+m)}{\epsilon}\right)$. Notably, when ϵ gets small the running time blows up.

Here, we prove that the most natural randomized algorithm for this problem, i.e. recursive random cutting, is a $\frac{2}{3}$-approximation with expected running time $O(n \log n)$. We further derandomize this algorithm to get a simple deterministic local-search style $\frac{2}{3}$-approximation algorithm.

If we also have structural constraints for the dissimilarity-HC, we show that the existing approaches fail. In fact we show that they lead to an $\Omega(n)$-approximation factor due to the lack of "exploration" (e.g. recursive densest cut). We then show that recursive random cutting is robust to adding user constraints, and indeed it preserves a constant approximation factor when there are, roughly speaking, constantly many user constraints.

Randomized $\frac{2}{3}$-approximation. Consider the most natural randomized algorithm for hierarchical clustering, i.e. recursively partition each cluster into two, where each point in the current cluster independently flips an unbiased coin and based on the outcome, it is put in one of the two parts.

Theorem 6.2.6 *`Recursive-Random-Cutting` is a $\frac{2}{3}$-approximation for maximizing dissimilarity-HC objective.*

Proof. [Proof sketch.] An alternative view of Dasgupta's objective is to divide the reward of the clustering tree between all possible triples $\{i, j, k\}$, where $(i, j) \in E$ and k is another point (possibly equal to i or j). Now, in any hierarchical clustering tree, if at the moment right before i and j become separated the vertex k has still been in the same cluster as $\{i, j\}$, then this triple contributes w_{ij} to the objective function. We claim this event happens with probability exactly $\frac{2}{3}$. To see this, consider an infinite independent sequence of coin flips for i, j, and k. Without loss of generality, condition on i's sequence to be all heads. The aforementioned event happens only if j's first tales in its sequence happens no later than k's first tales in its sequence. This happens with probability $\sum_{i\geq 1} \frac{1}{2}(\frac{1}{4})^{i-1} = \frac{2}{3}$. Therefore, the algorithm gets the total reward $\frac{2n}{3}\sum_{(i,j)\in E} w_{ij}$ in expectation. Moreover, the total reward of any hierarchical clustering is upper-bounded by $n\sum_{(i,j)\in E} w_{ij}$, which completes the proof of the $\frac{2}{3}$-approximation. ▮

Remark 13 *This algorithm runs in time $O(n \log n)$ in expectation, due to the fact that the binary clustering tree has expected depth $O(\log n)$ (see for example [40]) and at each level we only perform n operations.*

We now derandomize the recursive random cutting algorithm using the *method of conditional expectations*. At every recursion, we go over the points in the current cluster one by one, and decide whether to put them in the "left" partition or "right" partition for the next recursion. Once we make a decision for a point, we fix that point and go to the next one. Roughly speaking, these local improvements can be done in polynomial time, which will result in a simple local-search style deterministic algorithm.

Theorem 6.2.7 *There is a deterministic local-search style $\frac{2}{3}$-approximation algorithm for maximizing dissimilarity-HC objective that runs in time $O(n^2(n+m))$.*

Maximizing the Objective with User Constraints From a practical point of view, one can think of many settings in which the output of the hierarchical clustering algorithm should satisfy user-defined hard constraints. Now, combining the new perspective of maximizing Dasgupta's objective with this practical consideration raises a natural question: which algorithms are robust to adding user constraints, in the sense that a simple variation of these algorithms still achieve a decent approximation factor?

• **Failure of "Non-exploring" Approaches.** Surprisingly enough, there are convincing reasons that adapting existing algorithms for maximizing Dasgupta's objective (e.g. those proposed in [39]) to handle user constraints is either challenging or hopeless. First, bottom-up algorithms, e.g. average-linkage, fail to output a feasible outcome if they only consider each constraint separately and not all the constraints jointly (as we saw in Figure 6.1). Second, maybe more surprisingly, the natural extension of (locally) `Recursive-Densest-Cut`[6] algorithm proposed in [39] to handle user constraints performs poorly in the worst-case, even when we have only one constraint. `Recursive-Densest-Cut` proceeds by repeatedly picking the cut that has maximum density, i.e. $\arg\max_{S\subseteq V} \frac{w(S,\bar{S})}{|S|\cdot|\bar{S}|}$ and making two clusters. To handle the user constraints, we run it recursively on the supergraph generated by the constraints, similar to the approach in Section 6.2.1. Note that once the algorithm resolves a triplet constraint, it also breaks its corresponding supernode.

Now consider the following example in Figure 6.2.3, in which there is just one triplet constraint ab|c. The weight W should be thought of as large and ϵ as small. By choosing appropriate weights on the edges of the clique K_n, we can fool the algorithm into cutting the dense parts in the clique, without ever resolving the $ab|c$ constraint until it is too late. The algorithm gets a gain of $O(n^3+W)$ whereas `OPT` gets $\Omega(nW)$ by starting with the removal of the edge (b,c) and then removing (a,b), thus enjoying a gain of $\approx nW$.

• **Constrained Recursive Random Cutting.** The example in Figure 6.2.3, although a bit pathological, suggests that a meaningful algorithm for this problem should explore cutting low-weight edges that might lead to resolving constraints, maybe randomly, with the hope of unlocking rewarding edges that were hidden before this exploration.

Formally, our approach is showing that the natural extension of recursive random cutting for the constrained problem, i.e. by running it on the supergraph generated by constraints and unpacking supernodes as we resolve the constraints (in a similar fashion to `CSC`), achieves a constant factor approximation when the constraints have bounded *dependency*. In the remaining of this section, we define an appropriate notion of dependency between the constraints, under the name of

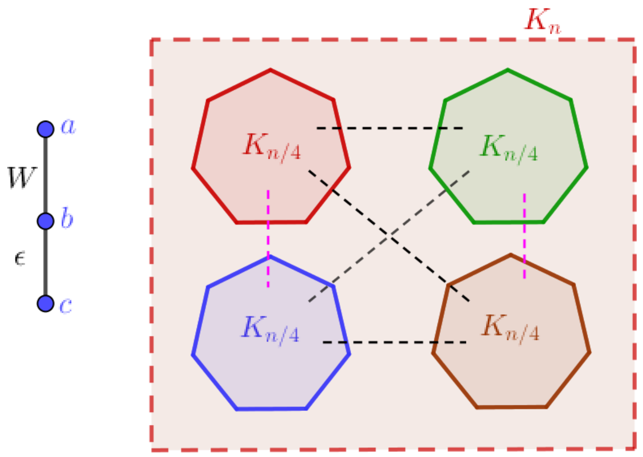

Figure 6.4: $\Omega(n)$-approximation lower bound instance for the constrained `Recursive-Densest-Cut` algorithm.

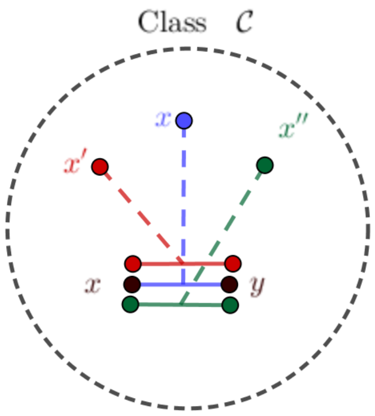

Figure 6.5: Description of a class $\mathcal{C}$ with base $\{x, y\}$.

dependency measure and analyze the approximation factor of constrained recursive random cutting (`Constrained-RRC`) based on this notion.

Suppose we are given an instance of hierarchical clustering with triplet constraints $\{c_1, \ldots, c_k\}$, where $c_i = x^i|y^iz^i, \forall i \in [k]$. For any triplet constraint c_i, lets call the pair $\{y^i, z^i\}$ the *base*, and z^i the *key* of the constraint. We first partition our constraints into equivalence classes $\mathcal{C}_1, \ldots, \mathcal{C}_N$, where $\mathcal{C}_i \subseteq \{c_1, \ldots, c_k\}$. For every i, j, the constraints c_i and c_j belong to the same class $\mathcal{C}$ if they share the same base (see Figure 6.5).

Definition 6.2.1 (Dependency digraph) *The Dependency digraph is a directed graph with vertex set $\{\mathcal{C}_1, \ldots, \mathcal{C}_L\}$. For every i, j, there is a directed edge $\mathcal{C}_i \to \mathcal{C}_j$ if $\exists$ $c = x|yz, c' = x'|y'z'$, such that $c \in \mathcal{C}_i, c' \in \mathcal{C}_j$, and either $\{x, z\} = \{y', z'\}$ or $\{x, y\} = \{y', z'\}$ (see Figure 6.6).*

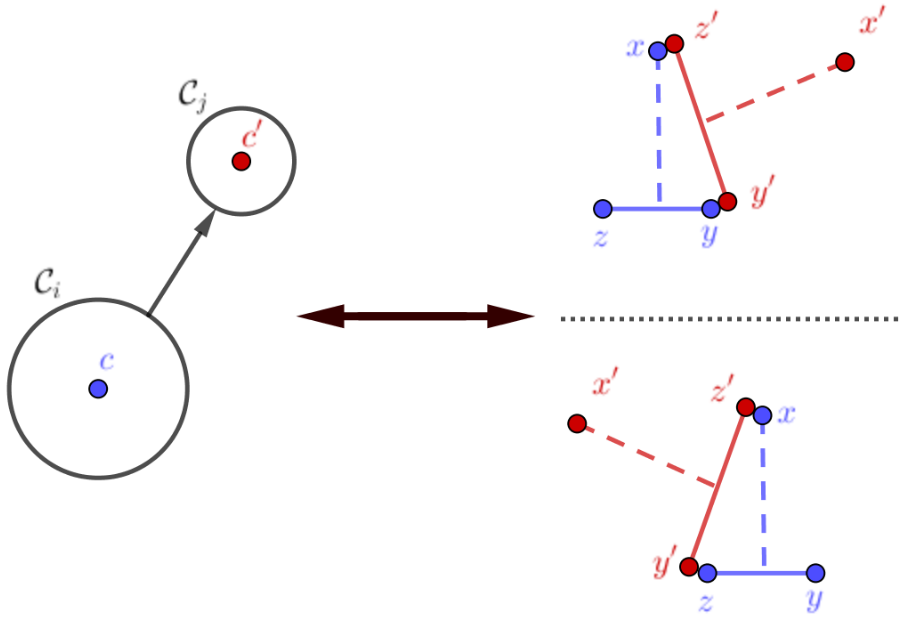

Figure 6.6: Classes $\{\mathcal{C}_i, \mathcal{C}_j\}$, and two situations for having $\mathcal{C}_i \to \mathcal{C}_j$.

