



PhD Thesis

# Clustering, Weighted Sequences, and Shortest Paths

Evangelos Kipouridis

Supervisor

Mikkel Thorup


Co-Supervisor


Christian Wulff-Nilsen

This thesis has been submitted to the PhD School of The Faculty of Science, University of Copenhagen.

Submission date: March 30, 2022.

## Funding

This research is supported by Thorup's Investigator Grant from the Villum Foundation under Grant No. 16582, Basic Algorithms Research Copenhagen (BARC), Denmark. 

Evangelos Kipouridis has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 801199. 

## Abstract

In this thesis we focus on clustering problems where the input is the ideal relationship between all pairs of objects in the final clustering. More particularly, we concern ourselves with the following problems.

*$L_1$ -fitting tree metrics and ultrametrics:* We are given the ideal distance between all pairs of  $n$  objects, and the goal is to output a weighted tree (resp. ultrametric) which spans the set of objects and minimizes the sum of pairwise distance errors. Both problems are closely related to evolutionary biology and the reconstruction of the tree of life. In fact, discussions related to the reconstruction of the optimal tree are traced back to Plato and Aristotle (350 BC), in the context of classification. Both problems were known to be APX-Hard and the best known approximation factor was  $O((\log n)(\log \log n))$  by Ailon and Charikar [FOCS '05]. We design asymptotically optimal constant factor approximations for both problems. Our paper “Fitting Distances by Tree Metrics Minimizing the Total Error within a Constant Factor” appeared in FOCS '21.

*Constrained Correlation Clustering:* For each pair of objects, we are given a preference related to whether the two objects should be in the same cluster or not. Furthermore, we are also given hard constraints for certain pairs. The output clustering must satisfy all hard constraint, and minimize the number of violated preferences. We design a deterministic combinatorial algorithm with a constant approximation factor. A key ingredient in our approach is a novel nearly-optimal pivoting algorithm for Correlation Clustering. This is a deterministic combinatorial algorithm achieving the best approximation factor among all known deterministic combinatorial algorithms for Correlation Clustering, not just pivoting ones. Part of these results have been submitted to ICALP '22.

Apart from clustering, we also study graph and string problems.

*Multiple Source Shortest Paths in Planar Digraphs:* Given an embedded planar digraph with positive edge weights and a face  $f$ , we are interested in a data structure supporting shortest path queries, where the source is in  $f$ . The best known data structure by Klein [SODA '05] requires  $O(n \log n)$  time for preprocessing and  $O(\log n)$  time for queries, where  $n$  is the number of nodes. We improve the preprocessing/query time to  $O(n \log |f|)/O(\log |f|)$ , where  $|f|$  is the number of nodes in  $f$ . More importantly, our approach is much simpler, requiring only single source shortest path computations and contractions. In contrast, Klein’s solution required persistency, dynamic trees, and an interplay between the primal and the dual graph. Our paper “A Simple Algorithm for Multiple-Source Shortest Paths in Planar Digraphs” appeared in SOSA '22.

*Longest Common Subsequence on Weighted Sequences:* Weighted sequences generalize the concept of strings, so that in each position we have a probability distribution over the alphabet, rather than a single character. The motivation comes from the inherent uncertainty of the actual methods used for “reading” a DNA sequence. We suggest that the alphabet size is a crucial parameter for this problem, and provide optimal results both in the case of bounded and unbounded alphabets. Furthermore, this is the first work on Weighted Sequences avoiding the Log-Probability model, a simplifying assumption related to exact computations of reals. Our paper “Longest Common Subsequence on Weighted Sequences” received the Best Paper Award at CPM '20.

## Resumé

Denne afhandling omhandler klyngedannelsesproblemer, hvori input er givet som en række egenskaber der beskriver ideelle forhold mellem alle par af objekter der indgår i en endelig klyngedannelse. Mere specifikt beskæftiger vi os med følgende problemer:

*$L_1$ -passende træmetrikker og ultrametrikker:* Vi er givet den ideelle afstand mellem alle par af  $n$  objekter, og målet er at beregne et vægtet (hhv. ultrametrisk) træ, der udspænder mængden af objekter og minimerer summen af parvise afstandsfejl. Begge problemer er motiveret af deres nære kobling til evolutionær biologi og genopbygningen af livets træ. Diskussioner vedrørende rekonstruktionen af det optimale træ daterer helt tilbage til Platon og Aristoteles (350 f.Kr.), i forbindelse med klassifikation. Det er velkendt at begge problemer er APX-hårde, og den hidtil bedst kendte tilnærmelsesfaktor var  $O((\log n)(\log \log n))$  af Ailon og Charikar [FOCS '05]. Vi beskriver asymptotisk optimale konstantfaktorapproximationer for begge problemer. Vores artikel "Fitting Distances by Tree Metrics Minimizing the Total Error within a Constant Factor" udkom i FOCS '21.

*Korreleret Klyngedannelse med Begrænsninger:* For ethvert par af objekter er vi givet en præference, der angiver hvorvidt det ønskes at to objekter skal indgå i samme klynge. Derudover er vi også givet hårde begrænsninger, der angiver hvilke par, der ikke må indgå i den samme klynge. Det er et krav at den resulterende klynge skal opfylde samtlige hårde begrænsninger imens antallet af uopfyldte præferencer samtidig minimeres. Til dette problem beskriver vi en deterministisk, kombinatorisk algoritme med en konstant tilnærmelsesfaktor. En nøglekomponent i vores tilgang er en ny, næroptimal, pivotbaseret algoritme til beregning af korrelerede klynger uden begrænsninger. Denne er en deterministisk, kombinatorisk algoritme, der opnår den bedste tilnærmelsesfaktor blandt alle kendte deterministiske, kombinatoriske algoritmer til dette problem, og altså dermed ikke kun de pivotbaserede. En delmængde af disse resultater er blevet indsendt til ICALP '22.

Udover klyngeproblemer studerer vi også graf- og strengproblemer.

*Korteste Veje fra flere Knuder i Plane Grafer:* Givet en indlejret, plan, orienteret graf med positive kantvægte og en flade  $f$ , er vi interesserede i en datastruktur, der understøtter kortestvejsforespørgsler af veje der har en knude i  $f$  som endepunkt. Den mest velkendte datastruktur til dette er beskrevet af Klein [SODA '05] og kræver  $O(n \log n)$  forbehandlingstid og  $O(\log n)$  forespørgselstid, hvor  $n$  er antallet af knuder. Vi forbedrer forbehandlings-, hhv. forespørgselstiden til  $O(n \log |f|)$  hhv.  $O(\log |f|)$ , hvor  $|f|$  angiver antallet af knuder i  $f$ . Vigtigere endnu er at vores tilgang er langt simpleere sammenlignet med Kleins idet den udelukkende beror på beregninger af korteste veje fra én enkelt knude samt kantsammentrækninger. I modsætning hertil anvender Kleins tilgang vedvarende datastrukturer, dynamiske træer og et samspil mellem den primære og den duale graf. Vores artikel "A Simple Algorithm for Multiple-Source Shortest Paths in Planar Digraphs" udkom i SOSA '22.

*Længste Fælles Delfølger af Vægtede Følger:* Vægtede følger generaliserer strenge, således at vi i for enhver indgang er givet en sandsynlighedsfordeling over alfabetet i stedet for et enkelt tegn. Motivationen kommer fra den iboende usikkerhed ved gængse metoder, der i praksis anvendes til at læse en DNA-sekvens. Vi foreslår, at størrelsen på alfabetet kan anvendes som en afgørende parameter, der beskriver kompleksiteten af problemet. Vi giver optimale resultater både i tilfældet af afgrænsede og ubegrænsede alfabeter. Ydermere er dette det første arbejde med vægtede sekvenser, der omgår log-sandsynlighedsmodellen, en forsimplende antagelse relateret til nøjagtige beregninger af reelle værdier. Vores artikel "Longest Common Subsequence on Weighted Sequences" blev tildelt prisen for bedste artikel ved CPM '20.

# Preface

The *General rules and guidelines for the PhD programme* at the Faculty of Science, University of Copenhagen allows for a PhD dissertation to be written “*as a synopsis with manuscripts of papers or already published papers attached*”. The present dissertation has this form.

Throughout my PhD, I have written 4 published papers and 3 unpublished manuscripts. Out of these, I include a paper and a manuscript related to clustering, as this was the main focus of my PhD. I also include 2 papers related to graphs and strings, to demonstrate the breadth of my PhD project. For completeness, I provide a very brief synopsis of all 7 results in this preface.

## Included in the thesis

**Clustering:** Clustering is a fundamental task related to unsupervised learning, with many applications in machine learning and data mining. The goal of clustering is to partition a set of objects into disjoint clusters, such that (ideally) all objects within a cluster are similar, and objects in different clusters are dissimilar. As no single definition best captures this high-level goal, a lot of different clustering objectives have been suggested. In this thesis we focus on clustering problems where the input is the ideal relationship between all pairs of objects in the final clustering. More particularly, we concern ourselves with the following problems.

*$L_1$ -fitting tree metrics and ultrametrics:* We are given the ideal distance between all pairs of  $n$  objects, and the goal is to output a weighted tree which spans the set of objects and respects the input distances as much as possible. More formally, for each pair of objects we have an error, the absolute difference between their distance in the input and their distance in the tree. The goal is to minimize the sum of errors. A closely related problem asks for the output tree to be a rooted tree whose leaves are the input objects and they are all in the same depth (ultrametric). An ultrametric naturally induces a hierarchical clustering of the objects. These problems closely relate to evolutionary biology and the reconstruction of the tree of life. In fact, discussions related to the reconstruction of the optimal tree are traced back to Plato and Aristotle (350 BC), in the context of classification. Both problems were known to be APX-Hard and the best known approximation factor was  $O((\log n)(\log \log n))$  by Ailon and Charikar [AC11] (FOCS '05). We design asymptotically optimal constant factor approximations for both problems. Our paper “Fitting Distances by Tree Metrics Minimizing the Total Error within a Constant Factor” [CDK<sup>+</sup>21] appeared in FOCS '21.

*Constrained Correlation Clustering:* For each pair of objects, we are given a preference related to whether the two objects should be in the same cluster or not. Furthermore, for certain pairs we are given a hard constraint related to whether the two objects must be in the same cluster or not. The goal is to provide a partition (clustering) of the objects so that no hard constraint is violated, and the number of violated preferences is minimized. Van Zuylen et al. [vZW09] (SODA '07) solved this APX-Hard problem within a 3 approximation factor, using a deterministic algorithm; even though in their paper they are mostly interested in deterministic combinatorial

algorithms, their solution to this problem requires solving an LP. We design a deterministic combinatorial algorithm with a constant approximation factor. A key ingredient in our approach is a novel nearly-optimal pivoting algorithm for Correlation Clustering (the version where no hard constraints are given). This is a deterministic combinatorial algorithm achieving the best approximation factor among all known deterministic combinatorial algorithms for Correlation Clustering, not just pivoting ones. Part of these results have been submitted to ICALP '22.

**Planar Graphs:** We study the Multiple Source Shortest Paths problem in Planar Digraphs. Given an embedded planar digraph with positive edge weights and a face  $f$ , we are interested in a data structure supporting shortest path queries, where the source is in  $f$ . The best known solution by Klein [Kle05] (SODA '05) requires  $O(n \log n)$  time for preprocessing and  $O(\log n)$  time for queries, where  $n$  is the number of nodes. We design a data structure with  $O(n \log |f|)/O(\log |f|)$  preprocessing/query time, where  $|f|$  is the number of nodes in  $f$ . More importantly, our approach is much simpler, requiring only single source shortest path computations and contractions. In contrast, Klein's solution required persistency, dynamic trees, and an interplay between the primal and the dual graph. Our paper "A Simple Algorithm for Multiple-Source Shortest Paths in Planar Digraphs" [DKGW22] appeared in SOSA '22.

**Strings:** In this work we study the Longest Common Subsequence problem on Weighted Sequences. Weighted sequences generalize the concept of strings, so that in each position we have a probability distribution over the alphabet, rather than a single character. The motivation comes from the inherent uncertainty of the actual methods used for "reading" a DNA sequence. We suggest that the alphabet size is a crucial parameter for this problem, and provide optimal results both in the case of bounded and unbounded alphabets. In the natural case of a bounded alphabet, we design an EPTAS while no FPTAS is possible. When the alphabet is of unbounded size, we prove that no EPTAS is possible, while a PTAS was known. This is the first work on Weighted Sequences avoiding the Log-Probability model, a simplifying assumption related to exact computations of reals. Our paper "Longest Common Subsequence on Weighted Sequences" [KT20] received the Best Paper Award at CPM '20.

## Not included in the thesis

**Hashing:** A general algorithmic paradigm (e.g. in streaming) is Bernoulli Sampling, where we do not process the whole input, but rather just a sampled subset of it. In certain cases, such as Set Similarity, Distinct Elements, and Trajectory Sampling, it is crucial that this sampling is coordinated. This is implemented by applying a hash function and keeping the elements whose hash value is below a certain threshold. However, due to the hash function the samples are no longer independent, which results in weak correctness guarantees. The standard solution is then to boost the probability of success by performing independent repetitions of the same algorithm.

In this work we show that using Tabulation-1Permutation, a hash function with strong concentration guarantees, we can run a single repetition of such algorithms without sacrificing the probability of success, as if the hash function was truly random. We support our theoretical results through experiments which also demonstrate the practicality of Tabulation-1Permutation. The manuscript [ADK<sup>+</sup>20] is submitted to VLDB '22.

**Network Dynamics:** In this work we suggest a particular type of Network Dynamics supporting structural dynamics, guided by local thresholding rules executed in each node. Its microscopic structure appears to be simple, so that we are able to rigorously argue about it, but still flexible, so that we are able to design meaningful microscopic local rules that give rise to interesting

macroscopic behaviors. We rigorously exhibit our claims by designing a simple protocol that provably computes the  $k$ -core of the network as well as by showing that the suggested Network Dynamics is Turing-Complete. Most importantly, we construct general tools for proving stabilization of our Network Dynamics and prove speed of convergence in a restricted setting. We also disprove the conjectured convergence of a certain community detection algorithm by Zhang et al. [ZWWZ09] (KDD '09), which falls under our framework. Our paper “Threshold-based Network Structural Dynamics” [KST21] was accepted at SIROCCO '21 and invited to a special issue of the Theoretical Computer Science journal.

**Labeling Schemes:** In Labeling Schemes for graph problems, we preprocess the input graph and assign a binary string (label) to each node. Then, given the labels of two nodes (but no access to the input graph) one should be able to compute some function of the graph and the two nodes. The goal is to minimize the maximum size of any label. We studied the problem of routing in trees, where the input graph is a tree and given the labels of two nodes  $u, v$  the goal is to output the first edge on the path from  $u$  to  $v$ . We improved the state-of-the-art solution from  $\log n + O(\log_{\log n} n \log \log \log n)$  bits to  $\log n + O(\log_{\log n} n)$ . This is an unpublished result, as few months after our solution the optimal size was improved to  $\log n + O(\log \log^2 n)$  by Gawrychowski et al. [SODA '21].

## Papers

### **Fitting distances by tree metrics minimizing the total error within a constant factor**

V. Cohen-Addad, D. Das, E. Kipouridis, N. Parotsidis, and M. Thorup  
FOCS 2021 [CDK<sup>+</sup>21]

### **Constrained Correlation Clustering: Deterministically and Combinatorially**

E. Kipouridis, J. Ø. Klausen, and M. Thorup  
Submitted to ICALP 2022

### **A simple algorithm for multiple-source shortest paths in planar digraphs**

D. Das, E. Kipouridis, M. P. Gutenberg, and C. Wulff-Nilsen  
SOSA 2022 [DKGW22]

### **Longest Common Subsequence on Weighted Sequences**

E. Kipouridis, K. Tsihclas  
CPM 2020 [KT20]

### **No repetition: Fast streaming with highly concentrated hashing**

A. Aamand, D. Das, E. Kipouridis, Jakob B. T. Knudsen, P. M. R. Rasmussen, and M. Thorup  
Submitted to VLDB 2022 [ADK<sup>+</sup>20]

### **Threshold-based network structural dynamics**

E. Kipouridis, K. Tsihclas, and P. Spirakis  
SIROCCO 2021 [KST21]

### **Compact routing schemes**

E. Kipouridis, and M. Thorup  
Unpublished Manuscript

## Acknowledgements

I hate how incomplete this section feels, mentioning just a tiny fraction of the people I want to acknowledge, and containing just a tiny fraction of what I want to convey.

At the very least, I would like to express my gratitude to my supervisor Mikkel Thorup, for our countless hours of algorithmic discussions, drinking, running, and mushroom hunting. You were always there to remind me that publications are only the fruit of research, not its end goal, when I was getting disappointed by rejections. You did not only give me space to make my own mistakes, but with your unique way you even taught me to embrace them. Above all, you taught me by example: always happy, full of energy, my academic mentor, and at the same time a dear friend to whom I felt safe entrusting even my most personal issues. Thank you Mikkel.

To my co-supervisor Christian, a beacon of sanity in the whole craziness of BARC. I especially thank you for our conversations regarding our common passion, teaching. Not just for the useful techniques (which I already try to embed in my own teaching), but mainly for sharing your view on its importance. Let me not forget our B5 Thursdays with Jacob Holm, Radu Curticapean and Nikos Parotsidis, as well as the good times of eating rocks in deserted islands...

To Karl Bringmann, for hosting me in Saarbrücken for 3 months and trusting me with a postdoc position. It was a pleasure meeting your group.

To Anne Primdahl Kristensen, the person behind everything. Whose days last 30 hours, so that, among everything else, she always finds time to prepare new bonding activities for BARC. Who so tenderly does everything she can to lift every single one's spirit. Thank you for all the cakes, for all the hugs, for all the smiles. Nothing would be nearly as nice without you.

To the incredibly hardworking Radu Curticapean, Debarati Das, and Nikos Parotsidis. It is an honor to be close friends with people I admire so much.

To Jonas Østergaard Klausen for the time we spent driving, organizing trips, drinking, and researching. And to Viktor Fredslund-Hansen, the most friendly colleague one can hope for.

To the rest of my co-authors: Anders Aamand, Vincent Cohen-Addad, Loukas Georgiadis, Jakob Bæk Tejs Knudsen, Charis Papadopoulos, Maximilian Probst Gutenberg, Peter Michael Reichstein Rasmussen, Paul Spirakis, and Kostas Tsichlas. For the moments of creativity as well as the moments of disappointment we shared.

To everyone related to BARC, for our trips, board games nights, winter bathing, late night discussions, and many more wonderful experiences. I was really not expecting such a wonderful time when starting my PhD.

And to two more people. My friend Sofia Kourtparasidou: I have no idea how I would have gone through the difficult times without you. I cannot thank you enough for that. And to my father Ioannis Kipouridis, for moving heaven and earth just because 14 years old Evangelos thought that "computers are cool". Thank you for setting this journey in motion, even though you are still not sure what exactly I am doing.



# Contents

|          |                                                                                                    |            |
|----------|----------------------------------------------------------------------------------------------------|------------|
| <b>1</b> | <b>Introduction</b>                                                                                | <b>1</b>   |
| <b>2</b> | <b>Fitting Distances by Tree Metrics Minimizing the Total Error within a Constant Factor</b>       | <b>2</b>   |
| 2.1      | Introduction . . . . .                                                                             | 2          |
| 2.2      | Previous work . . . . .                                                                            | 3          |
| 2.3      | Our contribution . . . . .                                                                         | 4          |
| 2.3.1    | Techniques . . . . .                                                                               | 4          |
| 2.4      | Further research directions . . . . .                                                              | 5          |
| <b>3</b> | <b>Constrained Correlation Clustering: Deterministically and Combinatorially</b>                   | <b>7</b>   |
| 3.1      | Introduction . . . . .                                                                             | 7          |
| 3.2      | Previous work . . . . .                                                                            | 8          |
| 3.3      | Our contribution . . . . .                                                                         | 9          |
| 3.4      | Further research directions . . . . .                                                              | 10         |
| <b>4</b> | <b>Multiple-Source Shortest Paths in Planar Digraphs</b>                                           | <b>12</b>  |
| 4.1      | Introduction . . . . .                                                                             | 12         |
| 4.2      | Applications . . . . .                                                                             | 12         |
| 4.3      | Previous work . . . . .                                                                            | 13         |
| 4.4      | Our contribution . . . . .                                                                         | 13         |
| 4.4.1    | Techniques . . . . .                                                                               | 14         |
| 4.5      | Further research directions . . . . .                                                              | 15         |
| <b>5</b> | <b>Longest Common Subsequence on Weighted Sequences</b>                                            | <b>16</b>  |
| 5.1      | Introduction . . . . .                                                                             | 16         |
| 5.2      | Previous work . . . . .                                                                            | 16         |
| 5.3      | Our contribution . . . . .                                                                         | 17         |
| 5.4      | Further research directions . . . . .                                                              | 18         |
| <b>A</b> | <b>FOCS: Fitting Distances by Tree Metrics Minimizing the Total Error within a Constant Factor</b> | <b>27</b>  |
| <b>B</b> | <b>Unpublished: Constrained Correlation Clustering: Deterministically and Combinatorially</b>      | <b>74</b>  |
| <b>C</b> | <b>SOSA: Multiple-Source Shortest Paths in Planar Digraphs</b>                                     | <b>103</b> |



# Chapter 1

## Introduction

In accordance with the guidelines of the PhD School of the University of Copenhagen, this thesis is presented as a synopsis of a subset of the papers produced over the course of the PhD programme. The main focus is on clustering problems, and more specifically on ones where the input describes the ideal pairwise relationship between any two objects in the final clustering. To demonstrate the breadth of my PhD work, I also include results related to shortest paths and strings.

**Organisation.** The thesis has been divided into four chapters, each one presenting a different paper or manuscript. The first two are related to clustering, while the following two touch on different aspects of Theoretical Computer Science. More specifically, the first chapter is related to a paper on finding the closest Tree Metric to a Distance Matrix. The second chapter discusses the results of a manuscript related to variants of Correlation Clustering. The third chapter discusses the results of a paper on the Multiple Source Shortest Paths in Planar Digraphs problem. Finally, the fourth chapter relates to a paper addressing the Longest Common Subsequence problem on a generalization of regular strings known as Weighted Sequences. The appendix contains the full versions of these 4 papers and manuscripts, in the same order.

Each chapter introduces the related problems, provides a survey of the relevant literature, and discusses the results of the corresponding paper or manuscript. Finally it considers further related research topics and open problems. As each chapter is a synopsis of an actual paper or manuscript, no full proofs are provided. Instead, the appendix contains the full versions for the interested reader.

**Notation.** Most notation will be introduced as needed. For  $n \in \mathbb{N}$ , we denote by  $[n]$  the set  $\{1, 2, \dots, n\}$ . We use  $\tilde{O}(\cdot)$ -notation to suppress logarithmic factors in  $n$ . We denote all subsets of size  $k$  of a set  $S$  by  $\binom{S}{k}$ . The symmetric difference between two sets  $A, B$  is denoted by  $A \Delta B$ .

## Chapter 2

# Fitting Distances by Tree Metrics Minimizing the Total Error within a Constant Factor

In this chapter, we present the paper “Fitting Distances by Tree Metrics Minimizing the Total Error within a Constant Factor”, presented at FOCS ’21. We start with a short introduction of the problem, proceed with an overview of previous work, and then discuss the results of the paper. We conclude with addressing further research directions and open problems.

### 2.1 Introduction

Suppose we are given a tree with  $n$  special nodes, which we call species, and  $O(n)$  other nodes. Furthermore, the edges of the tree have positive weights. The set of pairwise distances between the species forms a type of metric known as *tree metric*. In the special case where the tree is rooted, all the species are leaves, and all leaf-to-root distances are equal, we get a type of metric known as an *ultrametric*.

In this work we consider the numerical taxonomy problem [CSE67, SS62, SS63]. In the simple model introduced by Cavalli-Sforza and Edwards in 1967 [CSE67], the weight of an edge in an evolutionary tree is the evolutionary distance between its 2 endpoints, and the evolutionary distance between 2 non neighboring species is the sum of weights along their path. Therefore, given an evolutionary tree, the set of pairwise evolutionary distances induces a tree metric.

Now consider the reconstruction version of the above model, where instead of determining distances given a tree, we are given the distances and need to determine a tree. As there may be no tree metric exactly satisfying all distances, we ask for a *closest* tree metric (or closest ultrametric) to the input distances.

Perhaps the most natural definition of a closest tree metric (resp. ultrametric) is a tree metric (resp. ultrametric) on the same  $n$  species as the input distances, such that the sum of pairwise errors between the input distances and the metric distances is minimized. More formally, given a set  $S$  of  $n$  species and pairwise distances  $\mathcal{D} : \binom{S}{2} \rightarrow \mathbb{R}_{>0}$ , a closest tree metric (resp. ultrametric)  $T$  is any tree metric (resp. ultrametric)  $T$  minimizing

$$\|T - \mathcal{D}\| = \sum_{\{i,j\} \in \binom{S}{2}} |T(i,j) - \mathcal{D}(i,j)|$$

We call the problem  $L_1$ -fitting tree metrics when we ask for any tree metric, and  $L_1$ -fitting ultrametrics when we specifically ask for an ultrametric. Similarly, as we discuss in the related work, one can work on different  $L_p$  norms, and minimize

$$\|T - \mathcal{D}\|_p = \left( \sum_{\{i,j\} \in \binom{S}{2}} |T(i,j) - \mathcal{D}(i,j)|^p \right)^{1/p}$$

$L_1$  and  $L_2$  are the most interesting cases and this is exactly how the reconstruction problem was introduced back in the 1960s [CSE67, SS62, SS63].

These problems are directly related to evolutionary biology. For example, viewing the depth of a node in an ultrametric as the time in which a corresponding species existed, we can view all leaves as species that exist today. For these species we can estimate their evolutionary distance, and want to reconstruct the tree of evolution. Of course evolution is not that linear, and a tree metric may be more relevant than an ultrametric. In fact, the whole concept of distances in the tree relating to evolutionary distance may be too simplistic, as certain changes from parent to child may be reverted in the grandchild. However, Farach and Kannan show that even for more accurate stochastic models [Cav78, Far72], one can apply logarithms to convert estimated distances into a different type of distances for which we can then find the closest tree metric or ultrametric. In short, even when not applied directly, finding the closest tree metric or ultrametric is a powerful tool for solving problems related to the reconstruction of evolution.

Biology is not the only field related to our problems. Medicine, ecology and linguistics are just some of the fields where these concepts appear. In fact, even from as early as 350 BC, we have discussions between Plato and Aristotle concerning the optimal such trees, in the context of classification<sup>1</sup>. Moreover, ultrametrics are important objects for machine learning and data analysis [CM10], and many different algorithms (such as single, complete or average “linkage” algorithms) have been suggested to approach the closest ultrametric problem [CKMM19, MW17]. Finally, many NP-Hard problems on general metrics are easy to solve on tree metrics (see Chapter 10.2 “Solving NP-Hard Problems on Trees” in [KT06]), and thus finding the closest tree metric before solving them is a natural approach.

## 2.2 Previous work

In 1977, an  $O(|S|^2)$  time algorithm [WSSB77] (linear in the size of the input) for detecting the tree  $T$  was developed, in the special case where the input distances  $\mathcal{D}$  form a tree metric. The result trivially holds for the case of ultrametrics as well. However, in the case where input distances  $\mathcal{D}$  do not form a tree metric, progress was much slower.