The dependency digraph captures how groups of constraints impact each other. Formally, the existence of the edge $\mathcal{C}_i \to \mathcal{C}_j$ implies that all the constraints in $\mathcal{C}_j$ should be resolved before one can separate the two endpoints of the (common) base edge of the constraints in $\mathcal{C}_i$.

Remark 14 *If the constraints $\{c_1, \ldots, c_k\}$ are feasible, i.e. there exists a hierarchical clustering that can respect all the constraints, the dependency digraph is clearly acyclic.*

Definition 6.2.2 (Layered dependency subgraph) *Given any class $\mathcal{C}$, the* layered dependency subgraph *of $\mathcal{C}$ is the induced subgraph in the dependency digraph by all the classes that are reachable from $\mathcal{C}$. Moreover, the vertex set of this subgraph can be partitioned into layers $\{\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_L\}$, where L is the maximum length of any directed path leaving $\mathcal{C}$ and $\mathcal{I}_l$ is a subset of classes where the length of the longest path from $\mathcal{C}$ to each of them is exactly equal to l (see Figure 6.7).*

We are now ready to define a crisp quantity for every dependency graph. This will later help us give a more meaningful and refined *beyond-worst-case* guarantee for the approximation factor of the `Constrained-RRC` algorithm.

Definition 6.2.3 (Dependency measure) *Given any class $\mathcal{C}$, the* dependency measure *of $\mathcal{C}$ is defined as*

$$DM(\mathcal{C}) \triangleq \prod_{l=0}^{L} (1 + \sum_{\mathcal{C}' \in \mathcal{I}_l} |\mathcal{C}'|),$$

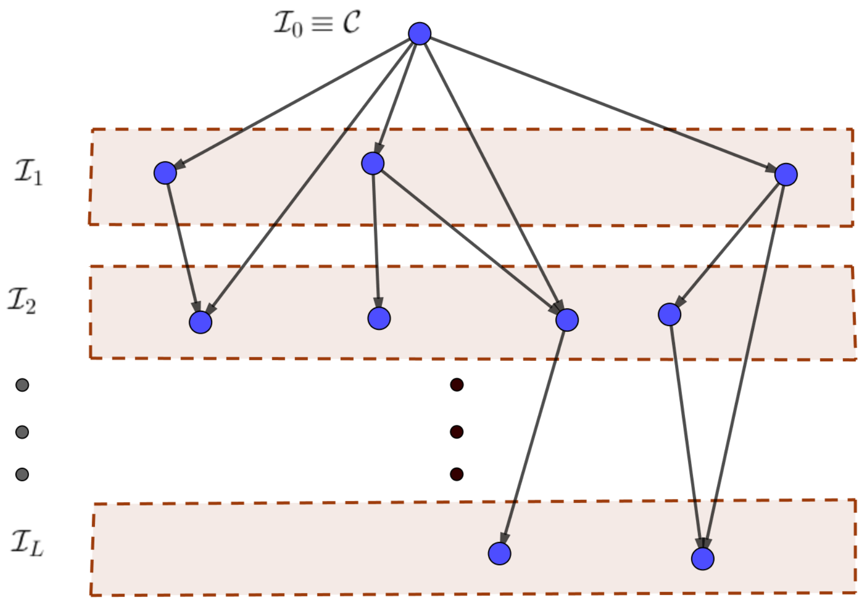

Figure 6.7: Layered dependency subgraph of class $\mathcal{C}$.

where $\mathcal{I}_0, \ldots, \mathcal{I}_L$ are the layers of the dependency subgraph of $\mathcal{C}$, as in Definition 6.2.2. Moreover, the dependency measure of a set of constraints $DMC(\{c_1, \ldots, c_k\})$ is defined as $\max_{\mathcal{C}} DM(\mathcal{C})$, where the maximum is taken over all the classes generated by $\{c_1, \ldots, c_k\}$.

Intuitively speaking, the notion of the dependency measure quantitatively expresses how "deeply" the base of a constraint is *protected* by the other constraints, i.e. how many constraints need to be resolved first before the base of a particular constraint is unpacked and the `Constrained-RRC` algorithm can enjoy its weight. This intuition is formalized through the following theorem, whose proof is deferred to the supplementary materials.

Theorem 6.2.8 *The constrained recursive random cutting (`Constrained-RRC`) algorithm is an α-approximation algorithm for maximizing dissimilarity-HC objective objective given a set of feasible constraints $\{c_1, \ldots, c_k\}$, where*

$$\alpha = \frac{2(1 - k/n)}{3 \cdot DMC(\{c_1, \ldots, c_k\})} \leq \frac{2(1 - k/n)}{3 \cdot \max_{\mathcal{C}} DM(\mathcal{C})}$$

Corollary 6.2.9 *`Constrained-RRC` is an $O(1)$-approximation for maximizing dissimilarity-HC objective, given feasible constraints of constant dependency measure.*

6.3 Old Biology Problems: Triplets/Quartets Consistency

Here we base our presentation on our manuscript [34]. We will only give some hardness proofs for tree ordering CSPs here but if you are interested on how one can get improvements under some uniform sampling model for sampling constraints, please see the final version of [34].

The high level of our work can be summarized as follows: We consider standard data analysis tasks, where the goal is to construct a ranking, clustering or phylogenetic tree by aggregating, perhaps inconsistent, information on overlapping sets of elements. Such information includes relative ordering constraints for ranking (e.g., pairwise comparisons, "betweenness" etc.), "must-link/cannot-link" constraints for clustering, whereas for hierarchical clustering, the analogous constraints are ("desired/forbidden") triplets or quartets, all specifying ordering relations to be included or avoided in the final output. In many instances, even checking consistency is intractable, so the goal becomes to maximize the extent of agreement with the provided information. In this paper, we first address an open question in computational biology raised in several previous works, by presenting near optimal hardness of approximation for several ordering problems on trees (triplets/quartets consistency). One consequence is optimal hardness for the *forbidden* triplets problem and to the best of our knowledge this is the first tight hardness of approximation result for a tree ordering problem. Then, we show how simple algorithms based on variations of MAX CUT with negative weights obtain improved approximations for all considered problems under a query model where noisy constraints are sampled uniformly at random from a ground-truth solution.

Recall some definitions about betweenness and non-betweenness from subsection 1.4.1.

6.3.1 Hardness for Rooted Triplets Consistency

We prove that under the UNIQUE GAMES CONJECTURE, it is hard to approximate the Desired Triplets Consistency problem better than a factor of $\frac{2}{3}$, even in the unweighted case. Notice that the current best approximation is $\frac{1}{3}$ achieved by a random tree (or a simple greedy algorithm). In fact our result is slightly stronger: it is hard to distinguish between two instances one of which is almost perfect (e.g., 99% of constraints are consistent) and the other is far from perfect (e.g., 67% of constraints are consistent). We base our hardness result on the following theorem by Guruswami et al. [58, 11] about the *Non-Betweeness* problem and its $\frac{2}{3}$-inapproximability:

Fact 6.3.1 *Let K be the total number of triplets constraints in an instance of* NON-BETWEENNESS. *For any $\epsilon > 0$, it is NP-hard to distinguish between* NON-BETWEENNESS *instances of the following two cases:*
YES*: $val(\pi^*) \geq (1-\epsilon)K$, i.e. the optimal permutation satisfies almost all constraints.*
NO*: $val(\pi^*) \leq (\frac{2}{3}+\epsilon)K$, i.e. the optimal permutation does not satisfy more than 2/3 fraction of the constraints.*

Given the above fact, we prove our $\frac{2}{3}$-inapproximability result for Triplets Consistency:

Theorem 6.3.2 *Let K be the total number of the triplet constraints in an instance of Desired Triplets Consistency. For any $\delta > 0$, it is NP-hard to distinguish between instances of the following two cases:*
***YES**: $val(T^*) \geq (\frac{1}{2} - \delta)K$, i.e. the optimal tree satisfies almost half of all the triplet constraints.*
***NO**: $val(T^*) \leq (\frac{1}{3} + \delta)K$, i.e. the optimal tree does not satisfy more than $\frac{1}{3}$ fraction of the triplet constraints.*

Then, our $\frac{2}{3}$-inapproximability result follows directly from the gap of these instances: $\frac{1}{3}/\frac{1}{2} = \frac{2}{3}$.
Proof. Start with a YES instance of the NON-BETWEENNESS problem with optimal permutation π^* and $val(\pi^*) \geq (1-\epsilon)K$. Viewing each NON-BETWEENNESS constraint as a desired triplet, we show how to construct a tree T such that $val(T) \geq (\frac{1}{2} - \delta(\epsilon))K$. In fact, the construction is straightforward: simply assign the n labels, either in the order they appear in π^* or reversed, as the leaves of a caterpillar tree (every internal node has at least one child that is a leaf). Observe that this tree satisfies:

$$val(T) \geq (1-\epsilon)K/2$$

This is because if a NON-BETWEENNESS constraint $ab|c$ was obeyed by π^*, it will also be obeyed by one of the two caterpillar trees above: if c appears first in the permutation then the former caterpillar will obey $ab|c$ as c gets separated first, otherwise if c appears last, then the reversed caterpillar tree will obey $ab|c$. Here the $\frac{1}{2}$ factor is tight, since for example, the two NON-BETWEENNESS constraints $ab|c$ and $bc|a$ are both satisfied by the ordering abc, but when viewed as desired triplets, they cannot both be satisfied by a tree.

The NO instance is slightly more challenging. Start with a NO instance of the NON-BETWEENNESS problem with optimal π^* of value $val(\pi^*) \leq (\frac{2}{3} + \epsilon)K$. Viewing the NON-BETWEENNESS constraints as desired triplets, we show that the optimum tree T^* cannot achieve better than $> (1/3 + 2\epsilon)K$, because this would imply that $val(\pi^*) > (\frac{2}{3} + \epsilon)K$, which is a contradiction.

For this, assume that some tree T scored a value $val(T) > (1/3 + 2\epsilon)K$. We will construct a permutation π from the tree T with value $val(\pi) > (2/3 + \epsilon)K$. Observe that directly projecting the leaves of T onto a line (just outputting the n leaves from left to right as they appear in the tree) would already satisfy $> (1/3 + 2\epsilon)K$, since every desired triplet $ab|c$ obeyed by the tree, will also be obeyed (as a NON-BETWEENNESS constraint) by π as c will either be first or last among the three labels a, b, c.

Moreover, there are potentially desired triplet constraints that are disobeyed by the tree T, yet obeyed by the permutation. We know that the number of remaining constraints is $K - (1/3 + 2\epsilon)K = (2/3 - 2\epsilon)K$. By randomly swapping each left and right child in the tree T before we do the projection to the permutation π, will actually lead to an excess of $1/2 \cdot (2/3 - 2\epsilon)K = (1/3 - \epsilon)K$ number of NON-BETWEENNESS constraints. To see this notice that for every triplet that is disobeyed in the tree, there is a $\frac{1}{2}$ probability that it becomes obeyed in the permutation. Summing up, we get $val(\pi) > (1/3 + 2\epsilon)K + (1/3 - \epsilon)K > (2/3 + \epsilon)K \implies val(\pi^*) \geq val(\pi) > (2/3 + \epsilon)K$, a contradiction. $\blacksquare$

6.3.2 Hardness for Forbidden Triplets: Random is Optimal

We prove that under the Unique Games Conjecture, it is hard to approximate the Forbidden Triplets Consistency problem better than a factor of $\frac{2}{3}$, even in the unweighted case. Notice that the current best approximation is in fact $\frac{2}{3}$ achieved by a random tree (or a simple greedy algorithm), hence we settle the computational complexity of the problem. Our result is slightly stronger: it is hard to distinguish between two instances one of which is almost perfect (e.g., 99% of constraints are consistent) and the other is far from perfect (e.g., 67% of constraints are consistent). We base our hardness result on the following theorem by Guruswami et al. [58, 11] about the Betweenness problem and its $\frac{1}{3}$-inapproximability:

Fact 6.3.3 *Let K be the total number of triplets constraints in an instance of* Betweenness*. For any $\epsilon > 0$, it is UGC-hard to distinguish between* Betweenness *instances of the following two cases:*
YES*: $val(\pi^*) \geq (1-\epsilon)K$, i.e. the optimal permutation satisfies almost all constraints.*
NO*: $val(\pi^*) \leq (\frac{1}{3}+\epsilon)K$, i.e. the optimal permutation does not satisfy more than 1/3 fraction of the constraints.*

Given the above fact, we prove our $\frac{2}{3}$-inapproximability result for Forbidden Triplets Consistency:

Theorem 6.3.4 *Let K be the total number of the triplet constraints in an instance of Forbidden Triplets Consistency. For any $\delta > 0$, it is UGC-hard to distinguish between instances of the following two cases:*
YES*: $val(T^*) \geq (1-\delta)K$, i.e. the optimal tree satisfies almost half of all the triplet constraints.*
NO*: $val(T^*) \leq (\frac{2}{3}+\delta)K$, i.e. the optimal tree does not satisfy more than $\frac{2}{3}$ fraction of the triplet constraints.*

Then, our $\frac{2}{3}$-inapproximability result follows directly from the gap of these instances: $\frac{2}{3}/1 = \frac{2}{3}$.
Proof. Start with a YES instance of the Betweenness problem with optimal permutation π^* and $val(\pi^*) \geq (1-\epsilon)K$. Viewing each Betweenness constraint $a|b|c$ as a forbidden triplet $ac|b$, we show how to construct a tree T such that $val(T) \geq (\frac{1}{-}\delta(\epsilon))K$. In fact, the construction is straightforward: simply assign the n labels, in the order they appear in π^*, as the leaves of a caterpillar tree (every internal node has its left child being a leaf). Observe that this caterpillar tree satisfies:

$$val(T) \geq (1-\epsilon)K$$

This is because if a Betweenness constraint $a|b|c$ was obeyed by π^*, it will also be avoided (viewed as a forbidden triplet $ac|b$) by the caterpillar tree above: if a appears first in the permutation then the caterpillar will avoid $ac|b$ as a gets separated first, otherwise if c appears first, then again the caterpillar tree will avoid $ac|b$ as c gets separated first.

The NO instance is slightly more challenging. Start with a NO instance of the Betweenness problem with optimal π^* of value $val(\pi^*) \leq (\frac{1}{3} + \epsilon)K$. Viewing the Betweenness constraints as forbidden triplets, we show that the optimum tree T^* cannot achieve better than $> (2/3 + 2\epsilon)K$, because this would imply that $val(\pi^*) > (\frac{1}{3} + \epsilon)K$, which is a contradiction.

For this, assume that some tree T scored a value $val(T) > (2/3 + 2\epsilon)K$. We will construct a permutation π from the tree T with value $val(\pi) > (1/3 + \epsilon)K$, a contradiction. Notice that there are forbidden triplets that may be avoided by the tree, yet obeyed by the permutation: for example for a forbidden triplet $t = ac|b$, the tree R that first removes a and then splits b, c will successfully avoid t, however the permutation acb can come from R by projection, however acb do not obey the Betweenness constraint $a|b|c$.

Hence directly projecting the leaves of T onto a line may not satisfy $> (1/3 + 2\epsilon)K$, since every forbidden triplet $ac|b$ avoided by T, can be ordered by this projected permutation in a way that would not obey the corresponding Betweenness constraint $a|b|c$.

However, just by randomly swapping each left and right child for every internal node in the tree before we do the projection to the permutation, would satisfy $1/2 \cdot (2/3 + 2\epsilon)K = (1/3 + \epsilon)K$ number of constraints. To see this, note that with probability $\frac{1}{2}$ a forbidden $ac|b$ avoided by T will be mapped to the desired abc (and not acb) or cba (and not cab) ordering.

Finally, we get $val(\pi) > (1/3 + \epsilon)K \implies val(\pi^*) \geq val(\pi) > (1/3 + \epsilon)K$, a contradiction that we were given a NO instance. ▮

Chapter 7

Conclusion & Open Questions

Prediction is very difficult,
especially about the future.

-Niels Bohr

In this chapter, we would like to conclude this thesis with a summary of our contributions and also state some open questions and conjectures regarding Hierarchical Clustering.

7.1 Conclusion

Hierarchical Clustering is a tool used across different scientific areas. It originated in Biology and then passed on as a powerful method to more computational tasks ubiquitous in computer science. However, no objective had been formulated enabling a theoretical understanding behind it. This thesis builds upon a recent "global" objective function associated with hierarchical clustering that was proposed by Dasgupta (2016) [43]. We develop theory for it, building on top of standard approximation algorithms, we unveil some interesting connections with old and new algorithms and make progress on older phylogenetic questions. Given that progress in standard clustering has been based on a variety of objectives, starting from the 1950s, that led to a comprehensive theory on clustering, we hope our work will help develop the much needed analogous theory in hierarachical clustering.

7.2 List of Open Problems and Conjectures

We separate the questions based on the relevant chapters in this thesis:

From Chapter 4:

(1) Can one obtain an approximation preserving reduction from Sparsest Cut to Dasgupta's cost? So far, we have seen the other way, how to use a black-box for Sparsest Cut in order to obtain a tree.

(2) Can we achieve a better approximation for Cohen-Addad et al. objective?

(3) Can we achieve a better approximation for Moseley-Wang objective? Recent work by Alon et al. [5], during the writing of this thesis improved our analysis of Max Uncut Bisection to get a 0.585-approximation. Can we do even better? Can we prove better than APX-hardness that we provided? [2].

(4) Can we substantially accelerate HC algorithms to run in huge graphs by using random walks and spectral guarantees, instead of sparsest cut explicitly? This would be really interesting.

(5) Another direction for research is going beyond worst case analysis for this problem. What can we say about exact recovery on γ-stable instances under the Bilu-Linial [21] notion of stability? Roughly speaking, a stable instance under this notion has the property that the structure of the optimum solution does not change even if weights in the instance are changed by a factor of γ. For example, in [74] they show that the standard SDP relaxation for Max-Cut is integral if the instance is sufficiently stable ($\gamma \geq c\sqrt{\log n}\log n$ for some absolute constant $c > 0$). Stability for clustering and other problems has been extensively studied; see [15, 21, 74, 36, 7] and references therein. It would be interesting to study stable instances for hierarchical clustering, in particular the performance of the recursive Sparsest Cut algorithm (or our SDP-HC) on such instances. This would not only explain the success of certain heuristics for HC based on finding sparsest cuts, but also justify their use in practice (assuming that stability is a good model for instances in real applications).

From Chapter 5:

(1) Can one get an improvement over $\frac{1}{3}$ for the problem of maximizing $\mathcal{F}^+(\mathcal{T})$ as a function of d for small d (with fast algorithms) ?

(2) Can projection on low-dimensional subspaces be used to improve the approximation ratio for the high-dimensional case even further?

(3) Does Average-Linkage achieve a 3/4-approximation for 1-dimensional data?