In 1993, about 25 years after the introduction of the simple model by Cavalli-Sforza and Edwards [CSE67], the problem of  $L_\infty$ -fitting ultrametrics was solved exactly [FKW95]. In 1996, the  $L_\infty$ -fitting tree metrics problem was approximated with a factor 3 in linear time [ABF<sup>+</sup>99]. In fact, the authors show something more powerful: under some technical assumptions, an  $\alpha$  approximation for the  $L_p$ -fitting ultrametrics problem can be converted to a  $3\alpha$  approximation for the  $L_p$ -fitting tree metrics problem, for any  $p$ .

The progress for  $L_p$  norms,  $p < \infty$ , was even slower. In 1999, Ma et al. [MWZ99] designed an  $O(n^{1/p})$  approximation for a problem related to  $L_p$ -fitting ultrametrics, where distances in the ultrametric are not smaller than the input distances. In 2004, Dhamdhare [Dha04], in an attempt to develop techniques to solve the  $L_1$ -fitting tree metrics problem, gave an  $O(\log n)$

---

<sup>1</sup><https://iep.utm.edu/classifi/>, Internet Encyclopedia of Philosophy

| Norm        | $L_1$       | $L_p, p < \infty$                  | $L_\infty$  |
|-------------|-------------|------------------------------------|-------------|
| Tree metric | $\Theta(1)$ | $O(((\log n)(\log \log n))^{1/p})$ | $\Theta(1)$ |
| Ultrametric | $\Theta(1)$ | $O(((\log n)(\log \log n))^{1/p})$ | 1           |

Table 2.1: Tree fitting approximation factors.

approximation for the problem of finding a line metric minimizing additive distortion. The first non-trivial result regarding the general  $L_p$ -fitting ultrametrics problem came from Harb, Kannan and McGregor [HKM05] in 2005. The authors designed an  $O(\min\{n, k \log n\}^{1/p})$  approximation, where  $k$  is the number of distinct input distances. The authors erroneously claim the same approximation for  $L_p$ -fitting tree metrics, using the reduction from [ABF<sup>+</sup>99]. However, this reduction may create  $\omega(k)$  distinct distances.

The same year as [HKM05], Ailon and Charikar [AC11] designed the first truly polylogarithmic approximation for general  $L_p$ -fitting tree metrics and  $L_p$ -fitting ultrametrics. They provide a general  $O(((\log n)(\log \log n))^{1/p})$  approximation for both problems. Their solution proceeds by rounding an LP, using techniques similar to the ones used for Multicut. In fact their approach also solves more difficult weighted versions of the problems, that are equivalent to Multicut. The authors also design a  $k + 2$  approximation for the special  $L_1$  case, where  $k$  is the number of distinct input distances.

Ailon and Charikar [AC11] conclude that “Determining whether an  $O(1)$  approximation can be obtained is a fascinating question. The LP formulation used in our [their] work could eventually lead to such a result”. The integrality gap of the natural LP that they round was only known to lie between 2 and  $O((\log n)(\log \log n))$ . However, breaking the  $\log n$  barrier requires considerably new techniques, as an  $o(\log n)$  approximation for Multicut would be a major breakthrough.

## 2.3 Our contribution

In this work, we provide constant factor approximations, both for  $L_1$ -fitting tree metrics and for  $L_1$ -fitting ultrametrics. Under some assumptions (which we also remove in this work) it was known that these problems are APX-Hard, therefore our algorithms achieve asymptotically optimal approximation factors. Furthermore, we show that the integrality gap of the natural LP relaxation for  $L_1$ -fitting ultrametrics is  $O(1)$ . Finally, we prove that for any  $p$ , an  $\alpha$  approximation for  $L_p$ -fitting ultrametrics translates to a  $(3 + o(1))\alpha$  approximation for  $L_p$ -fitting tree metrics, bypassing the need for the extra assumptions of the similar result in [ABF<sup>+</sup>99].

### 2.3.1 Techniques

The main technical contribution of our work is solving  $L_1$ -fitting ultrametrics, as the tree metrics version follows by modifications of the result in [ABF<sup>+</sup>99]. In what follows, we discuss some key points of our approach.

**Correlation Clustering** One can view an ultrametric as a hierarchical clustering, where at the bottommost level (leaves) every species is in a singleton cluster, on the topmost level (root) every species is in the same cluster, and in general the clustering at any level subdivides the clustering at higher levels. Therefore, if we approximate the optimal clustering at each level within a constant factor, the total cost is within a constant factor of the optimal cost, with the only problem being that the clusterings we produce are not consistent with each other.

In our algorithm we first solve each level of the problem independently, and then focus on modifying the acquired clusterings so that they become consistent. The per-level problem we need to solve is Correlation Clustering, a well studied clustering problem for which efficient constant factor approximations are known.

**Modifying the objective of the LP** The objective of the natural LP relaxation for  $L_1$ -fitting ultrametrics has *hierarchical consistency*, in the sense that for any two species  $u, v$ , it wants them separated from the bottommost level up to some level  $l_{u,v}$ , and wants them in the same cluster at levels higher than  $l_{u,v}$ . However, it does not have *level consistency*, in the sense that for three species  $u, v, w$  and a level  $t$ , it may be that at level  $t$  it wants  $u, v$  together,  $u, w$  together, but  $v, w$  separated. Interestingly enough, existing solutions do not make any use of the hierarchical consistency of the LP relaxation; in fact they would work even in the more general setting where none of the aforementioned properties hold<sup>2</sup>.

Our solution solves the more general setting as well. It does so by using the per-level clusterings to carefully modify the objective of the LP relaxation to a new objective that has level consistency, but not hierarchical consistency. The modifications are done in a way that ensures that a solution to the new objective is an approximate solution to the initial objective as well<sup>3</sup>.

**LP-Cleaning** Having a new LP at hand, we use it to produce candidate clusters for each level. In particular, we start with the clusters produced by solving Correlation Clustering per level. Then we use the solution of the LP to determine which of these clusters are indeed good candidates, and which can be completely disregarded. In fact, the LP is used to remove species from clusters; we only keep the clusters that are mostly intact (say more than 99% of them is intact), and disregard the rest. We note that we only use the solution of the LP for this cleaning part of the algorithm.

**Deriving Hierarchy** Finally, using the candidate clusters, we apply a simple algorithm that modifies them in a way that produces a consistent ultrametric. The algorithm works bottom-up, and never modifies an already processed cluster. When processing a new cluster  $C$  at level  $t$  that intersects but does not contain a processed cluster  $C'$  at a lower level, the algorithm simply decides whether to remove their intersection from  $C$  or extend  $C$  to include  $C'$ . The only criterion in this decision is whether the two clusters intersected before applying the Deriving Hierarchy part of the algorithm (in which case  $C$  extends to include  $C'$ ) or not.

## 2.4 Further research directions

Perhaps the most important open question related to our work is an  $O(1)$  approximation for the  $L_2$ -fitting tree metrics problem. Given the result from [ABF<sup>+</sup>99], and its extension in our work, it is enough to give an  $O(1)$  approximation for  $L_2$ -fitting ultrametrics. However, the techniques used in the current work exploited properties of  $L_1$  that  $L_2$  does not have. Therefore it seems that considerably new techniques need to be developed for  $L_2$ .

Concerning  $L_1$ , the best known approximation factor for the weighted version (where the cost of an edge is weighted by an input edge weight) is  $O((\log n)(\log \log n))$ , by Ailon and Charikar[AC11]. Getting rid of the  $\log \log n$  factor would be an interesting problem. However

---

<sup>2</sup>We explicitly define this more general setting as the *Hierarchical Correlation Clustering* problem, as it is of independent interest.

<sup>3</sup>We explicitly define this new problem as the *Hierarchical Cluster Agreement* problem, as we believe it to be of independent interest.

going beyond that would also improve the best known approximation factor for Multicut, a notoriously hard problem.

Other than the weighted version of  $L_1$ -fitting tree metrics, there is also another interesting generalization, namely the constrained version of the problem. In this problem, we are given all pairwise distances between  $n$  species; furthermore, for each pair of species  $u, v$  we are given a lower bound  $x_{u,v}$  and an upper bound  $y_{u,v}$ . The goal is to find a tree metric minimizing the  $L_1$  error, with the constraint that for any pair of species  $\{u, v\}$ , the distance between  $u, v$  in the tree is in  $[x_{u,v}, y_{u,v}]$ .

Finally, our algorithms for  $L_1$ -fitting tree metrics and ultrametrics does not need to employ randomization. As also stated by Ailon and Charikar [AC11], it would be nice if one can provide a combinatorial algorithm that even avoids solving an LP.



## Chapter 3

# Constrained Correlation Clustering: Deterministically and Combinatorially

In this chapter, we present the manuscript “Constrained Correlation Clustering: Deterministically and Combinatorially”, part of which was submitted to ICALP ’22. We start with a short introduction of the problem, proceed with an overview of previous work, and then discuss the results of the manuscript. We conclude with addressing further research directions and open problems.

### 3.1 Introduction

Given a graph  $G = (V, E)$ , a *clustering*  $C = \{C_1, \dots, C_k\}$  is a partition of  $V$ . Each  $C_i \in C$  is called a *cluster*. With each clustering  $C$ , we associate an edge-set  $E_C$  defined as  $E_C = \bigcup_{i=1}^k \binom{C_i}{2}$ . In other words, we create a clique connecting all nodes in the same cluster of the clustering.

In the *Correlation Clustering* problem, we are given an input graph  $G = (V, E)$  and the output is a clustering  $C$ . The goal is to minimize the symmetric difference  $|E \Delta E_C|$ . Intuitively, we want to approximate a given graph by a collection of cliques.

One can view Correlation Clustering in a slightly different way: the input graph  $G$  contains all pairwise *preferences* of the nodes. This means that for every two nodes, we know whether we prefer to include them in the same cluster or not. The goal is then to produce a clustering that violates the minimum number of such preferences.

There are certain applications however where some pairs are more important than others, and we must always satisfy their preference in the output clustering. This motivates the *Constrained Correlation Clustering* problem, where the input is a graph  $G = (V, E)$ , a set of friendly pairs  $F \subseteq \binom{V}{2}$  and a set of hostile pairs  $H \subseteq \binom{V}{2}$ . The output is a clustering  $C$  such that for all pairs  $\{u, v\} \in F$  we have  $u, v$  in the same cluster, and for all pairs  $\{u', v'\} \in H$  we have  $u', v'$  in different clusters. The clustering  $C$  shall minimize  $|E \Delta E_C|$  among all clusterings that satisfy the same properties related to  $F, H$ .

In the case of Constrained Correlation Clustering we refer to the pairs in  $F$  and  $H$  as *hard constraints*, or simply constraints, because we must always satisfy them.

Finally, we introduce the Node-Weighted Correlation Clustering problem, where each node  $u$  has a different importance. The input is a graph  $G = (V, E)$  and a function  $w : V \rightarrow \mathbb{N}_{>0}$ . The

output is a clustering  $C$  of  $V$  minimizing

$$\sum_{\{u,v\} \in E \Delta E_C} w(u) \cdot w(v)$$

Therefore, instead of having a cost 1 for each violated preference, as in the previous two problems, we have a cost  $w(u) \cdot w(v)$ .

## 3.2 Previous work

Correlation Clustering is one of the most successful clustering objectives, mainly due to its simplicity and the fact that, in contrast with clustering objectives like k-Means and k-Median, the number of clusters in the output clustering does not need to be given as input. It was introduced by Bansal et al. [BBC04], who proved NP-Hardness and provided a deterministic constant factor approximation, the constant being about 15,000. Subsequently a 4 approximation was given by Charikar et al. [CGW05], a 2.5 approximation by Ailon et al. [ACN08], and a deterministic 2.06 approximation by Chawla et al. [CMSY15]. The last three solutions are all based on rounding the natural LP.

Due to the important practical applications of Correlation Clustering, minimizing the approximation factor is not the only research direction. Among several other directions, such as parameterized algorithms [FKP<sup>+</sup>14], sublinear and streaming algorithms [AW21], massively parallel computation (MPC) algorithms [CLM<sup>+</sup>21], and differentially private algorithms [BEK21], in this work we mainly focus in the following:

1. *(Randomized) Combinatorial Algorithms:* We use the term combinatorial to refer to algorithms not solving an LP. Charikar et al. [ACN08] provide a randomized combinatorial 3 approximation; even though its approximation is worse than the 2.5 approximation they design in the same paper, its advantage is being combinatorial.
2. *Deterministic Combinatorial Algorithms:* Much later than the aforementioned solutions with less than 5 approximation factors, a deterministic combinatorial 6 approximation algorithm was designed in [Vel21]. Deterministic combinatorial algorithms were also pursued in [vZW09].
3. *Pivoting Algorithms:* These are algorithms which select one node  $u$  and include  $u$  and all nodes that prefer to be with  $u$  in one cluster. Then they remove these nodes and recurse. The simplicity of these algorithms, as well as some of their applications make them desirable. The 3 approximation in [ACN08] as well as algorithms in [vZW09] fall under this category.

The main problem with the formulation of Correlation Clustering is that each violated preference costs the same. In the weighted version of Correlation Clustering, we are given a weight for each preference, and violating a preference induces cost equal to its weight. Even though this problem has more applications, its  $O(\log n)$  approximation is very difficult to improve (if at all possible), as the problem is equivalent to Multicut [DEFI06].

Due to the hardness of the general weighted version, research has focused on special cases where constant factor approximations are possible [MTG21, PM15]. Constrained Correlation Clustering is one such special case, where certain preferences, which we call hard-constraints, must never be violated (infinite weight), while the rest of the preferences all have weight 1.