From Chapter 6, we believe the following conjecture is true for tree ordering CSPs similar to the case for permutations shown in [58]:

Conjecture 1 *For any tree ordering CSP, no polynomial time algorithm can achieve a constant factor approximation better than random.*

We proved partially the conjecture for the specific problem of forbidden triplets, as random achieves a $\frac{2}{3}$-approximation and we showed that one cannot beat random assuming UNIQUE GAMES CONJECTURE. For maximizing rooted triplets consistency, we were able to show only a $\frac{2}{3}$ hardness result, but we believe the right answer is $\frac{1}{3}$, i.e., the same of the performance of a random tree.

Appendix A

Omitted Proofs, Discussions and Experiments

The ability to play chess is the sign of a gentleman. The ability to play chess *well* is the sign of a wasted life.

-Paul Morphy

A.1 Omitted Proofs from Chapters.

A.1.1 Missing proofs and discussion in Section 6.2.1

Proof. [Proof of Proposition 6.1.1] For nodes $u, v \in T$, let $P(u)$ denote the parent of u in the tree and $\text{LCA}(u, v)$ denote the lowest common ancestor of u, v. For a leaf node $l_i, i \in [k]$, we say that its label is l_i, whereas for an internal node of T, we say that its label is the label of any of its two children. As long as there are any two nodes a, b that are siblings (i.e. $P(a) \equiv P(b)$), we create a constraint $ab|c$ where c is the label of the second child of $P(P(a))$. We delete leaves a, b from the tree and repeat until there are fewer than 3 leaves left. To see why the above procedure will only create at most k constraints, notice that every time a new constraint is created, we delete two nodes of the given tree T. Since T has k leaves and is binary, it can have at most $2k - 1$ nodes in total. It follows that we create at most $\frac{2k-1}{2} < k$ triplet constraints. For the equivalence between the constraints imposed by T and the created triplet constraints, observe that *all* triplet constraints we create are explicitly imposed by the given tree (since we only create constraints for two leaves that are siblings) and that for any three datapoints $a, b, c \in T$ with $\text{LCA}(a, c)$=$\text{LCA}(b, c)$, our set of triplet constraints will indeed imply $ab|c$, because $\text{LCA}(a, b)$ appears further down the tree than

LCA(a,c) and hence a,b become siblings before a,c or b,c. ▮

Proof. [Proof of Fact 6.2.2 from [27]] We will measure the contribution of an edge $e=(u,v)\in E$ to the RHS and to the LHS. Suppose that r denotes the size of the *minimal* cluster in $\texttt{OPT}$ that contains both u and v. Then the contribution of the edge $e=(u,v)$ to the LHS is by definition $r\cdot w_e$. On the other hand, $(u,v)\in \texttt{OPT}(t), \forall t\in\{0,...,r-1\}$. Hence the contribution to the RHS is also $r\cdot w_e$. ▮

Proof. [Proof of Fact 6.2.3 from [27]] We rewrite $\texttt{OPT}$ using the fact that

$$w(\texttt{OPT}(t))\geq 0$$

at every level $t\in[n]$:

$$\begin{aligned}6k\cdot\texttt{OPT}&=6k\sum_{t=0}^{n}w(\texttt{OPT}(t))\\&=6k(w(\texttt{OPT}(0))+\cdots+w(\texttt{OPT}(n)))\\&\geq 6k(w(\texttt{OPT}(0))+\cdots+w(\texttt{OPT}(\lfloor\tfrac{n}{6k}\rfloor)))\\&=\sum_{t=0}^{n}w(\texttt{OPT}(\lfloor\tfrac{t}{6k}\rfloor))\end{aligned}$$

▮

Proof. [Proof of Lemma 6.2.2] By using the previous lemma we have:

$$\texttt{CRSC}=\sum_A r_A w(B_1,B_2)\leq\leq O(\alpha_n)\sum_A s_A w(\texttt{OPT}(\lfloor\tfrac{r_A}{6k_A}\rfloor)\cap A)$$

Observe that $w(\texttt{OPT}(t))$ is a decreasing function of t, since as t decreases, more and more edges are getting cut. Hence we can write:

$$\sum_A s_A\cdot w(\texttt{OPT}(\lfloor\tfrac{r_A}{6k}\rfloor)\cap A)\leq\sum_A\sum_{t=r_A-s_A+1}^{r_A}w(\texttt{OPT}(\lfloor\tfrac{r_A}{6k_A}\rfloor)\cap A)$$

To conclude with the proof of the first part all that remains to be shown is that:

$$\sum_A\sum_{t=r_A-s_A+1}^{r_A}w(\texttt{OPT}(\lfloor\tfrac{t}{6k_A}\rfloor)\cap A)\leq\sum_{t=0}^{n}w(\texttt{OPT}(\lfloor\tfrac{t}{6k}\rfloor))$$

To see why this is true consider the clusters A with a contribution to the LHS. We have that $r_A-s_A+1\leq t\leq r_A$, hence $|B_2|<t$ meaning that A is a *minimal* cluster of size $|A|\geq t>|B_2|\geq|B_1|$, i.e. if both A's children are of size less than t, then this cluster A contributes such a term. The set of all such A form a disjoint partition of V because of the definition for minimality (in order for them to overlap in the hierarchical clustering, one of them needs to be ancestor of the other and this cannot happen because of minimality). Since $\texttt{OPT}(\lfloor \frac{t}{6k} \rfloor) \cap A$ for all such A forms a disjoint partition of $\texttt{OPT}(\lfloor \frac{t}{6k} \rfloor)$, the claim follows by summing up over all t.

Note that so far our analysis handles clusters A with size $r_A \geq 6k$. However, for clusters with smaller size $r_A < 6k$ we can get away by using a crude bound for bounding the total cost and still not affecting the approximation guarantee that will be dominated by $O(k\alpha_n)$:

$$\sum_{|A|<6k} r_A w(B_1, B_2) < 6k \cdot \sum_{ij \in E} w_{ij} = 6k \cdot \texttt{OPT}(1) \leq 6k \cdot \texttt{OPT}$$

This finishes the proof of the fact. ▮

Theorem A.1.1 (The divisive algorithm using balanced cut) *Given a weighted graph $G(V, E, w)$ with k triplet constraints $ab|c$ for $a, b, c \in V$, the constrained recursive balanced cut algorithm (same as CRSC, but using balanced cut instead of sparsest cut) outputs a HC respecting all triplet constraints and achieves an $O(k\alpha_n)$-approximation for the HC objective (6.1).*

Proof. It is not hard to show that one can use access to balanced cut rather than sparsest cut and achieve the same approximation factor by the recursive balanced cut algorithm.

We will follow the same notation as in the sparsest cut analysis and we will use some of the facts and inequalities we previously proved about `OPT(t)`. Again, for a cluster A of size r, the important observation is that the partition $A_1, \ldots, A_l$ (at the end, we will again choose $l = 6k_A$) induced inside the cluster A by $\texttt{OPT}(\frac{r}{l})$ can be separated into two groups, let's say (C_1, C_2) such that $r/3 \leq |C_1|, |C_2| \leq 2r/3$. In other words we can demonstrate a Balanced Cut with ratio $\frac{1}{3} : \frac{2}{3}$ for the cluster A. Since we cut fewer edges when creating C_1, C_2 compared to the partitioning of $\texttt{OPT}(\frac{r}{l})$:

$$w(C_1, C_2) \leq w(\texttt{OPT}(\lfloor \tfrac{r}{l} \rfloor) \cap A)$$

By the fact we used an α_n-approximation to balanced cut we can get the following inequality (similarly to Lemma 6.2.1):

$$r \cdot w(C_1, C_2) \leq O(\alpha_n) \cdot s \cdot w(\texttt{OPT}(\lfloor \tfrac{r}{l} \rfloor) \cap A)$$

Finally, we have to sum up over all the clusters A (now in the summation we should write r_A, s_A instead of just r, s, since there is dependence in A) produced by the constrained recursive balanced cut algorithm for Hierarchical Clustering and we get that we can approximate the HC objective function up to $O(k\alpha_n)$. ▮

Remark 15 *Using balanced-cut can be useful for two reasons. First, the runtime of sparsest and balanced cut on a graph with n nodes and m edges are $\tilde{O}(m+n^{1+\epsilon})$. When run recursively however as* *in our case, taking recursive sparsest cuts might be worse off by a factor of n (in case of unbalanced splits at every step) in the worst case. However, recursive balanced cut is still $\tilde{O}(m+n^{1+\epsilon})$. Second, it is known that an α-approximation for the sparsest cut yields an $O(\alpha)$-approximation for balanced cut, but not the other way. This gives more flexibility to the balanced cut algorithm, and there is a chance it can achieve a better approximation factor (although we don't study it further in this paper).*

A.1.2 Missing proofs in Section 6.2.2

Proof. [Proof sketch of Proposition 6.2.5] Here the main obstacle is similar to the one we handled when proving Theorem (6.2.1): for a given cluster A created by the `R-HSC` algorithm, different constraints are, in general, active compared to the `OPT` decomposition for this cluster A. Note of course, that `OPT` itself will not respect all constraints, but because we don't know which constraints are active for `OPT`, we still need to use a charging argument to low levels of `OPT`. Observe that here we are allowed to cut an edge ab even if we had the $ab|c$ constraint (incurring the corresponding cost $c_{ab|c}$), however we cannot possibly hope to charge this to the `OPT` solution, as `OPT`, for all we know, may have respected this constraint. In the analysis, we crucially use a merging procedure between sub-clusters of A having active constraints between them and this allows us to compare the cost of our `R-HSC` with the cost of `OPT` . ∎

Proof. [3-hyperedges to triangles for general weights] Even though the general reduction presented in Section 6.2.2 (Figure 6.3) to transform a 3-hyperedge to a triangle is valid even for general instances of HSC with 3-hyperedges and arbitrary weights, the reduced sparsest cut problem may have negative weights, e.g. when $w_{bc|a} + w_{ac|b} < w_{ab|c}$. To the best of our knowledge, sparsest cut with negative weights has not been studied. Notice however that if the original weights $w_{bc|a}, w_{ac|b}, w_{ab|c}$ satisfy the triangle inequality (or as a special case, if two of them are zero which is usually the case when we have a triplet constraints), then we can actually solve (approximately) the HSC instance, as the sparsest cut instance will only have non-negative weights. ∎

A.1.3 Missing proofs in Section 6.2.3

Proof. [Proof of Theorem 6.2.6] We start by looking at the objective value of any algorithm as the summation of contributions of different triples i, j and k to the objective, where $(i, j) \in E$ and k is some other point (possibly equal to i or j).

$$\mathtt{OBJ} = \sum_{(i,j)\in E} w_{ij}|T_{ij}| = \sum_{(i,j)\in E, k\in V} w_{ij}\mathbf{1}\{k \in \text{leaves}(T_{ij})\} = \sum_{(i,j)\in E}\sum_{k\in V} \mathcal{Y}_{i,j,k},$$

where random variable $\mathcal{Y}_{i,j,k}$ denotes the contribution of the edge (i, j) and vertex k to the objective value. The vertex k is a leaf of T_{ij} if and only if right before the time that i and j gets separated k

is still in the same cluster as i and j. Therefore,

$$\mathcal{Y}_{i,j,k} = w_{ij}\mathbf{1}\{i \text{ separates from } k \text{ no earlier than } j\ \}$$

We now show that $\mathbf{E}\left[\mathcal{Y}_{i,j,k}\right] = \frac{2}{3}w_{ij}$. Given this, the expected objective value of recursive random cutting algorithm will be at least $\frac{2n}{3}\sum_{(i,j)\in E} w_{ij}$. Moreover, the objective value of the optimal hierarchical clustering, i.e. maximizer of the Dasgupta's objective, is no more than $n\sum_{(i,j)\in E} w_{ij}$, and we conclude that recursive random cutting is a $\frac{2}{3}$-approximation. To see why $\mathbf{E}\left[\mathcal{Y}_{i,j,k}\right] = \frac{2}{3}w_{ij}$, think of randomized cutting as flipping an independent unbiased coin for each vertex, and then deciding on which side of the cut this vertex belongs to based on the outcome of its coin. Look at the sequence of the coin flips of i, j and k. Our goal is to find the probability of the event that for the first time that i and j sequences are not matched, still i's sequence and k's sequence are matched up to this point, or still j's sequence and k's sequence are matched up to this. The probability of each of these events is equal to $\frac{1}{3}$. To see this for the first event, suppose i's sequence is all heads (H). We then need the pair of coin flips of (j,k) to be a sequence of (H,H)'s ending with a (T,H), and this happens with probability $\sum_{i\geq 1}(\frac{1}{4})^i = \frac{1}{3}$. The probability of the second event is similarly calculated. Now, these events are disjoint. Hence, the probability that i is separated from k no earlier than j is exactly $\frac{2}{3}$, as desired. ∎

Proof. [Proof of Theorem 6.2.7] We derandomize the recursive random cutting algorithm using the *method of conditional expectations*. At every recursion, we go over the points in the current cluster one by one, and decide whether to put them in the "left" partition or "right" partition for the next recursion. Once we make a decision for a point, we fix that point and go to the next one. Now suppose for a cluster C we have already fixed points $S \subseteq C$, and now we want to make a decision for $i \in C \setminus S$. The reward of assigning to left(right) partition is now defined as the expected value of recursive random cutting restricted to C, when the points in S are fixed (i.e. it is already decided which points in S are going to the left partition and which ones are going to the right partition), i goes to the left(right) partition and $j \in C \setminus (\{i\} \cup S)$ are randomly assigned to either the left or right. Note that these two rewards (or the difference of the two rewards) can be calculated *exactly* in polynomial time by considering all triples consisting of an edge and another vertex, and then calculating the probability that this triple contributes to the objective function (this is similar to the proof of Theorem 6.2.6, and we omit the details for brevity here). Because we know the randomized assignment of i gives a $\frac{2}{3}$-approximation (Theorem 6.2.6), we conclude that assigning to the better of left or right partition for every vertex will remain to be at least a $\frac{2}{3}$-approximation. For running time, we have at most n clusters to investigate. Moreover, a careful counting argument shows that the total number of operations required to calculate the differences of the rewards of assigning to left and right partitions for all vertices is at most $n(n+2m)$. Hence, the running time is bounded by $O(n^2(n+m))$. ∎

Proof. [Proof sketch of Theorem 6.2.8.] Before starting to prove the theorem, we prove the following simple lemma.

Lemma A.1.1 *There is no edge between any two classes in the same layer $\mathcal{I}_l$.*

Proof. [Proof of Lemma A.1.1] If such an edge exists, then there is a path of length $l+1$ from $\mathcal{C}$ to a class in $\mathcal{I}_l$, a contradiction. ▮

Now, similar to the proof of Theorem 6.2.6, we consider every triple $\{x, y, z\}$, where $(x, y) \in E$ and z is another point , but this time we only consider z's that are not involved in any triplet constraint (there are at least $n-k$ such points). We claim with probability at least $\frac{2}{3 \cdot \mathrm{DMC}(\{c_1,\ldots,c_k\})}$ the supernode containing z is still in the same cluster as supernodes containing x and y right before x and y gets separated. By summing over all such triples, we show that the algorithm gets a gain of at least $\frac{2(n-k)}{3 \cdot \mathrm{DMC}(\{c_1,\ldots,c_k\})} \sum_{(x,y)\in E} w_{xy}$, which proves the α-approximation as the optimal clustering has a reward bounded by $n \sum_{(x,y)\in E} w_{xy}$.

To prove the claim, if (x, y) is not the base of any triplet constraint then a similar argument as in the proof of Theorem 6.2.6 shows the desired probability is exactly $\frac{2}{3}$ (with a slight adaptation, i.e. by looking at the coin sequences of supernodes containing x and y, which are going to be disjoint in this case at all iterations, and the coin sequence of z). Now suppose (x, y) is the base of any constraint c and suppose c belongs to a class $\mathcal{C}$. Consider the layered dependency subgraph of $\mathcal{C}$ as in Definition 6.2.2 and let the layers to be $\mathcal{I}_0, \ldots, \mathcal{I}_L$. In order for z to be in the same cluster as x and y when they get separated, a chain of $L+1$ independent events needs to happen. These events are defined inductively; for the first event, consider the coin sequence of z, coin sequence of (the supernode containing all the bases of) constraints in $\cup_{l=0}^{L}\mathcal{I}_l$ and coin sequences of all the keys of constraints in $\mathcal{I}_L$ (there are $\sum_{\mathcal{C}'\in\mathcal{I}_L}|\mathcal{C}'|$ of them). Without loss of generality, suppose the coin sequence of (the supernode containing) $\cup_{l=0}^{L}\mathcal{I}_l$ is all heads. Now the event happens only if at the time z flips its first tales all keys of $\mathcal{I}_L$ have already flipped at least one tales. Conditioned on this event happening, all the constraints in $\mathcal{I}_L$ will be resolved and z remains in the same cluster as x and y. Now, remove $\mathcal{I}_L$ from the dependency subgraph and repeat the same process to define the events $2, \ldots, L$ in a similar fashion. For the l^{th} event to happen, we need to look at $1+\sum_{\mathcal{C}'\in\mathcal{I}_L}|\mathcal{C}'|$ number of i.i.d. symmetric geometric random variable, and calculate the probability that first of them is no smaller than the rest. This event happens with a probability at least $\left(1+\sum_{\mathcal{C}'\in\mathcal{I}_L}|\mathcal{C}'|\right)^{-1}$. Moreover the events are independent, as there is no edge between any two classes in $\mathcal{I}_l$ for $l \in [L]$, and different classes have different keys. After these L events, the final event that needs to happen is when all the constraints are unlocked, and z needs to remain in the same cluster as x and y at the time they get separated. This event happens with probability $\frac{2}{3}$. Multiplying all of these probabilities due to independence implies the desired approximation factor. ▮

A.2 Experiments and Discussion from Chapter 6

The purpose of this section is to present the benefits of incorporating triplet constraints when performing Hierarchical Clustering. We will focus on real data using the Zoo dataset ([71]) for a taxonomy application. We demonstrate that using our approach, the performance of simple recursive spectral clustering algorithms can be improved by approximately 9% as measured by the Dasgupta's Hierarchical Clustering cost function (6.1). More specifically:

- *The Zoo dataset*: It contains 100 animals forming 7 different categories (e.g. mammals, amphibians etc.). The features of each animal are provided by a 16-dimensional vector containing information such as if the animal has hair or feathers etc.

- *Evaluation method*: Given the feature vectors, we can create a similarity matrix $M(\cdot,\cdot)$ indexed by the labels of the animals. We choose the widely used cosine similarity to create M.

- *Algorithms*: We use a simple implementation of spectral clustering based on the second eigenvector of the normalized Laplacian of M. By applying the spectral clustering algorithm once, we can create two clusters; by applying it recursively we can create a complete hierarchical decomposition, which is ultimately the output of the HC algorithm.

- *Baseline comparison*: Since triplet constraints are especially useful when there is noisy information (i.e. noisy features), we simulate this situation by hiding some of the features of our Zoo dataset. Specifically, when we want to find the target HC tree T^*, we use the full 16-dimensional feature vectors, but for the comparison between the unconstrained and the constrained HC algorithms we will use a noisy version of the feature vectors which consists of only the first 10 coordinates from every vector.

  In more detail, the first step in our experiments is to evaluate the cost of the target clustering T^*. For this, we use the full feature vectors and perform repeated spectral clustering to get a hierarchical decomposition (without incorporating any constraints). We call this cost `OPT`.