Regarding Constrained Correlation Clustering, van Zuylen et al. designed a deterministic 3 approximation [vZW09] working as follows:

1. Using the hard constraints  $F, H$ , the input graph  $G = (V, E)$  is modified to produce a new graph  $G' = (V, E')$ .
2. Graph  $G' = (V, E')$  is treated as an instance of (unconstrained) Correlation Clustering, ignoring the actual hard constraints. A pivoting algorithm is applied on  $G'$ .

We note that no pivoting algorithm is possible for Constrained Correlation Clustering. Even though in the last step a pivoting algorithm is used, it is applied in a graph  $G'$  different from the original graph  $G$ . The use of this pivoting subroutine is however crucial, as this is what ensures that no hard-constraints are violated.

Throughout [vZW09], the authors focus on deterministic algorithms, and, whenever possible, combinatorial algorithms are preferred. Their solution for Constrained Correlation Clustering is, however, not combinatorial; in fact both the modification of the graph and the pivoting algorithm need to use the values of the LP variables.

### 3.3 Our contribution

In this work, we give a deterministic combinatorial constant factor approximation for Constrained Correlation Clustering. In order to do so, we design the first deterministic combinatorial pivoting algorithm for Correlation Clustering. In fact, our algorithm is the best known deterministic combinatorial algorithm for Correlation Clustering, even among non-pivoting algorithms. Finally, using the insight from our Constrained Correlation Clustering solution, we introduce the Node Weighted Correlation Clustering problem and provide a constant factor approximation.

**Constrained Correlation Clustering** Regarding Constrained Correlation Clustering, we prove the following theorem:

**Theorem 3.1.** *There exists a deterministic combinatorial algorithm that solves Constrained Correlation Clustering in  $O(|V|^6)$  time with an approximation factor less than 29. There exists a faster such algorithm running in  $O(|V|^3)$  time, with a 42 approximation.*

To solve this problem, we provide deterministic combinatorial counterparts for both steps of the algorithm in [vZW09].

Concerning the first step, we carefully modify the input graph so that any pivoting algorithm on the modified graph produces a clustering that does not violate any hard constraint. We also make sure that the cost of the output clustering in the modified graph is not much larger than the cost of the same clustering in the input graph. From a high level view, this is guaranteed by only modifying parts of the input graph for which even the optimal clustering is bound to pay a lot. As we cannot use an LP to guide these modifications, we determine 4 types of subgraphs for which any clustering is bound to pay.

Concerning the second step, we design the first deterministic combinatorial pivoting algorithm for (unconstrained) Correlation Clustering.

**Correlation Clustering** Regarding Correlation Clustering, we prove the following theorem:

**Theorem 3.2.** *There exists a deterministic combinatorial algorithm based on pivoting that solves Correlation Clustering in  $O(|V|^6)$  time with a 5.8 approximation factor. There exists a faster such algorithm running in  $O(|V|^3)$  time, with a 9 approximation.*

Our deterministic combinatorial pivoting algorithm for Correlation Clustering achieves the best known approximation factor among any deterministic combinatorial algorithm for Correlation Clustering, even among non-pivoting ones.

To the best of our knowledge, all deterministic algorithms for Correlation Clustering lower bound the optimal clustering using bad triplets. A bad triplet is an induced subgraph on 3 vertices, such that only one pair is not connected. Notice that any clustering needs to pay at least 1 for the 3 pairs in a bad triplet.

To go below 6 approximation (the state-of-the-art deterministic combinatorial approximation [Vel21]), we use new subgraphs for which any optimal clustering needs to pay. The main advantage of these new subgraphs is that the minimum cost any clustering needs to pay for them is larger than the maximum number of pair-disjoint bad triplets in them.

As an example, take the claw graph ( $K_{1,3}$ ). Any clustering needs to pay at least 2 for the four pairs in  $K_{1,3}$ . However no 2 bad triplets in  $K_{1,3}$  are pair disjoint, which made previous approaches relate  $K_{1,3}$  with a cost less than 2.

Using our stronger lower bounds, we are able to give a deterministic combinatorial 5.8 approximation algorithm for Correlation Clustering based on pivoting.

We also prove a lower bound on the approximation factor of any pivoting algorithm for Correlation Clustering:

**Theorem 3.3.** *No pivoting algorithm for Correlation Clustering has  $3 - \Omega(1)$  approximation factor.*

This means that if we allow randomization, the solution from [ACN08] is optimal. Similarly, if we do not allow randomization but allow solving an LP, the solution from [vZW09] is optimal.

**Node Weighted Correlation Clustering** As weighted Correlation Clustering is equivalent to Multicut, an  $o(\log n)$  lower bound requires a major breakthrough. In this work we introduce an alternative type of weighting, where weights are assigned on the nodes, rather than on pairs of nodes. We provide a constant factor approximation for this problem.

The solution relates an instance  $I$  of Node Weighted Correlation Clustering with an exponentially larger instance  $I'$  of Constrained Correlation Clustering. Despite of its large size, the special structure of  $I'$  allows us to simulate an approximation algorithm in time linear in the size of  $I$ .

We prove the following result:

**Theorem 3.4.** *There exists a randomized combinatorial algorithm for Node Weighted Correlation Clustering with a 3 (expected) approximation factor. The running time of the algorithm is  $O(|V| + |E|)$ .*

### 3.4 Further research directions

Perhaps the most interesting research direction stemming from this work is the design of optimal pivoting algorithms for Correlation Clustering. Our lower bound shows that any pivoting algorithm has approximation factor at least 3. If randomization is allowed, the algorithm from [ACN08] is thus optimal, and if solving an LP is allowed, then the algorithm from [vZW09] is optimal. However there is still a small gap (3 vs 5.8) when we insist on deterministic combinatorial algorithms.

Furthermore, Node Weighted Correlation Clustering seems to be a very natural generalization of Correlation Clustering. It would be interesting to see practical applications of it. From a more theoretical perspective, it would be interesting to design a deterministic combinatorial algorithm;

it is not clear if efficiently simulating our deterministic combinatorial algorithm for Correlation Clustering in the larger implicit instance is possible, as we did with the randomized algorithm from [ACN08].

Finally, for Constrained Correlation Clustering, the analysis seems somewhat loose: the two parts of the algorithm are analyzed separately, and the final approximation factor is, roughly, the product of the two approximations. It would be interesting to improve the current analysis or at least find some example showing it is tight.

## Chapter 4

# Multiple-Source Shortest Paths in Planar Digraphs

In this chapter, we present the paper “A Simple Algorithm for Multiple-Source Shortest Paths in Planar Digraphs” [DKGW22], presented at SOSA '22. We start with a short introduction of the problem, proceed with an overview of applications and previous work, and then discuss the results of the paper. We conclude with addressing further research directions and open problems.

### 4.1 Introduction

Let  $G = (V, E)$  be a planar embedded directed graph with non-negative edge weights, and  $f$  be a face of  $G$ . We are interested in a data structure supporting shortest path queries from vertices in  $f$  to any other vertex. The objective is to minimize the preprocessing and the query time.

This problem is known as the *Multiple-Source Shortest Paths Problem in Planar Digraphs (MSSP)*, and is a fundamental building block in many state-of-the-art algorithms for planar graph problems, including All Pairs Shortest Paths, Max Flow and Min Cut algorithms. Due to the problem’s importance and the technical complexity of previous work, our aim was also a significantly simpler algorithm.

### 4.2 Applications

**APSP** The most straightforward application of Multiple-Source Shortest Paths in Planar Digraphs is arguably the All Pairs Shortest Paths problem (APSP) in Planar Digraphs. There are 3 different versions of APSP whose state-of-the-art solutions use MSSP as a building block:

- Exact Distance Oracles: The most standard version of APSP where we are interested in exact solutions. Recently a data structure with  $n^{o(1)}$  query time, requiring  $n^{1+o(1)}$  space, was achieved through a series of breakthrough papers [CADWN17, GMWW18, CGMW19, LP21]. MSSP is used as a subroutine by the state-of-the-art result in [LP21], as well as by [CGMW19].
- Approximate Distance Oracles: In this version of APSP we are satisfied with  $1 + \epsilon$  approximating the answer. Such a data structure using  $\tilde{O}(n\epsilon^{-1})$  space and preprocessing time was designed by Thorup [Tho04], assuming that the ratio between the largest and the

smallest edge weight is polynomial in  $n$ . The query time is only  $O(\log \log n + \epsilon^{-1})$ . Further improvements have then sped-up the preprocessing time by polylogarithmic factors, via improvements to the state-of-the-art MSSP data structure.

- **Exact Dynamic APSP:** The exact version of APSP under the dynamic setting, where updates of edge weights are supported. The classic data structure in [FR06] uses  $\tilde{O}(n)$  space and preprocessing time, and  $\tilde{O}(\sqrt{n})$  query time for the static case. It can be modified to work in the dynamic setting, with  $\tilde{O}(n^{2/3})$  update time and  $\tilde{O}(n^{2/3})$  query time. Again, while [FR06] does not directly employ MSSP, Klein [Kle05] sped-up this solution by polylogarithmic factors, via incorporating an improved MSSP data structure.

There are various improvements over these seminal results, mainly by considering different trade-offs [FHMWN20, LP21] or by improving logarithmic or doubly-logarithmic factors [MWN10, GK18, MNW18]. The important point is that MSSP is present in almost all of those articles. Furthermore, it is important that improvements via the use of MSSP result in a more modular and re-usable design that makes APSP algorithms simpler to understand and implement.

One more direct application of MSSP is dense distance graphs. From a high level view, this is a common strategy for planar graph problems, where the input graph  $G$  is decomposed using vertex separators recursively. The dense distance graph of any (planar) subgraph  $G'$  obtained by such a decomposition is basically a complete graph on the vertices of the outer face of  $G'$ , such that the weight of an edge is the shortest path distance in  $G'$  between its two endpoints. MSSP can then be applied to obtain the dense distance graph, which in turn finds applications in cuts and flows problems [BKM<sup>+</sup>17, BSWN15, INSWN11].

### 4.3 Previous work

Fakcharoenphol’s and Rao’s solution to MSSP [FR06] gives the first non-trivial bounds for the problem. It requires  $\tilde{O}(n)$  preprocessing time and space and  $\tilde{O}(\sqrt{n})$  query time. Klein’s work [Kle05] significantly improved the query time to  $O(\log n)$ , while requiring  $O(n \log n)$  preprocessing time and space. This preprocessing time and space is tight in  $n$ , as later demonstrated by [EK13]. Klein and Eisenstat [EK13] also show how to remove all logarithmic factors in the special but natural case of undirected, unit-weighted planar graphs.

The solution of Klein requires the use of dynamic trees, persistency [DSST89], and the explicit use of the dual of the input planar graph by the algorithm. It is based on the non-trivial observation that, on average, the shortest path trees of two successive vertices along the face  $f$  differ by a constant number of edges.

Finally, Cabello, Chambers and Erickson [CCE13] used the same structural observations of Klein, along with a new perspective to the problem, to extend the results to surface-embedded graphs of genus  $g$ . Their data structure requires  $O(gn \log n)$  preprocessing time and space, and  $O(\log n)$  query time.

### 4.4 Our contribution

**Bounds** In this work we give a new solution to MSSP that slightly improves the state-of-the-art solution by Klein [Kle05] when the number of vertices  $|f|$  in the face  $f$  is subpolynomial in  $n$ , and recovers his bounds otherwise. More specifically, it requires  $O(n \log |f|)$  preprocessing time and space, which is optimal for parameters  $n, |f|$  by a straightforward extension of the arguments in [EK13]. The query time is  $O(\log |f|)$ .

**Simplicity** An important feature of our algorithm is its simplicity. We only apply a standard divide and conquer approach on the input graph. In each recursive call we compute two shortest path trees from two vertices and contract certain edges that are common in both trees before recursing. The analysis is also considerably simpler.

In contrast with Klein’s data structure which requires black-box access to algorithms for single source shortest paths, persistency, dynamic trees, and an interplay between primal and dual planar graphs, our algorithm only requires black-box access to single source shortest paths computations. Replacing the  $O(n)$  result by Henzinger et al. [HKRS97] by standard Dijkstra’s algorithm gives a self-contained algorithm and analysis that we believe can be taught even at an advanced undergraduate level, at the expense of an  $O(\log n)$  factor in the preprocessing time.

**Practical algorithm** We believe that our algorithm, using Dijkstra’s algorithm for computing single source shortest paths, is also a very practical one. That is because it does not need to use persistency or dynamic trees, while Dijkstra’s algorithm is known to perform exceptionally well on real-world graphs. In fact, a very successful method of computing shortest-paths in road networks is based on contraction hierarchies and fast single source shortest path computations [GSSD08]. It is perceivable that our algorithm can be implemented rather easily by adapting the components of this framework.

#### 4.4.1 Techniques

From a high-level view, our algorithm is a divide and conquer one. Initially we want to compute the shortest path trees from all vertices  $r_0, \dots, r_{|f|-1}$  in  $f$ . To do that, we compute the shortest path trees from two vertices that split the problem in two halves, namely  $r_0$  and  $r_{\lfloor \frac{|f|-1}{2} \rfloor}$ . Instead of directly recursing on the two natural subproblems, which would result in the trivial algorithm, we first contract certain edges that are common in both shortest path trees computed (different set of contractions for each one of the two subproblems). This effectively reduces the size of the graphs when recursing, and gives us the desired time bounds.

The intuition is that we contract an edge when we verify that all shortest paths trees related to a subproblem contain this edge. We also need to change the weight of edges whose tail coincides with the head of a contracted edge in a straightforward manner, but we omit the details in this synopsis.

The whole approach therefore boils down to detecting edges contained in all shortest path trees related to a subproblem (see Figure 4.1 for a graphical example). Suppose the shortest path  $P$  from  $r_0$  to a vertex  $y$  and the shortest path  $P'$  from  $r_{\lfloor \frac{|f|-1}{2} \rfloor}$  to  $y$  both contain an edge  $(x, y)$ . For simplicity, assume that shortest paths are unique,  $f$  is the outer face of the graph, and that  $(x, y)$  is the only edge shared by these two shortest paths.

By planarity, the union of these two paths (modulo the edge  $(x, y)$ ) is a path that splits the graph in two faces, one containing  $r_1, \dots, r_{\lfloor \frac{|f|-1}{2} \rfloor - 1}$ , and the other containing  $r_{\lfloor \frac{|f|-1}{2} \rfloor + 1}, \dots, r_{|f|-1}$ . Assuming that  $(x, y)$  lies in the second face, notice that a shortest path from anywhere in the first face to  $y$  intersects either  $P$  or  $P'$  by Jordan’s Curve Theorem; it is straightforward to verify that it thus includes  $(x, y)$ , by properties of shortest paths. Similarly if  $(x, y)$  lies in the first face.

For a query  $(r_i, u)$ , we find the recursive call where we computed the shortest path tree from  $r_i$ , and retrieve the distance from  $r_i$  to the supervertex  $U$  containing  $u$ . By the preprocessing, each supervertex corresponds to a contracted rooted tree. Then we move to the parent recursive call and retrieve the distance from the root of  $U$  to the supervertex  $U'$  containing  $u$ , and so on until we reach the initial call. As the initial call refers to the input graph, the supervertex



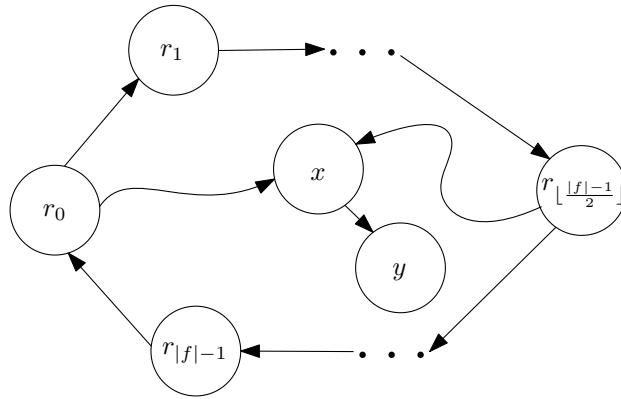


Figure 4.1: Both the shortest  $r_0 - y$  and the shortest  $r_{\lfloor \frac{|f|-1}{2} \rfloor} - y$  paths use edge  $(x, y)$ . We contract  $(x, y)$  when recursing on the top-half, as  $(x, y)$  is contained in the bottom half. That is because the shortest path from any vertex in the top-half (e.g.  $r_1$ ) to  $y$  intersects either the  $r_0 - y$  or the  $r_{\lfloor \frac{|f|-1}{2} \rfloor} - y$  shortest path, and thus contains  $(x, y)$ .

containing  $u$  is the vertex  $u$  itself. It is thus straightforward to verify that adding these distances gives the distance from  $r_i$  to  $u$  in  $O(\log |f|)$  time.

## 4.5 Further research directions

The solution of Klein [Kle05] was later extended to work on surface-embedded graphs of genus  $g$  with an extra factor  $g$  in the preprocessing time and space [CCE13]. It would be interesting to see if such an extension is possible with the current approach as well.

Furthermore, it is possible that under some assumptions on the input (either bounded weights or further structural assumptions), there exist asymptotically improved solutions.

Finally, our approach replaced the  $\log n$  factor in Klein's [Kle05] solution with a  $\log |f|$  factor. An obvious future direction would be to find applications where MSSP can be used as a subroutine and indeed the size of  $f$  is subpolynomial in  $n$ , so that they benefit from our approach.

## Chapter 5

# Longest Common Subsequence on Weighted Sequences

In this chapter we present the paper “Longest Common Subsequence on Weighted Sequences”, which received the Best Paper Award at CPM ’20. We start with a short introduction of the problem and previous related work, and then discuss the results of the paper. We conclude with addressing further research directions and open problems.

### 5.1 Introduction

Weighted Sequences are generalizations of classical strings, where in each position we have a probability distribution over some alphabet  $\Sigma$ , instead of a single character from  $\Sigma$ . We say that the probability of  $\sigma \in \Sigma$  appearing in position  $i$  of a Weighted Sequence  $X$  is  $p_X(\sigma, i)$ . For any  $i$ , it holds that  $p_X(\sigma, i) \geq 0$  for all  $\sigma \in \Sigma$  and  $\sum_{\sigma \in \Sigma} p_X(\sigma, i) = 1$ . The realized character in position  $i$  is independent from the realized characters in all other positions.

In molecular biology, there is an inherent uncertainty related with strategies for obtaining chromosome sequences (e.g. whole-genome shotgun strategies [Mye00, Ven02]). Representing chromosome sequences obtained by such strategies as a classical string disregards information and oversimplifies the data. This is what motivates the introduction of Weighted Sequences, as they better capture the obtained information.

In the *Weighted Longest Common Subsequence problem (WLCS)*, we are given two Weighted Sequences  $X, Y$  and two cut-off probabilities  $\alpha_X, \alpha_Y > 0$ . The goal is to provide a maximum length string  $s = (\sigma_1, \sigma_2, \dots, \sigma_{|s|})$  such that there exists a strictly increasing sequence  $i_1, i_2, \dots, i_{|s|}$  with  $\prod_{k=1}^{|s|} p_X(\sigma_k, i_k) \geq \alpha_X$ , and similarly a strictly increasing sequence  $j_1, j_2, \dots, j_{|s|}$  with  $\prod_{k=1}^{|s|} p_Y(\sigma_k, j_k) \geq \alpha_Y$ . Intuitively, we ask for a string that has a high probability of appearing in both Weighted Sequences. It is straightforward to verify that when  $X, Y$  are classical strings (in each position one character has weight 1 and the others 0), WLCS coincides with the classical Longest Common Subsequence problem, irrespective of the values of  $\alpha_X, \alpha_Y$ .

### 5.2 Previous work

Weighted sequences have been theoretically studied in many different contexts, such as weighted suffix trees [IMP<sup>+</sup>04], Crochemore’s partitioning [BIP14, BP18, CIM<sup>+</sup>06], the Karp-Miller-Rabin

algorithm [CIM<sup>+</sup>06], approximate and gapped pattern matching [AIKP06, ZGI10a, RS20], online pattern matching [CIPR19], weighted indexing [ACI<sup>+</sup>08, BKL<sup>+</sup>20], swapped matching [ZGI04], the all-covers and all-seeds problem [ZGFI10, ZGI10b], extracting motifs [IPT<sup>+</sup>04], and the weighted shortest common supersequence problem [AGS11, CKP<sup>+</sup>19]. Due to their applications, researchers have also provided more practical results, even without theoretical guarantees [AIMP09, IMP12].

Concerning WLCS, it was introduced by Amir, Gotthilf and Shalom [AGS10]. In this paper, the authors extend the notion of LCS in strings to the case of Weighted Sequences. They introduce two different formulations, one being the WLCS we defined, and one being an easier version for which they provide a polynomial time algorithm. Unfortunately, the applications of the second formulation are much more limited. They also hint on the NP-Hardness of WLCS by using a Turing reduction to prove the hardness of a closely related problem that replaces the products in the definition of WLCS with sums. Furthermore, their hardness result only works for unbounded alphabets. Finally, they provide a  $|\Sigma|$  approximation algorithm.

Later, Cygan et al. [CKR<sup>+</sup>16] strengthen the hardness evidence by using a Karp reduction that works even for alphabets of size 2 (notice that for alphabets of size 1 the problem is trivial) on the closely related problem that replaces products with sums. They also show that, assuming the NP-Hardness of WLCS, no FPTAS is possible unless P=NP. On the positive side, they provide a PTAS for WLCS, and a more practical 2 approximation algorithm. Finally, they show that it is enough to assume  $\alpha_X = \alpha_Y$  in the definition of WLCS, by implicitly assuming exact computations of roots and logarithms are possible.

Charalampopoulos et al. [CKP<sup>+</sup>19] proved that, unless P=NP, WLCS cannot be solved in  $O(n^{f(a)})$  time, for any function  $f(a)$ , where  $a$  is the minimum of the cut-off probabilities. We note that this result concerns exact solutions rather than approximations.

Finally, the reporting version of WLCS has been considered from a practical point of view in [BP18].

### 5.3 Our contribution

In our work, we essentially close the gap between upper and lower bounds for WLCS by improving both.

The first question we settle is the actual hardness of WLCS. We show that the problem is NP-Hard for all alphabet sizes other than the trivial  $|\Sigma| = 1$  where no uncertainty is involved and thus the answer is simply the length of the shorter Weighted Sequence. Together with our NP-Hardness, previous results imply that there exists a PTAS but no FPTAS (unless P=NP) for WLCS.

Based on the above, the most natural open question left was the existence of an EPTAS. We show that the problem behaves differently depending on the alphabet size. In the natural case of bounded alphabets (e.g. when studying DNA sequences we have  $|\Sigma| = 4$ ) we design the first EPTAS for WLCS, which is also an improved PTAS for the case of unbounded alphabets. In the case of unbounded alphabets we show that WLCS is W[1]-Hard, meaning no EPTAS exists unless FPT=W[1].

In the latter case we even provide a lower bound on the running time of any PTAS, assuming the Exponential Time Hypothesis. This lower bound shows that the dependency of our PTAS in  $\epsilon$  is in fact optimal, up to constant factors.

Finally, we reprove that assuming the cut-off probabilities  $\alpha_X, \alpha_Y$  to be equal is sufficient. Unlike the result of Cygan et al., we do not need to assume exact computations of roots and logarithms.

Table 5.1 summarizes the above discussion. Table 5.2 summarizes our results depending on the alphabet-size.

**New ideas** There are two main areas on which we made progress, namely proving the NP-Hardness of WLCS, and studying the existence of an EPTAS.

Concerning the existence of an EPTAS, our most crucial observation is the fact that the problem behaves differently in the two natural cases of bounded and unbounded alphabets. This distinction is crucial because, on the one hand, no EPTAS could be designed as none exists for unbounded alphabets. On the other hand, proving that no EPTAS exists would be very challenging, as the existing techniques were working even with bounded alphabet sizes, for which an EPTAS in fact exists. Therefore, without significantly new ideas, modifying existing reductions would be futile.

Concerning the NP-Hardness of WLCS, previous work considered a closely related problem, replacing products with sums in the definition of WLCS. In fact this is not unique to WLCS, but rather a paradigm known as the Log-Probability model, often used when studying the hardness of problems on Weighted Sequences. The underlying assumption is that the difference between sums and products is just a technicality. It is then natural to attempt a reduction from the log-probability version to the original problem. To the best of our knowledge, no such general reduction is known and the straightforward ones assume exact computations with reals. In our work, we show how to work directly with products, circumventing the need to go through the log-probability version of the problem. This same set of techniques allowed us to reprove the result of [CKR<sup>+</sup>16] related to the cut-off probabilities without assuming exact computations with reals.

Table 5.1: Results on WLCS.

|                                                           | Amir et al.                                                                                         | Cygan et al.                                                                                 | Our results                                                            |
|-----------------------------------------------------------|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|------------------------------------------------------------------------|
| NP-Hardness of WLCS                                       | Hinted, by NP-Hardness of Log-probability version (Turing reduction - only for unbounded alphabets) | Hinted, by NP-Hardness of Log-probability version (Karp reduction - for alphabets of size 2) | Proved (Karp Reduction - for alphabets of size 2)                      |
| Approximation Algorithms                                  | $\frac{1}{2}$ -Approximation                                                                        | <i>PTAS</i>                                                                                  | <i>EPTAS</i> for bounded alphabets, Improved <i>PTAS</i> for unbounded |
| Proof that no <i>EPTAS</i> exists for unbounded alphabets | No                                                                                                  | No                                                                                           | Yes                                                                    |
| Lower bound on any <i>PTAS</i>                            | No                                                                                                  | No                                                                                           | Matching the upper bound, under <i>ETH</i>                             |
| Reduction to a restricted class of instances              | No                                                                                                  | Yes, by assuming exact computations of logarithms                                            | Yes, without any extra assumptions                                     |

## 5.4 Further research directions

To the best of our knowledge, this is the only work related to Weighted Sequences that drops the Log-Probability Model. It would be very interesting to design a general framework for proving

Table 5.2: Results depending on the Alphabet Size

| Alphabet Size          | Previous Results         | Our results                                                                 |
|------------------------|--------------------------|-----------------------------------------------------------------------------|
| 1                      | Trivial                  | Trivial                                                                     |
| Constant Size          | No <i>FPTAS</i> possible | Achieved <i>EPTAS</i>                                                       |
| Depending on the input | Achieved <i>PTAS</i>     | No <i>EPTAS</i> possible,<br>Improved <i>PTAS</i> ,<br>Matching Lower Bound |

NP-Hardness of problems on Weighted Sequences without resorting to the Log-Probability Model. The understanding we gained by this work hints that non-trivial problems immediately become hard due to the very nature of Weighted Sequences (at least if the alphabet size is large). Given the techniques we used in our work, we believe it is possible to capture this understanding in a clear framework where, assuming a problem satisfies certain simple properties, one can immediately claim NP-Hardness.

Concerning WLCS, most natural questions have been answered by this work. An open problem is the exact classification of the complexity of WLCS in the  $W[]$ -Hierarchy.