  The second step is to perform unconstrained HC but with noisy information, i.e. to run the spectral clustering algorithm repeatedly on the 10-dimensional feature vectors (again without taking into account any triplet constraints). This will output a hierarchical tree that has cost in terms of the Dasgupta's HC cost `Unconstrained_Noisy_Cost`.[1]

  The final step is to choose some structural constraints (that are valid in T^*)[2] and perform again HC with noisy information. We again use the 10-dimensional feature vectors but the spectral clustering algorithm is allowed only cuts that do not violate any of the given structural constraints. Repeating until we get a decomposition gives us the final output which will have cost in terms of the Dasgupta's HC cost `Constrained_Noisy_Cost`.

#animals	`OPT`	`Unconstrained_Noisy_Cost`	`Constrained_Noisy_Cost`	% Improvement
20	1137	1286	1142	12.63
50	23088	25216	23443	7.68
80	89256	99211	90419	9.85
100	171290	190205	173499	9.75

Table A.1: Results obtained for the Zoo dataset. The improvement corresponds to the lower cost of the output HC tree after incorporating structural constraints, even in the presence of noisy features. Observe that in all cases the performance of `Constrained_Noisy_Cost` is extremely close to the `OPT` cost.

The first main result of our experimental evaluation is that the `Constrained_Noisy_Cost` is surprisingly close to `OPT`, even though to get the `Constrained_Noisy_Cost` the features used were noisy and the second main result is that incorporating the structural constraints yields $\approx 9\%$ improvement over the noisy unconstrained version of HC with cost `Unconstrained_Noisy_Cost`. Now that we have presented the experimental set-up, we can proceed by describing our results and final observations in greater depth.

A.2.1 Experimental Results

We ran our experiments for $20, 50, 80$ and 100 animals from the Zoo dataset and for the evaluation of the % improvement in terms of the Dasgupta's HC cost (6.1), we used the following formula:

$$\frac{\texttt{Unconstrained\_Noisy\_Cost} - \texttt{Constrained\_Noisy\_Cost}}{\texttt{OPT}}$$

The improvements obtained due to the constrained version are presented in Table A.1.

Some observations regarding the structural constraints are the following:

- When we add triplet constraints to the input as advice for the algorithm, it is crucial for the triplet constraints to actually be useful. "Easy" constraints that are readily implied by the similarity scores will have no extra use and will not lead to better solutions.
- We also observed that having "nested" constraints can be really useful. Nested constraints can guide our algorithm to perform good cuts as they refer to a larger portion of the optimum tree T^* (i.e. contiguous subtrees) rather than just different unrelated subtrees of it. The usefulness of the given constraints is correlated with the depth of the nested constraints and their accordance with the optimum tree T^* based on Dasgupta's objective.
- Furthermore, since most of the objective cost comes from the large initial clusters, we focused on the partitions that created large clusters and imposed triplet constraints that ensured good

cuts in the beginning. Actually in some cases, just the first 3 or 4 cuts are enough to guarantee that we get $\approx 12\%$ improvement.

- Finally, we conclude that just the number of the given triplet constraints may not constitute a good metric for their usefulness. For example, a large number of constraints referring to wildly different parts of T^*, may end up being much less useful than a smaller number of constraints guiding towards a good first cut.

A.3 Deferred Proofs of Section 4.2.1

Proof. [Proof of Theorem 4.2.1]

The bottom-up merging strategy that first merges the edges e inside the $K_{n^{2/3}}$ cliques attains HC value close to $n\mathcal{W}$ (the rest of the edges don't matter). Since `OPT` is only better than this merging strategy, the claim follows. To see that, note that all such edges e will have a multiplier of `non-leaves` $\geq n - n^{2/3}$. Since there are $\frac{1}{2}n^{2/3} \cdot (n^{2/3}-1) \cdot n^{1/3}$ such edges, `OPT` $\geq (n - n^{2/3}) \cdot \frac{1}{2}n^{2/3} \cdot (n^{2/3}-1) \cdot n^{1/3} \geq \frac{1}{2}n^{8/3} - O(n^{7/3})$.
∎

Proof. [Proof of Theorem 4.2.2] Because of the $1+\epsilon$ weight of the edges going across the $K_{n^{2/3}}$ cliques, $Average - Linkage$ will first start merging the $K_{n^{1/3}}$ cliques consisting of one node out of each $K_{n^{2/3}}$ clique. There are $n^{2/3}$ such $K_{n^{1/3}}$ cliques so the total objective contribution of the edges involved in this first phase is insignificant since it is certainly smaller than $n \cdot \frac{1}{2}n^{1/3} \cdot (n^{1/3}-1) \cdot n^{2/3} \cdot (1+\epsilon) = O(n^{7/3})$. Observe that after the first phase of $Average - Linkage$ the remaining subclusters to be merged form a clique on $n^{2/3}$ supernodes each with size $s = n^{1/3}$ and weighted edges with uniform weights $n^{1/3}$. By Theorem 4.2.1 and taking into account the size of every supernode, we obtain the final value for $Average - Linkage$ $= \frac{1}{3}n^{2/3} \cdot \binom{n^{2/3}}{2} \cdot w \cdot s = \frac{1}{3}n^{2/3} \cdot \frac{1}{2}n^{2/3} \cdot (n^{2/3}-1) \cdot n^{1/3} \cdot n^{1/3} \leq \frac{1}{6}n^{8/3} + O(n^{7/3})$. ∎

Proof. [Proof of Theorem 4.2.3] The `OPT` solution can get all the weight by performing the cut (L, R) and then proceed arbitrarily. The HC value is then `OPT` $= n\mathcal{W} = \frac{1}{4}n^3 - O(n^2)$. ∎

Proof. [Proof of Theorem 4.2.4] Since there are a lot of 0 weight edges in the graph, $Average - Linkage$ first tries to merge endpoints of such edges. Note that $Average - Linkage$ is underspecified since there are ties here, but these ties are not affecting the overall outcome as we can break ties arbitrarily by using small edge weights $\epsilon > 0$. Hence, we can assume that $Average - Linkage$ first merged the two endpoints of edges in the perfect matching M. After this first step, the remaining subclusters to be merged form a clique on $\frac{n}{2}$ supernodes each with size $s = 2$ and weighted edges with uniform weights $w = 2$. By using the Theorem 4.2.1 and taking into account the size of every supernode, we obtain the final value for $Average - Linkage$ $= \frac{2}{3}\frac{n}{2} \cdot \binom{n/2}{2} \cdot w \cdot s = \frac{2}{3}\frac{n}{2} \cdot \frac{1}{2} \cdot \frac{n}{2} \cdot (\frac{n}{2}-1) \cdot 2 \cdot 2 \leq \frac{1}{6}n^3$. ∎

A.4 Deferred Proofs of Section 4.2.3

Proof. [Proof of Theorem 4.2.6] First of all, the optimization $\mathcal{P}_{\text{lower-bound}}$ decomposes over variables $\{\theta_{ik}\}_{k\neq i}$ and variables $\{\theta_{jk}\}_{j\neq k}$. Due to symmetry, we only lower-bound the optimal objective value of the following minimization program,:

$$\begin{aligned} \textit{minimize} \quad & \sum_{k\neq i,j} \theta_{ik} \\ \textit{subject to} \quad & \sum_{k\neq i} \cos(\theta_{ik}) \leq n/2-1, \\ & 0 \leq \theta_{ik} \leq \frac{\pi}{2}, \qquad \forall k, \end{aligned} \tag{P-1}$$

and then use $\mathtt{OBJ}(\mathcal{P}_{\text{lower-bound}})=\frac{1}{2\pi}\big(2\,\mathtt{OBJ}(\text{P-1})-(n-2)\bar{\theta}\big)$. Suppose $\{\theta^*_{ik}\}_{k\neq i}$ is the optimal solution of the above program (P-1). We first claim that in any optimal solution the first constraint is tight, i.e. $\sum_{k\neq i}\cos(\theta^*_{ik}) = n/2-1$. This simply holds because otherwise one can slightly decrease one of the non-zero θ^*_{ik} and strictly decrease the objective, a contradiction. Next, we claim that $\theta^*_{ik} \in \{0, \frac{\pi}{2}\}$ for all $k \neq i$, except for at most one $k = k_0$. To prove by contradiction, suppose it is not true. Therefore, there exist $k_1, k_2 \neq i$ such that $0 < \theta^*_{ik_1} \leq \theta^*_{ik_2} < \pi/2$. If we decrease $\theta^*_{ik_1}$ by infinitesimal d_θ and increase $\theta^*_{ik_2}$ by the same d_θ, then the objective value does not change. However, because of the concavity of the cosine function over the interval $[0, \pi/2]$, there will be an additional slack in the first constraint of P-1, a contradiction to the first claim that in any optimal solution this constraint is tight.