# Bibliography

- [ABF<sup>+</sup>99] Richa Agarwala, Vineet Bafna, Martin Farach, Mike Paterson, and Mikkel Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM J. Comput.*, 28(3):1073–1085, 1999. Announced at SODA 1996.
- [AC11] Nir Ailon and Moses Charikar. Fitting tree metrics: Hierarchical clustering and phylogeny. *SIAM J. Comput.*, 40(5):1275–1291, 2011. Announced at FOCS 2005.
- [ACI<sup>+</sup>08] Amihod Amir, Eran Chencinski, Costas S. Iliopoulos, Tsvi Kopelowitz, and Hui Zhang. Property matching and weighted matching. *Theoretical Computer Science*, 395(2-3):298–310, 2008.
- [ACN08] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. Announced in STOC 2005.
- [ADK<sup>+</sup>20] Anders Aamand, Debarati Das, Evangelos Kipouridis, Jakob Bæk Tejs Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup. No repetition: Fast streaming with highly concentrated hashing. *CoRR*, abs/2004.01156, 2020.
- [AGS10] Amihod Amir, Zvi Gotthilf, and B. Riva Shalom. Weighted LCS. *Journal of Discrete Algorithms*, 8(3):273–281, 2010.
- [AGS11] Amihod Amir, Zvi Gotthilf, and B. Riva Shalom. Weighted shortest common supersequence. In *String Processing and Information Retrieval, 18th International Symposium, SPIRE 2011, Pisa, Italy, October 17-21, 2011. Proceedings*, pages 44–54, 2011.
- [AIKP06] Amihod Amir, Costas S. Iliopoulos, Oren Kapah, and Ely Porat. Approximate matching in weighted sequences. In *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, pages 365–376, 2006.
- [AIMP09] Pavlos Antoniou, Costas S. Iliopoulos, Laurent Mouchard, and Solon P. Pissis. Algorithms for mapping short degenerate and weighted sequences to a reference genome. *International Journal of Computational Biology and Drug Design*, 2(4):385–397, 2009.
- [AW21] Sepehr Assadi and Chen Wang. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. *CoRR*, abs/2109.14528, 2021.
- [BBC04] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, 2004.

- [BEK21] Mark Bun, Marek Eliás, and Janardhan Kulkarni. Differentially private correlation clustering. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1136–1146. PMLR, 2021.
- [BIP14] Carl Barton, Costas S. Iliopoulos, and Solon P. Pissis. Optimal computation of all tandem repeats in a weighted sequence. *Algorithms for Molecular Biology*, 9:21, 2014.
- [BKL<sup>+</sup>20] Carl Barton, Tomasz Kociumaka, Chang Liu, Solon P. Pissis, and Jakub Radoszewski. Indexing weighted sequences: Neat and efficient. *Information and Computation*, 270, 2020.
- [BKM<sup>+</sup>17] Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM J. Comput.*, 46(4):1280–1303, 2017.
- [BP18] Carl Barton and Solon P. Pissis. Crochemore’s partitioning on weighted strings and applications. *Algorithmica*, 80(2):496–514, 2018.
- [BSWN15] Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. *ACM Trans. Algorithms*, 11(3), January 2015.
- [CADWN17] Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 962–973. IEEE, 2017.
- [Cav78] James A. Cavender. Taxonomy with confidence. *Mathematical Biosciences*, 40(3):271–280, 1978.
- [CCE13] Sergio Cabello, Erin W Chambers, and Jeff Erickson. Multiple-source shortest paths in embedded graphs. *SIAM Journal on Computing*, 42(4):1542–1571, 2013.
- [CDK<sup>+</sup>21] Vincent Cohen-Addad, Debarati Das, Evangelos Kipouridis, Nikos Parotsidis, and Mikkel Thorup. Fitting distances by tree metrics minimizing the total error within a constant factor. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 468–479. IEEE, 2021.
- [CGMW19] Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Almost optimal distance oracles for planar graphs. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 138–151, 2019.
- [CGW05] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005. Announced in FOCS 2003.
- [CIM<sup>+</sup>06] Manolis Christodoulakis, Costas S. Iliopoulos, Laurent Mouchard, Katerina Perdikuri, Athanasios K. Tsakalidis, and Kostas Tsihlias. Computation of repetitions and regularities of biologically weighted sequences. *Journal of Computational Biology*, 13(6):1214–1231, 2006.

- [CIPR19] Panagiotis Charalampopoulos, Costas S. Iliopoulos, Solon P. Pissis, and Jakub Radoszewski. On-line weighted pattern matching. *Information and Computation*, 266:49–59, 2019.
- [CKMM19] Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. *J. ACM*, 66(4):26:1–26:42, 2019. Announced at SODA 2018.
- [CKP<sup>+</sup>19] Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba. Weighted shortest common supersequence problem revisited. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings*, volume 11811 of *Lecture Notes in Computer Science*, pages 221–238. Springer, 2019.
- [CKR<sup>+</sup>16] Marek Cygan, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Polynomial-time approximation algorithms for weighted LCS problem. *Discrete Applied Mathematics*, 204:38–48, 2016.
- [CLM<sup>+</sup>21] Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2069–2078. PMLR, 2021.
- [CM10] Gunnar E Carlsson and Facundo Mémoli. Characterization, stability and convergence of hierarchical clustering methods. *J. Mach. Learn. Res.*, 11(Apr):1425–1470, 2010.
- [CMSY15] Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 219–228. ACM, 2015.
- [CSE67] L. L. Cavalli-Sforza and A. W. F. Edwards. Phylogenetic analysis models and estimation procedures. *The American Journal of Human Genetics*, 19:233–257, 1967.
- [DEFI06] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2-3):172–187, 2006.
- [Dha04] Kedar Dhamdhere. Approximating additive distortion of embeddings into line metrics. In *APPROX-RANDOM*, pages 96–104, 2004.
- [DKGW22] Debarati Das, Evangelos Kipouridis, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. A simple algorithm for multiple-source shortest paths in planar digraphs. In Karl Bringmann and Timothy Chan, editors, *5th Symposium on Simplicity in Algorithms, SOSA 2022, Virtual Conference, January 10-11, 2022*, pages 68–73. SIAM, 2022.



- [DSST89] James R Driscoll, Neil Sarnak, Daniel D Sleator, and Robert E Tarjan. Making data structures persistent. *Journal of computer and system sciences*, 38(1):86–124, 1989.
- [EK13] David Eisenstat and Philip N Klein. Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 735–744, 2013.
- [Far72] James S. Farris. Estimating phylogenetic trees from distance matrices. *The American Naturalist*, 106(951):645–688, 1972.
- [FHMWN20] Viktor Fredslund-Hansen, Shay Mozes, and Christian Wulff-Nilsen. Truly sub-quadratic exact distance oracles with constant query time for planar graphs. *arXiv preprint arXiv:2009.14716*, 2020.
- [FKP<sup>+</sup>14] Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014.
- [FKW95] Martin Farach, Sampath Kannan, and Tandy J. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13(1/2):155–179, 1995. Announced at STOC 1993.
- [FR06] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006.
- [GK18] Pawel Gawrychowski and Adam Karczmarz. Improved bounds for shortest paths in dense distance graphs. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- [GMWW18] Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 515–529. SIAM, 2018.
- [GSSD08] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *International Workshop on Experimental and Efficient Algorithms*, pages 319–333. Springer, 2008.
- [HKM05] Boulos Harb, Sampath Kannan, and Andrew McGregor. Approximating the best-fit tree under  $l_p$  norms. In *APPROX-RANDOM*, pages 123–133, 2005.
- [HKRS97] Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.
- [IMP<sup>+</sup>04] Costas S. Iliopoulos, Christos Makris, Yannis Panagis, Katerina Perdikuri, Evangelos Theodoridis, and Athanasios K. Tsakalidis. Efficient algorithms for handling molecular weighted sequences. In *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on*

*Theoretical Computer Science (TCS2004)*, 22-27 August 2004, Toulouse, France, pages 265–278, 2004.

- [IMP12] Costas S. Iliopoulos, Mirka Miller, and Solon P. Pissis. Parallel algorithms for mapping short degenerate and weighted DNA sequences to a reference genome. *International Journal of Foundations of Computer Science*, 23(2):249–259, 2012.
- [INSWN11] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, STOC '11*, page 313–322, New York, NY, USA, 2011. Association for Computing Machinery.
- [IPT<sup>+</sup>04] Costas S. Iliopoulos, Katerina Perdikuri, Evangelos Theodoridis, Athanasios K. Tsakalidis, and Kostas Tsichlas. Motif extraction from weighted sequences. In *String Processing and Information Retrieval, 11th International Conference, SPIRE 2004, Padova, Italy, October 5-8, 2004, Proceedings*, pages 286–297, 2004.
- [Kle05] Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 146–155. SIAM, 2005.
- [KST21] Evangelos Kipouridis, Paul G. Spirakis, and Kostas Tsichlas. Threshold-based network structural dynamics. In Tomasz Jurdzinski and Stefan Schmid, editors, *Structural Information and Communication Complexity - 28th International Colloquium, SIROCCO 2021, Wrocław, Poland, June 28 - July 1, 2021, Proceedings*, volume 12810 of *Lecture Notes in Computer Science*, pages 127–145. Springer, 2021.
- [KT06] Jon M. Kleinberg and Éva Tardos. *Algorithm design*. Addison-Wesley, 2006.
- [KT20] Evangelos Kipouridis and Kostas Tsichlas. Longest common subsequence on weighted sequences. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPICs*, pages 19:1–19:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- [LP21] Yaowei Long and Seth Pettie. Planar distance oracles with better time-space trade-offs. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2517–2537. SIAM, 2021.
- [MNW18] Shay Mozes, Yahav Nussbaum, and Oren Weimann. Faster shortest paths in dense distance graphs, with applications. *Theoretical Computer Science*, 711:11–35, 2018.
- [MTG21] Domenico Mandaglio, Andrea Tagarelli, and Francesco Gullo. Correlation clustering with global weight bounds. In Nuria Oliver, Fernando Pérez-Cruz, Stefan Kramer, Jesse Read, and José Antonio Lozano, editors, *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part II*, volume 12976 of *Lecture Notes in Computer Science*, pages 499–515. Springer, 2021.

- [MW17] Benjamin Moseley and Joshua R. Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *NeurIPS*, pages 3094–3103, 2017.
- [MWN10] Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in  $o(n \log^2 n / \log \log n)$  time. In *European Symposium on Algorithms*, pages 206–217. Springer, 2010.
- [MWZ99] Bin Ma, Lusheng Wang, and Louxin Zhang. Fitting distances by tree metrics with increment error. *J. Comb. Optim.*, 3(2-3):213–225, 1999.
- [Mye00] Gene Myers. The whole genome assembly of drosophila. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, page 753, 2000.
- [PM15] Gregory J. Puleo and Olgica Milenkovic. Correlation clustering with constrained cluster sizes and extended weights bounds. *SIAM J. Optim.*, 25(3):1857–1872, 2015.
- [RS20] Jakub Radoszewski and Tatiana Starikovskaya. Streaming  $k$ -mismatch with error correcting and applications. *Information and Computation*, 271:104513, 2020.
- [SS62] Peter H.A. Sneath and Robert R. Sokal. Numerical taxonomy. *Nature*, 193(4818):855–860, 1962.
- [SS63] Peter H.A. Sneath and Robert R. Sokal. *Numerical Taxonomy. The Principles and Practice of Numerical Classification*. Freeman, 1963.
- [Tho04] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM (JACM)*, 51(6):993–1024, 2004.
- [Vel21] Nate Veldt. Faster deterministic approximation algorithms for correlation clustering and cluster deletion. *CoRR*, abs/2111.10699, 2021.
- [Ven02] J. Craig Venter. Sequencing the human genome. In *RECOMB*, pages 309–309. ACM, 2002.
- [vZW09] Anke van Zuylen and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Math. Oper. Res.*, 34(3):594–620, 2009. Announced at SODA '07.
- [WSSB77] M.S. Waterman, T.F. Smith, M. Singh, and W.A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64(2):199–213, 1977.
- [ZGFI10] Hui Zhang, Qing Guo, Jing Fan, and Costas S. Iliopoulos. Loose and strict repeats in weighted sequences of proteins. *Protein and Peptide Letters*, 17(9):1136–1142, 2010.
- [ZGI04] Hui Zhang, Qing Guo, and Costas S. Iliopoulos. String matching with swaps in a weighted sequence. In *CIS*, volume 3314 of *Lecture Notes in Computer Science*, pages 698–704. Springer, 2004.
- [ZGI10a] Hui Zhang, Qing Guo, and Costas S. Iliopoulos. An algorithmic framework for motif discovery problems in weighted sequences. In *Algorithms and Complexity, 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings*, pages 335–346, 2010.

- [ZGI10b] Hui Zhang, Qing Guo, and Costas S. Iliopoulos. Varieties of regularities in weighted sequences. In *AAIM*, volume 6124 of *Lecture Notes in Computer Science*, pages 271–280. Springer, 2010.
- [ZWWZ09] Yuzhou Zhang, Jianyong Wang, Yi Wang, and Lizhu Zhou. Parallel community detection on large networks with propinquity dynamics. In John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki, editors, *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 997–1006. ACM, 2009.

## Appendix A

# FOCS: Fitting Distances by Tree Metrics Minimizing the Total Error within a Constant Factor

# Fitting Distances by Tree Metrics Minimizing the Total Error within a Constant Factor

Vincent Cohen-Addad\*    Debarati Das<sup>†</sup>    Evangelos Kipouridis<sup>†‡</sup>    Nikos Parotsidis\*  
Mikkel Thorup<sup>†</sup>


## Abstract


We consider the numerical taxonomy problem of fitting a positive distance function  $\mathcal{D} : \binom{S}{2} \rightarrow \mathbb{R}_{>0}$  by a tree metric. We want a tree  $T$  with positive edge weights and including  $S$  among the vertices so that their distances in  $T$  match those in  $\mathcal{D}$ . A nice application is in evolutionary biology where the tree  $T$  aims to approximate the branching process leading to the observed distances in  $\mathcal{D}$  [Cavalli-Sforza and Edwards 1967]. We consider the total error, that is the sum of distance errors over all pairs of points. We present a deterministic polynomial time algorithm minimizing the total error within a constant factor. We can do this both for general trees, and for the special case of ultrametrics with a root having the same distance to all vertices in  $S$ .