Because $\theta^*_{ik} \in \{0, \pi/2\}$ for $k \neq i, k_0$, we have:

$$\#\{k \neq i, k_0 : \theta^*_{ik} = 0\} = \sum_{k\neq i,k_0} \cos(\theta^*_{i,k}) \leq \sum_{k\neq i} \cos(\theta^*_{i,k}) \leq n/2-1$$

We then conclude that for at least $n-2-(n/2-1) = \frac{n-2}{2}$ values of $k \neq i, k_0$ we have $\theta^*_{ik} = \pi/2$, and hence the optimal objective value of P-1 is lower-bounded by $\frac{n-2}{2}\cdot\pi/2 = \frac{(n-2)\pi}{4}$. This lower-bound immediately implies that the optimal objective value of $\mathcal{P}_{\text{lower-bound}}$ is also lower-bounded by

$$\frac{2\cdot\frac{(n-2)\pi}{4} - (n-2)\bar{\theta}}{2\pi} = (n-2)\left(\frac{1}{4} - \frac{\bar{\theta}}{2\pi}\right),$$

which completes the proof of the lemma. ▮

Proof. [Details of final calculations in the proof of Theorem 4.2.5] To get the final approximation factor, we balanced out the two cases:

$$\left(1-\frac{2\epsilon_1}{\epsilon_2}\right)\left(\frac{1}{2} - \frac{2\cos^{-1}(1-\epsilon_2)}{3\pi}\right) = \frac{1}{3(1-\epsilon_1)} \tag{A.1}$$

By solving for ϵ_1, the optimal value of ϵ_1 as a function of ϵ_2 is calculated to be the following function:

$$\epsilon_1^*(\epsilon_2) = \frac{1}{4} \cdot \left((\epsilon_2 + 2) - \sqrt{(\epsilon_2+2)^2 - 8\epsilon_2 \left(1 - \frac{1}{3 \cdot (\frac{1}{2} - \frac{2 \arccos 1 - \epsilon_2}{3\pi})} \right)} \right)$$

we then draw $\alpha(\epsilon_2) = \frac{1}{3(1-\epsilon_1^*(\epsilon_2))}$ for $\epsilon_2 \in [0,1]$ by the aid of a computer software (`WolframAlpha`). This function peaks at around $\epsilon_2 \approx 0.139$. By plugging this number into $\alpha(\epsilon_2)$, we get the final factor. ∎

A.5 Deferred Proofs of Section 4.2.4

Proof. [Proof of Theorem 4.2.9] Recall that in the partition of T^*, we have $\max(|L_k|, |R_k|) < n(1-\delta)$. Since both pieces L_k, R_k have sizes at most $(1-\delta)n$, edges cut within the subtrees rooted at L_k and at R_k can only have contribution to OPT at most $n(1-\delta)w(L_k) + n(1-\delta)w(R_k)$. Hence:

$$\texttt{OPT} \le n(1-\delta)w(L_k) + n(1-\delta)w(R_k) + n(\mathcal{W} - w(L_k) - w(R_k))$$

Combining with our assumption $\texttt{OPT} \ge (1-\epsilon)n\mathcal{W}$, we get:

$$(1-\epsilon)n\mathcal{W} \le n(1-\delta)w(L_k) + n(1-\delta)w(R_k) + n(\mathcal{W} - w(L_k) - w(R_k))$$

and the claim follows by rearranging the terms after the cancelations. ∎

Proof. [Proof of Theorem 4.2.13] We start by $\texttt{ALG} = \texttt{ALG}_{\text{peel}} + \texttt{ALG}_{\text{cut}}$. From Theorem 4.2.7 and Lemma 4.2.10 we have:

$$\texttt{ALG}_{\text{peel}} \ge (1 - \tfrac{2\mathcal{W}}{n\tau})\texttt{OPT}_{\text{red}}$$

$$\texttt{ALG}_{\text{cut}} \ge \rho_{\text{GW}} \left(n - \tfrac{2\mathcal{W}}{\tau}\right) MaxCut_{blue} \ge \rho_{\text{GW}} \left(1 - \tfrac{2\mathcal{W}}{n\tau}\right) \left(\texttt{OPT}_{\text{blue-cut}} - nw(L_k) - nw(R_k) + \tfrac{\texttt{OPT}_{\text{blue-chain}}}{2}\right)$$

Hence,

$$\texttt{ALG} \ge \rho_{\text{GW}} \left(1 - \tfrac{2\mathcal{W}}{n\tau}\right) \left(\texttt{OPT}_{\text{red}} + \texttt{OPT}_{\text{blue-cut}} + \texttt{OPT}_{\text{blue-chain}} - nw(L_k) - nw(R_k) - \tfrac{\texttt{OPT}_{\text{blue-chain}}}{2}\right)$$

Because $\texttt{OPT} \ge (1-\epsilon)n\mathcal{W} \implies n\mathcal{W} \le \frac{\texttt{OPT}}{1-\epsilon}$, by Claim 4.2.9 and Theorem 4.2.8 we get ($\delta\tau n^2 = 2\delta\gamma n\mathcal{W}$):

$$\texttt{ALG} \ge \rho_{\text{GW}} \left(1 - \tfrac{1}{\gamma}\right) \left(1 - \tfrac{\epsilon}{\delta(1-\epsilon)} - \tfrac{\delta\gamma}{1-\epsilon}\right) \texttt{OPT}$$

We have to balance out the two factors obtained from **Case 1** and **Case 2**, so we get the final equation:

$$\rho_{\text{GW}} \left(1 - \tfrac{1}{\gamma}\right) \left(1 - \tfrac{\epsilon/\delta}{1-\epsilon} - \tfrac{\delta\gamma}{1-\epsilon}\right) = \tfrac{2}{3(1-\epsilon)}$$

In terms of the parameter δ, it's easy to see that the choice of $\delta = \sqrt{\frac{\epsilon}{\gamma}}$ is optimal, so substituting:

$$\rho_{\mathrm{GW}} \left(1 - \tfrac{1}{\gamma}\right) \left(1 - \tfrac{2\sqrt{\epsilon\gamma}}{1-\epsilon}\right) = \tfrac{2}{3(1-\epsilon)}$$

Rearranging the terms we get:

$$\rho_{\mathrm{GW}} \left(1 - \tfrac{1}{\gamma}\right) \epsilon + 2\rho_{\mathrm{GW}} \left(1 - \tfrac{1}{\gamma}\right) \sqrt{\gamma} \cdot \sqrt{\epsilon} + \tfrac{2}{3} - \rho_{\mathrm{GW}} \left(1 - \tfrac{1}{\gamma}\right) = 0$$

Maximizing over γ for $\sqrt{\epsilon} \geq 0$ (dropping the negative solution):

$$\sqrt{\epsilon} = \frac{-2\sqrt{\gamma} + \sqrt{4\gamma - 4\left(\frac{2}{3}\frac{1}{\rho_{\mathrm{GW}}}\frac{\gamma}{\gamma-1} - 1\right)}}{2}$$

we get the final optimal answer for $\gamma \approx 11.1$ and $\epsilon \approx 0.000612$.

▮

Bibliography

[1] Amir Abboud, Vincent Cohen-Addad, and Hussein Houdrougé. Subquadratic high-dimensional hierarchical clustering. In *Advances in Neural Information Processing Systems*, pages 11576–11586, 2019.

[2] Sara Ahmadian, Vaggos Chatziafratis, Alessandro Epasto, Euiwoong Lee, Mohammad Mahdian, Konstantin Makarychev, and Grigory Yaroslavtsev. Bisect and conquer: Hierarchical clustering via max-uncut bisection. *The 23rd International Conference on Artificial Intelligence and Statistics*, 2020.

[3] Alfred V. Aho, Yehoshua Sagiv, Thomas G. Szymanski, and Jeffrey D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

[4] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):1–27, 2008.

[5] Noga Alon, Yossi Azar, and Danny Vainstein. Hierarchical clustering: a 0.585 revenue approximation. 2020.

[6] Christoph Ambühl, Monaldo Mastrolilli, and Ola Svensson. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM Journal on Computing*, 40(2):567–596, 2011.

[7] Haris Angelidakis, Pranjal Awasthi, Avrim Blum, Vaggos Chatziafratis, and Chen Dan. Bilu-linial stability, certified algorithms and the independent set problem. *arXiv preprint arXiv:1810.08414*, 2018.

[8] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2):1–37, 2009.

[9] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Geometry, flows, and graph-partitioning algorithms. *Commun. ACM*, 51(10):96–105, 2008.

[10] Per Austrin, Siavosh Benabbas, and Konstantinos Georgiou. Better balance by being biased: A 0.8776-approximation for max bisection. *ACM Transactions on Algorithms (TALG)*, 13(1):2, 2016.

[11] Per Austrin, Rajsekar Manokaran, and Cenny Wenner. On the NP-hardness of approximating ordering constraint satisfaction problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 26–41. Springer, 2013.

[12] Pranjal Awasthi, Maria Florina Balcan, and Konstantin Voevodski. Local algorithms for interactive clustering. *The Journal of Machine Learning Research*, 18(1):75–109, 2017.

[13] Maria-Florina Balcan and Avrim Blum. Clustering with interactive feedback. In *International Conference on Algorithmic Learning Theory*, pages 316–328. Springer, 2008.

[14] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. A discriminative framework for clustering via similarity functions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 671–680, 2008.

[15] Maria Florina Balcan and Yingyu Liang. Clustering under perturbation resilience. In *International Colloquium on Automata, Languages, and Programming*, pages 63–74. Springer, 2012.

[16] Maria-Florina Balcan, Yingyu Liang, and Pramod Gupta. Robust hierarchical clustering. *Journal of Machine Learning Research*, 15:3831, 2014.

[17] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[18] Yair Bartal. Graph decomposition lemmas and their role in metric embedding methods. In *European Symposium on Algorithms*, pages 89–97. Springer, 2004.

[19] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[20] Mikhail Bilenko, Sugato Basu, and Raymond J Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 11. ACM, 2004.

[21] Yonatan Bilu and Nathan Linial. Are stable instances easy? *Combinatorics, Probability and Computing*, 21(05):643–660, 2012.

[22] Gerth Stølting Brodal, Rolf Fagerberg, Thomas Mailund, Christian NS Pedersen, and Andreas Sand. Efficient algorithms for computing the triplet and quartet distance between trees of arbitrary degree. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1814–1832. Society for Industrial and Applied Mathematics, 2013.

[23] Jaroslaw Byrka, Sylvain Guillemot, and Jesper Jansson. New results on optimizing rooted triplets consistency. *Discrete Applied Mathematics*, 158(11):1136–1147, 2010.

[24] Luigi L Cavalli-Sforza and Anthony WF Edwards. Phylogenetic analysis: models and estimation procedures. *Evolution*, 21(3):550–570, 1967.

[25] Arthur Cayley. Xxviii. on the theory of the analytical forms called trees. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 13(85):172–176, 1857.

[26] Arthur Cayley. A theorem on trees. *Quarterly J. Math*, 23:376–378, 1889.

[27] Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 841–854. SIAM, 2017.

[28] Moses Charikar, Vaggos Chatziafratis, and Rad Niazadeh. Hierarchical clustering better than average-linkage. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2291–2304. SIAM, 2019.

[29] Moses Charikar, Vaggos Chatziafratis, Rad Niazadeh, and Grigory Yaroslavtsev. Hierarchical clustering for euclidean data. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2721–2730, 2019.

[30] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.

[31] Moses Charikar, Mohammad Taghi Hajiaghayi, Howard Karloff, and Satish Rao. l_2^2 spreading metrics for vertex ordering problems. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1018–1027. Society for Industrial and Applied Mathematics, 2006.

[32] Evangelos Chatziafratis, Yingrui Zhang, and Osman Yağan. On the robustness of power systems: optimal load-capacity distributions and hardness of attacking. In *2016 Information Theory and Applications Workshop (ITA)*, pages 1–10. IEEE.

[33] Vaggos Chatziafratis, Neha Gupta, and Euiwoong Lee. Hardness for dissimilarity hierarchical clustering and fair correlation clustering. *work in progress*, 2020.

[34] Vaggos Chatziafratis, Mohammad Mahdian, and Sara Ahmadian. Aggregating inconsistent information in ranking, clustering and phylogenetic trees. In *manuscript under submission*, 2020.

[35] Vaggos Chatziafratis, Rad Niazadeh, and Moses Charikar. Hierarchical clustering with structural constraints. In *International Conference on Machine Learning*, pages 774–783, 2018.

[36] Vaggos Chatziafratis, Tim Roughgarden, and Jan Vondrák. Stability and recovery for independence systems. *European Symposium on Algorithms (ESA)*, 2017.

[37] Andrew Chester, Riccardo Dondi, and Anthony Wirth. Resolving rooted triplet inconsistency by dissolving multigraphs. In *International Conference on Theory and Applications of Models of Computation*, pages 260–271. Springer, 2013.

[38] Vincent Cohen-Addad, Varun Kanade, and Frederik Mallmann-Trenn. Hierarchical clustering beyond the worst-case. In *Advances in Neural Information Processing Systems*, pages 6202–6210, 2017.

[39] Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. *Journal of the ACM (JACM)*, 66(4):1–42, 2019.

[40] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[41] Katharina Dannenberg, Jesper Jansson, Andrzej Lingas, and Eva-Marta Lundell. The approximability of maximum rooted triplets consistency with fan triplets and forbidden triplets. *Discrete Applied Mathematics*, 257:101–114, 2019.

[42] Sanjoy Dasgupta. Performance guarantees for hierarchical clustering. In *International Conference on Computational Learning Theory*, pages 351–363. Springer, 2002.

[43] Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2016, pages 118–127, New York, NY, USA, 2016. ACM.

[44] Nikhil R Devanur, Subhash A Khot, Rishi Saket, and Nisheeth K Vishnoi. Integrality gaps for sparsest cut and minimum linear arrangement problems. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 537–546. ACM, 2006.

[45] Ibai Diez, Paolo Bonifazi, Iñaki Escudero, Beatriz Mateos, Miguel A Muñoz, Sebastiano Stramaglia, and Jesus M Cortes. A novel brain partition highlights the modular skeleton shared by structure and function. *Scientific reports*, 5:10532, 2015.

[46] Michael B Eisen, Paul T Spellman, Patrick O Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998.

[47] Ehsan Emamjomeh-Zadeh and David Kempe. Adaptive hierarchical clustering using ordinal queries. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 415–429. SIAM, 2018.

[48] Guy Even, Joseph Seffi Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *Journal of the ACM (JACM)*, 47(4):585–616, 2000.

[49] Uriel Feige and James R Lee. An improved approximation ratio for the minimum linear arrangement problem. *Information Processing Letters*, 101(1):26–29, 2007.

[50] Joseph Felsenstein. *Inferring phylogenies*, volume 2. Sinauer associates Sunderland, MA, 2004.

[51] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[52] Xiping Fu, Brendan McCane, Steven Mills, and Michael Albert. Nokmeans: Non-orthogonal k-means hashing. In *Asian Conference on Computer Vision*, pages 162–177. Springer, 2014.

[53] Naveen Garg, Vijay V Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi) cut theorems and their applications. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 698–707. ACM, 1993.

[54] Thomas Gärtner. A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5(1):49–58, 2003.

[55] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[56] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[57] Talha Cihad Gulcu, Vaggos Chatziafratis, Yingrui Zhang, and Osman Yağan. Attack vulnerability of power systems under an equal load redistribution model. *IEEE/ACM Transactions on Networking*, 26(3):1306–1319, 2018.

[58] Venkatesan Guruswami, Johan Håstad, Rajsekar Manokaran, Prasad Raghavendra, and Moses Charikar. Beating the random ordering is hard: Every ordering csp is approximation resistant. *SIAM Journal on Computing*, 40(3):878–914, 2011.

[59] Venkatesan Guruswami, Rajsekar Manokaran, and Prasad Raghavendra. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 573–582. IEEE, 2008.

[60] Jesper Jansson. *Consensus algorithms for trees and strings*. 2003.

[61] N Jardine and R Sibson. A model for taxonomy. *Mathematical Biosciences*, 2(3-4):465–482, 1968.

[62] Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[63] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[64] Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 217–226. Society for Industrial and Applied Mathematics, 2014.

[65] 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*, pages 767–775. ACM, 2002.

[66] Matthäus Kleindessner and Ulrike von Luxburg. Kernel functions based on triplet comparisons. In *Advances in Neural Information Processing Systems*, pages 6810–6820, 2017.

[67] Robert Krauthgamer, Joseph Seffi Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 942–949. Society for Industrial and Applied Mathematics, 2009.

[68] Tom Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 422–431. IEEE, 1988.

[69] 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, 1999.

[70] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets.* Cambridge university press, 2014.

[71] Moshe Lichman. UCI machine learning repository, zoo dataset, 2013.

[72] Guolong Lin, Chandrashekhar Nagarajan, Rajmohan Rajaraman, and David P Williamson. A general approach for incremental approximation and hierarchical clustering. *SIAM Journal on Computing*, 39(8):3633–3669, 2010.

[73] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[74] Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Bilu-linial stable instances of max cut and minimum multiway cut. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 890–906, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics.

[75] Charles F Mann, David W Matula, and Eli V Olinick. The use of sparsest cuts to reveal the hierarchical community structure of social networks. *Social Networks*, 30(3):223–234, 2008.

[76] Frank McSherry. Spectral partitioning of random graphs. In *focs*, page 529. IEEE, 2001.

[77] Nicholas Monath, Manzil Zaheer, Daniel Silva, Andrew McCallum, and Amr Ahmed. Gradient-based hierarchical clustering using continuous representations of trees in hyperbolic space. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 714–722, 2019.

[78] Benjamin Moseley and Joshua Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *Advances in Neural Information Processing Systems*, pages 3097–3106, 2017.

[79] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems*, pages 6338–6347, 2017.

[80] Richard Peng. Approximate undirected maximum flows in o (m polylog (n)) time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1862–1867. SIAM, 2016.

[81] C Greg Plaxton. Approximation algorithms for hierarchical location problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 40–49. ACM, 2003.

[82] Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 755–764. ACM, 2010.

[83] Prasad Raghavendra, David Steurer, and Madhur Tulsiani. Reductions between expansion problems. In *2012 IEEE 27th Conference on Computational Complexity*, pages 64–73. IEEE, 2012.

[84] F James Rohlf. Adaptive hierarchical clustering schemes. *Systematic Biology*, 19(1):58–82, 1970.

[85] Aurko Roy and Sebastian Pokutta. Hierarchical clustering via spreading metrics. In *Neural Information Processing Systems (NIPS)*, pages 2316–2324, 2016.

[86] Jonah Sherman. Breaking the multicommodity flow barrier for o (vlog n)-approximations to sparsest cut. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 363–372. IEEE, 2009.

[87] Jonah Sherman. Nearly maximum flows in nearly linear time. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 263–269. IEEE, 2013.

[88] Peter HA Sneath and Robert R Sokal. Numerical taxonomy. *Nature*, 193(4818):855–860, 1962.

[89] Sagi Snir and Satish Rao. Using max cut to enhance rooted trees consistency. *IEEE/ACM transactions on computational biology and bioinformatics*, 3(4):323–333, 2006.

[90] Michael Steinbach, George Karypis, and Vipin Kumar. A comparison of document clustering techniques. In *TextMining Workshop at KDD2000 (May 2000)*, 2000.

[91] 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.

[92] Omer Tamuz, Ce Liu, Serge Belongie, Ohad Shamir, and Adam Tauman Kalai. Adaptively learning the crowd kernel. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 673–680. Omnipress, 2011.

[93] Madhur Tulsiani, 2019. Personal Communication.

[94] Michele Tumminello, Fabrizio Lillo, and Rosario N Mantegna. Correlation, hierarchies, and networks in financial markets. *Journal of economic behavior & organization*, 75(1):40–58, 2010.

[95] Andrea Vedaldi and Brian Fulkerson. Vlfeat: An open and portable library of computer vision algorithms. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1469–1472. ACM, 2010.

[96] Sharad Vikram and Sanjoy Dasgupta. Interactive Bayesian hierarchical clustering. In *International Conference on Machine Learning*, pages 2081–2090, 2016.

[97] Kiri Wagstaff and Claire Cardie. Clustering with instance-level constraints. *AAAI/IAAI*, 1097:577–584, 2000.

[98] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *ICML*, volume 1, pages 577–584, 2001.

[99] Chenchen Wu, Donglei Du, and Dachuan Xu. An improved semidefinite programming hierarchies rounding approximation algorithm for maximum graph bisection problems. *Journal of Combinatorial Optimization*, 29(1):53–66, 2015.

[100] Grigory Yaroslavtsev and Adithya Vadapalli. Massively parallel algorithms and hardness for single-linkage clustering under ℓ_p-distances. *arXiv preprint arXiv:1710.01431*, 2017.

[101] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In *Advances in neural information processing systems*, pages 1601–1608, 2005.