The problems are APX-hard, so a constant factor is the best we can hope for in polynomial time. The best previous approximation factor was  $O((\log n)(\log \log n))$  by Ailon and Charikar [2005] who wrote “Determining whether an  $O(1)$  approximation can be obtained is a fascinating question”.

---

\*Google Research, Switzerland. Email: {cohenaddad,nikosp}@google.com

<sup>†</sup>Department of Computer Science, University of Copenhagen. Research of this author is supported by VILLUM Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), Denmark. Email: {debaratix710,mikkel2thorup}@gmail.com, kipouridis@di.ku.dk 

<sup>‡</sup>Evangelos Kipouridis has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 801199. 

# 1 Introduction

Taxonomy or hierarchical classification of species goes back at least to discussions between Aristotle and his teacher Plato<sup>1</sup> (~350BC) while modern taxonomy is often attributed to Linnaeus<sup>2</sup> (~1750). The discussions of evolution in the nineteenth century, clarified the notion of evolutionary trees, or phylogenies, and the notion that species were close due to a common past ancestor. Such evolutionary trees are seen in the works of Hitchcock<sup>3</sup> (1840) and Darwin<sup>4</sup> (1859). Viewing the descendants of each node as a class, the evolutionary tree induces a hierarchical classification.

In the 1960s came the interest in computing evolutionary trees based on present data, the so-called numerical taxonomy problem [11, 49, 50]. Our focus is on the following simple model by Cavalli-Sforza and Edwards from 1967 [11]. In an evolutionary tree, let the edge between the child and its parent be weighted by the evolutionary distance between them. Then the evolutionary distance between any two species is the sum of weights on the unique simple path between them. We note that the selection of the root plays no role for the distances. What we are saying is that any tree with edge weights, induces distances between its nodes; a so-called tree metric assuming that all weights are positive.

We now have the converse reconstruction problem of numerical taxonomy [11, 49, 50]: given a set  $S$  of species with measured distances between them, find a tree metric matching those observed distances on  $S$ . Thus we are looking for an edge-weighted tree  $T$  which includes  $S$  among its nodes with the right distances between them. Importantly,  $T$  may have nodes not in  $S$  representing ancestors explaining proximity between different species. The better the tree metric  $T$  matches the measured distances on  $S$ , the better the tree  $T$  explains these measured distances.

**Other applications** This very basic reconstruction problem also arises in various other contexts. First, concerning the evolutionary model, it may be considered too simplistic to just add up distances along the paths in the tree. Some changes from parent to child could be reverted for a grandchild. Biologists [12, 34] have suggested stochastic models for probabilistic changes that also have a chance of being reverted further down. However, Farach and Kannan [32] have shown that applying logarithms appropriately, we can convert estimated distances into some other distances for which we find a matching tree metric that we can then convert back into a maximum likelihood stochastic tree. The basic point is that finding tree metrics can be used as powerful tool to invert evolution even in cases where tree metric model does not apply directly.

Obviously, the numerical taxonomy problem is equally relevant to other historical sciences with an evolutionary branching process leading to evolutionary distances, e.g., historical linguistics.

More generally, if we can approximate distances with a tree metric, then the tree of this metric provides a very compact and convenient representation that is much easier to navigate than a general distance function. Picking any root, the tree induces a distance based hierarchical classification, and referring to the discussions between Plato and Aristotle's, humans have been interested in such hierarchical classifications since ancient time.

It is not just humans but also computers that understand trees and tree metrics much better than general metrics. Many questions that are NP-hard to answer in general can be answered very

---

<sup>1</sup><https://iep.utm.edu/classifi/>, Internet Encyclopedia of Philosophy

<sup>2</sup><https://britannica.com/science/taxonomy/The-Linnaean-system>

<sup>3</sup>[https://en.wikipedia.org/wiki/Edward\\_Hitchcock](https://en.wikipedia.org/wiki/Edward_Hitchcock)

<sup>4</sup>[https://en.wikipedia.org/wiki/On\\_the\\_Origin\\_of\\_Species](https://en.wikipedia.org/wiki/On_the_Origin_of_Species)

efficiently based on trees (see, e.g., Chapter 10.2 “Solving NP-Hard Problems on Trees” in the text book [41]).

Computing “good” tree representations is nowadays also a major tool to learn from data. In this context, we are sometimes interested in a special kind of tree metrics, called *ultrametrics*, defined by rooted trees whose sets of leaves is  $S$  and where the leaf-to-root distance is the same for all points in  $S$ . Equivalently, an ultrametric is a metric so that for any three points  $i, j, k$ , the distance from  $i$  to  $j$  is no bigger than the maximum of the distance from  $i$  to  $j$  and the distance from  $j$  to  $k$  <sup>5</sup>.

An ultrametric can be seen as modeling evolution that is linear over time. This may be not the case in biology where the speed of evolution depends on the local evolutionary pressure for example. However, ultrametrics are key objects in machine learning and data analysis, see e.g.: [10], and there are various algorithms for embedding arbitrary metrics into ultrametrics such as the popular “linkage” algorithms (single, complete or average linkage), see also [24, 45].

## 1.1 Tree fitting (Numerical Taxonomy Problem)

Typically our measured distances do not have an exact fit with any tree metric. We then have the following generic optimization problem for any  $L_p$ -norm:

**Problem:**  $L_p$ -fitting tree (ultra) metrics

**Input:** A set  $S$  with a distance function  $\mathcal{D} : \binom{S}{2} \rightarrow \mathbb{R}_{>0}$ . <sup>6</sup>

**Desired Output:** A tree metric (or ultrametric)  $T$  that spans  $S$  and fits  $\mathcal{D}$  in the sense of minimizing the  $L_p$ -norm

$$\|T - \mathcal{D}\|_p = \left( \sum_{\{i,j\} \in \binom{S}{2}} |\text{dist}_T(i, j) - \mathcal{D}(i, j)|^p \right)^{1/p}. \quad (1)$$

Cavalli-Sforza and Edwards [11] introduced this numerical taxonomy problem for both tree and ultrametrics in the  $L_2$ -norm in 1967. Farris suggested using  $L_1$ -norm in 1972 [34, p. 662].

## 1.2 Our result

In this paper we focus on the  $L_1$ -norm, that is, the total sum of errors. The problem is APX-hard, both for tree metrics and ultrametrics (see Section 9 and also [3]), so a constant approximation factor is the best we can hope for in polynomial time. The best previous approximation factor for both tree metrics and ultrametrics was  $O((\log n)(\log \log n))$  by Ailon and Charikar [3].

In this paper we present a deterministic polynomial time constant factor approximation both for tree metrics and for ultrametrics, that is, in both cases, we can find a tree  $T$  minimizing the  $L_1$ -norm within a constant factor of the best possible.

Thus, we will prove the following theorem.

**Theorem 1.** *The  $L_1$ -fitting tree metrics problem can be solved in deterministic polynomial time within a constant approximation factor. The same holds for the  $L_1$ -fitting ultrametrics problem.*

<sup>5</sup>[https://en.wikipedia.org/wiki/Ultrametric\\_space](https://en.wikipedia.org/wiki/Ultrametric_space).

<sup>6</sup> $\binom{S}{k}$  denotes all subsets of  $S$  of size  $k$ .



### 1.3 History of $L_p$ tree fitting

Since Cavalli-Sforza and Edwards introduced the tree fitting problem, the problem has collected an extensive literature. In 1977 [54], it was shown that if there is a tree metric coinciding exactly with  $\mathcal{D}$ , it is unique and it can be found in time linear in the input size, i.e.,  $O(|S|^2)$  time. The same then also holds trivially for ultrametrics. Unfortunately there is typically no tree metric coinciding exactly with  $\mathcal{D}$ , and in 1987 [27] it was shown that for  $L_1$  and  $L_2$  the numerical taxonomy problem is NP-hard, both in the tree metric and the ultrametric cases. The problems are in fact APX-hard (see Section 9), which rules out the possibility of a polynomial-time approximation scheme. Thus, a constant factor, like ours for  $L_1$ , is the best one can hope for from a complexity perspective for these problems.

For the  $L_\infty$  numerical taxonomy problem, there was much more progress. In 1993 [33] it was shown that for the ultrametric case an optimal solution can be found in time proportional to the number of input distance pairs (i.e.: the number of entries in  $S$ ). More recently, it was shown that when the points are embedded into  $\mathbb{R}^d$  and the distances are given by the pairwise Euclidean distances, the problem can be approximated in subquadratic time [25, 22]. For the general trees case (still in the  $L_\infty$ -norm objective), [2] gave an  $O(|S|^2)$  algorithm that produces a constant factor approximation and proved that the problem is APX-hard (unlike the ultrametric case).

The technical result from [2] was a general reduction from general tree metrics to ultrametrics. It modifies the input distance matrix and asks for fitting this new input with an ultrametric that can later be converted to a tree metric for the original distance matrix. The result states that for any  $p$ , if we can minimize the restricted ultrametric  $L_p$  error within a factor  $\alpha$  in polynomial-time, then there is a polynomial-time algorithm that minimizes the tree metric  $L_p$  error within a factor  $3\alpha$ . The reduction from [2] imposes a certain restriction on the ultrametric, but the restriction is not problematic and in Section 8, we will even argue that the restriction can be completely eliminated with a slightly modified reduction. With  $n$  species, the reduction from tree metrics to ultrametrics can be performed in time  $O(n^2)$ . Applying this to the optimal ultrametric algorithm from [33] for the  $L_\infty$ -norm objective yielded a factor 3 for general metrics for the  $L_\infty$ -norm objective. The generic impact is that *for any  $L_p$ , later algorithms only had to focus on the ultrametric case to immediately get results for the often more important tree metrics case, up to losing a factor 3 in the approximation guarantee.* Indeed, the technical result of this paper is a constant factor approximation for ultrametric. Thus, when it comes to approximation factors, we have

$$\text{TreeMetric} \leq 3 \cdot \text{UltraMetric}$$

For  $L_p$  norms with constant  $p$ , the developments have been much slower. Ma et al. [43] considered the problem of finding the best  $L_p$  fit by an ultrametric where distances in the ultrametric are no smaller than the input distances. For this problem, they obtained an  $O(n^{1/p})$  approximation.

Later, Dhamdhare [28] considered the problem of finding a line metric to minimize additive distortion from the given data (measured by the  $L_1$  norm) and obtained an  $O(\log n)$  approximation. In fact, his motivation for considering this problem was to develop techniques that might be useful for finding the closest tree metric with distance measured by the  $L_1$  norm. Harb, Kannan and McGregor [39] developed a factor  $O(\min\{n, k \log n\}^{1/p})$  approximation for the closest ultrametric under the  $L_p$  norm where  $k$  is the number of distinct distances in the input<sup>7</sup>.

---

<sup>7</sup>The authors erroneously claim that they get the same approximation for the closest tree metric problem. However, the known reduction may create  $\omega(k)$  distinct distances.

The best bounds known for the ultrametric variant of the problem are due to Ailon and Charikar [3]. They first focus on ultrametrics in  $L_1$  and show that if the distance matrix has only  $k$  distinct distances, then it is possible to approximate the  $L_1$  error within a factor  $k+2$ . Next they obtain an LP-based  $O((\log n)(\log \log n))$  approximation for arbitrary distances matrices. Finally they sketch how it can be generalized to an  $O((\log n)(\log \log n))^{1/p}$  approximation of the  $L_p$  error for any  $p$ . Using the reduction from [2], they also get an  $O((\log n)(\log \log n))^{1/p}$  approximation for tree metrics under the  $L_p$ -norm objective. The  $O((\log n)(\log \log n))^{1/p}$  approximation comes from an  $O((\log n)(\log \log n))$  approximation of the  $p$ 'th moment  $F_p$ :

$$\|T - \mathcal{D}\|_p^p = \left( \sum_{\{i,j\} \in \binom{S}{2}} |\text{dist}_T(i,j) - \mathcal{D}(i,j)|^p \right). \quad (2)$$

Technically, Ailon and Charikar [3] present a simple LP relaxation for  $L_1$  ultrametric fitting—an LP that will also be used in our paper. They get their  $O((\log n)(\log \log n))$  approximation using an LP rounding akin to the classic  $O(\log n)$  rounding of Leighton and Rao for multicut [42]. The challenge is to generalize the approach to deal with the hierarchical issues associated with ultrametric and show that this can be done paying only an extra factor  $O(\log \log n)$  in the approximation factor. Next they show that their LP formulation and rounding is general enough to handle different variants, including other  $L_p$  norms as mentioned above, but also they can handle the *weighted case*, where for each pair of species  $i, j$ , the error contribution to the overall error is multiplied by a value  $w_{ij}$ . However, this weighted problem captures the multicut problem (and the weighted minimization correlation clustering problem) [3]. Since the multicut cannot be approximated within a constant factor assuming the unique games conjecture [19] and the best known approximation bound remains  $O(\log n)$ , it is beyond reach of current techniques to do much better in these more general settings.

Ailon and Charikar [3] conclude that “Determining whether an  $O(1)$  approximation can be obtained is a fascinating question. The LP formulation used in our [their] work could eventually lead to such a result”. For their main LP formulation for the (unweighted)  $L_1$  ultrametric fitting, the integrality gap was only known to be somewhere between 2 and  $O((\log n)(\log \log n))$ . To break the  $\log n$ -barrier we must come up with a radically different way of rounding this LP and free ourselves from the multicut-inspired approach.

For  $L_1$  ultrametric fitting, we give the first constant factor approximation, and we show this can be obtained by rounding the LP proposed by Ailon and Charikar, thus demonstrating a constant integrality gap for their LP. Our solution breaks the  $\log n$  barrier using the special combinatorial structure of the  $L_1$  problem.

Stepping a bit back, having different solutions for different norms should not come as a surprise. As an analogue, take the generic problem of placing  $k$  facilities in such a way that each of  $n$  cities is close to the nearest facility. Minimizing the vector of distances in the  $L_1$  norm is called the  $k$ -median problem. In the  $L_2$  norm it is called the  $k$ -means problem, and in the  $L_\infty$ -norm is called the  $k$ -center problems. Indeed, while the complexity of the  $k$ -center problem has been settled in the mid-1980s thanks to Gonzalez’ algorithm [37], it has remained a major open problem for the next 15 years to obtain constant factor approximation for the  $k$ -median and the  $k$ -means problems. Similarly, our understanding of the  $k$ -means problems ( $L_2$ -objective) remains poorer than our understanding of the  $k$ -median problem, and the problem is in fact provably harder (no better than  $1 + 8/e$ -approximation algorithm [38] while  $k$ -median can be approximated within a factor 2.675 at the

| Norm        | $L_1$       | $L_p, p < \infty$                  | $L_\infty$  |
|-------------|-------------|------------------------------------|-------------|
| Treemetric  | $\Theta(1)$ | $O(((\log n)(\log \log n))^{1/p})$ | $\Theta(1)$ |
| Ultrametric | $\Theta(1)$ | $O(((\log n)(\log \log n))^{1/p})$ | 1           |

Table 1: Tree fitting approximation factors.

moment [9]).

For our tree fitting problem, the  $L_\infty$  norm has been understood since the 1990s, and our result shows that the  $L_1$  norm admits a constant factor approximation algorithm. The current status of affairs for tree and ultrametrics is summarized in Table 1. The status for  $L_p$  tree fitting is that we have good constant factor approximation if we want to minimize the total error  $L_1$  or the maximal error  $L_\infty$ . For all other  $L_p$  norms, we only have the much weaker but general  $O(((\log n)(\log \log n))^{1/p})$  approximation from [3]. In particular, we do not know if anything better is possible with  $L_2$ . The difference is so big that even if we are in a situation where we would normally prefer an  $L_2$  approximation, our much better approximation guarantee with  $L_1$  might be preferable.

#### 1.4 Other related work

**Computational Biology.** Researchers have also studied reconstruction of phylogenies under stochastic models of evolution (see Farach and Kannan [32] or Mossel et al. [46] and the references therein, see also Henzinger et al. [40]).

Finally, related to our hierarchical correlation clustering problem is the hierarchical clustering problem introduced by Dasgupta [26] where the goal is, given a similarity matrix, to build a hierarchical clustering tree where the more similar two points are, the lower in the tree they are separated (formally, a pair  $(u, v)$  induces a cost of  $\text{similarity}(u, v)$  times the size of the minimal subtree containing both  $u$  and  $v$ , the goal is to minimize the sum of the costs of the pairs). This has received a lot of attention in the last few years ([5, 14, 15, 16, 18, 23, 24, 44, 47], see also [1, 6, 13, 21]), but differs from our settings since the resulting tree may not induce a metric space.

**Metric Embeddings.** There is a large body of work of metric embedding problems. For example, the metric violation distance problem asks to embed an arbitrary distance matrix into a metric space while minimizing the  $L_0$ -objective (i.e.: minimizing the number of distances that are not preserved in the metric space). The problem is considerably harder and is only known to admit an  $O(\text{OPT}^{1/3})$ -approximation algorithm [30, 31, 35] while no better than a 2 hardness of approximation is known. More practical results on this line of work includes [51] and [36]. Sidiropoulos et al [48] also considered the problem of embedding into metric, ultrametric, etc. while minimizing the total number of outlier points.

There is also a rich literature on metric embedding problems where the measure of interest is the multiplicative distortion, and the goal of the problem is to approximate the absolute distortion of the metric space (as opposed to approximating the optimal embedding of the metric space). Several such problems have been studied in the context of approximating metric spaces via tree metrics (e.g. [8, 29]). The objective of these works is very different since they are focused on the absolute expected multiplicative distortion over all input metrics while we aim at approximating

the *optimal* expected additive distortion for each individual input metric.

While these techniques have been very successful for designing approximation algorithms for various problems in a variety of contexts, they are not aimed at numerical taxonomy. Their goal is to do something for general metrics. However, for our tree-fitting problem, the idea is that the ground-truth is a tree, e.g., a phylogeny, and that the distances measured, despite noise and imperfection of the model, are close to the metric of the true tree. To recover an approximation to the true tree, we therefore seek a tree that compares well against the best possible fit of a tree metric.

## 1.5 Techniques

We will now discuss the main idea of our algorithm. Our solution will move through several combinatorial problems that code different aspects of the  $L_1$ -fitting of ultrametrics, but which do not generalize nicely to other norms.

Our result follows from a sequence of constant-factor-approximation reductions between problems. To achieve our final goal, we introduce several new problems that have a key role in the sequence of reductions. Some of the reductions and approximation bounds have already been extensively studied (e.g.: Correlation Clustering). A roadmap of this sequence of results is given in Figure 1.

### 1.5.1 Correlation Clustering

Our algorithms will use a subroutine for what is known as the unweighted minimizing-disagreements correlation clustering problem on complete graphs [7]. We simply refer to this problem as Correlation Clustering throughout the paper.

First, for any family  $P$  of disjoint subsets of  $S$ , let

$$\mathcal{E}(P) = \bigcup_{T \in P} \binom{T}{2}$$

Thus  $\mathcal{E}(P)$  represents the edge sets over an isolated clique over each set  $T$  in  $P$ . Often  $P$  will be a partition of  $S$ , that is,  $\bigcup P = S$ .

The *correlation clustering* takes as input an edge set  $E \subseteq \binom{S}{2}$  and seeks a partition  $P$  minimizing

$$|E \Delta \mathcal{E}(P)|,$$

where  $\Delta$  denotes symmetric difference. It is well-known that correlation clustering is equivalent to ultrametric fitting with two distances (see, e.g., [39]).

A randomized polynomial time  $2.06 + \epsilon$  factor approximation from [20] (see also [4]) and a 2.5 deterministic approximation algorithm [52] are known. We shall use this as a subroutine with approximation factor

$$\text{CorrClust} = O(1) \tag{D} \text{ from Figure 1}$$

We note that Ailon and Charikar, who presented the previous best  $O((\log n)(\log \log n))$  approximation for tree metrics and ultrametrics at FOCS'05 [3] had presented a 2.5 approximation for correlation clustering at the preceding STOC'05 with Newman [4]. In fact, inspired by this connection they proposed in [3] a pivot-based  $(M + 2)$ -approximation algorithm for the  $L_1$  ultrametric problem where  $M$  is the number of distinct input distances.

| Problem                                             | Introduced in          |
|-----------------------------------------------------|------------------------|
| TreeMetric                                          | Beginning of Section 1 |
| UltraMetric                                         | Beginning of Section 1 |
| Correlation Clustering (CorrClust)                  | Section 1.5.1          |
| Hierarchical Correlation Clustering (HierCorrClust) | Section 1.5.2          |
| Hierarchical Cluster Agreement (HierClustAgree)     | Section 1.5.3          |

$$\begin{aligned} \text{TreeMetric} &\leq (3 + o(1)) \cdot \text{UltraMetric} && \text{(A)} \\ \text{UltraMetric} &\leq \text{HierCorrClust} && \text{(B)} \\ \text{HierCorrClust} &< (\text{CorrClust} + 1)(\text{HierClustAgree} + 1) && \text{(C)} \\ \text{CorrClust} &= O(1) && \text{(D)} \\ \text{HierClustAgree} &= O(1) && \text{(E)} \end{aligned}$$

**Approximation factors.** Abbreviated problem names used as approximation factors.

- 1: **procedure** TreeMetric( $S, \mathcal{D}$ ) ▷ See Section 8
- 2:     Reduction to UltraMetric based on [2]
- 3:
- 4: **procedure** UltraMetric( $S, \mathcal{D}$ ) ▷ See Section 7
- 5:     Reduction to HierCorrClust based on [3, 39]
- 6:
- 7: **procedure** HierCorrClust( $S, E^{(*)}, \delta^{(*)}$ ) ▷ NEW. See Section 3
- 8:     **for**  $t \in [\ell]$  **do**  $Q^{(t)} \leftarrow \text{CorrClust}(E^{(t)})$
- 9:     **return** HierClustAgree( $S, Q^{(*)}, \delta^{(*)}$ )
- 10:
- 11: **procedure** CorrClust( $S, E$ ) ▷ See Section 1.5.1
- 12:     Use Algorithm from [7]
- 13:
- 14: **procedure** HierClustAgree( $S, Q^{(*)}, \delta^{(*)}$ ) ▷ NEW. See Section 4
- 15:      $x^{(*)} \leftarrow \text{Solve}(\text{LP-relaxation}(S, Q^{(*)}, \delta^{(*)}))$
- 16:      $L^{(*)} \leftarrow \text{LP-Cleaning}(S, Q^{(*)}, x^{(*)})$
- 17:     **return** Derive-Hierarchy( $S, L^{(*)}$ )

Figure 1: Roadmap leading to our result for  $L_1$ -fitting tree metrics.

### 1.5.2 Hierarchical correlation clustering

We are going to work with a generalization of the problem of  $L_1$ -fitting ultrametric which is implicit in previous work [3, 39], but where we will exploit the generality in new interesting ways.

**Problem** Hierarchical Correlation Clustering.

**Input** The input is  $\ell$  weights  $\delta^{(1)}, \dots, \delta^{(\ell)} \in \mathbb{R}_{>0}$  and  $\ell$  edge sets  $E^{(1)}, \dots, E^{(\ell)} \subseteq \binom{S}{2}$ .

**Desired output**  $\ell$  partitions  $P^{(1)}, \dots, P^{(\ell)}$  of  $S$  that are hierarchical in the sense that  $P^{(t)}$  subdivides  $P^{(t+1)}$  and such that we minimize

$$\sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(P^{(t)})| \quad (3)$$

Thus we are having a combination of  $\ell$  correlation clustering problems where we want the output partitions to form a hierarchy, and where the objective is to minimize a weighted sum of the costs for each level problem.

We shall review the reduction from  $L_1$ -fitting of ultrametrics to hierarchical correlation clustering in Section 7. The instances we get from ultrametrics will always satisfy  $E^{(1)} \subseteq \dots \subseteq E^{(\ell)}$ , but as we shall see shortly, our new algorithms will reduce to instances where this is not the case, even if the original input is from an ultrametric.

### 1.5.3 Hierarchical cluster agreement

We will be particularly interested in the following special case of Hierarchical Correlation Clustering.

**Problem** Hierarchical Cluster Agreement.

**Input** The input is  $\ell$  weights  $\delta^{(1)}, \dots, \delta^{(\ell)} \in \mathbb{R}_{>0}$  and  $\ell$  partitions  $Q^{(1)}, \dots, Q^{(\ell)}$  of  $S$ .

**Desired output**  $\ell$  partitions  $P^{(1)}, \dots, P^{(\ell)}$  of  $S$  that are hierarchical in the sense that  $P^{(t)}$  subdivides  $P^{(t+1)}$  and such that we minimize

$$\sum_{t=1}^{\ell} \delta^{(t)} |\mathcal{E}(Q^{(t)}) \Delta \mathcal{E}(P^{(t)})| \quad (4)$$

This is the special case of hierarchical correlation clustering, where the input edge set  $E^{(t)}$  are the disjoint clique edges from  $\mathcal{E}(Q^{(t)})$ . The challenge is that the input partitions may disagree in the sense that  $Q^{(t)}$  does not subdivide  $Q^{(t+1)}$ , or equivalently,  $\mathcal{E}(Q^{(t)}) \not\subseteq \mathcal{E}(Q^{(t+1)})$ , so now we have to find the best hierarchical agreement.

We are not aware of any previous work on hierarchical cluster agreement, but it plays a central role in our hierarchical correlation clustering algorithm, outlined below.

## 1.6 High-level algorithm for hierarchical correlation clustering

Our main technical contribution in this paper is solving Hierarchical Correlation Clustering. Reductions from Ultrametric to Hierarchical Correlation Clustering, and from general Tree Metric to Ultrametric are already known from [3, 39] and [2] respectively. We discuss both reductions in Sections 7 and 8. This includes removing some restrictions in the reduction from Tree Metrics to Ultrametrics.

Focusing on Hierarchical Correlation Clustering, our input is the  $\ell$  weights  $\delta^{(1)}, \dots, \delta^{(\ell)} \in \mathbb{R}_{>0}$  and  $\ell$  edge sets  $E^{(1)}, \dots, E^{(\ell)} \subseteq \binom{S}{2}$ .

**Step 1: Solve correlation clustering independently for each level** The first step in our solution is to solve the correlation clustering problem defined by  $E^{(t)}$  for each level  $t = 1, \dots, \ell$  independently, thus obtaining an intermediate partitioning  $Q_t$ . As we mentioned in Section 1.5.1, this can be done so that  $Q_t$  minimizes  $|E^{(t)} \Delta E(Q_t)|$  within a constant factor.

**Step 2: Solve hierarchical cluster agreement** We now use the  $\ell$  weights  $\delta^{(1)}, \dots, \delta^{(\ell)} \in \mathbb{R}_{>0}$  and  $\ell$  partitions  $Q^{(1)}, \dots, Q^{(\ell)}$  of  $S$  as input to the hierarchical cluster agreement problem, which we solve using an LP very similar to the one Ailon and Charikar [3] used to solve general hierarchical correlation clustering. However, when applied to the special case of hierarchical cluster agreement, it allows a special simple LP rounding where the LP decides which sets from the input partitions are important to the hierarchy, and which sets can be ignored. Having decided the important sets, it turns out that a very simple combinatorial algorithm can generate the hierarchical output partitions  $P^{(1)}, \dots, P^{(\ell)}$  bottom-up. The result is a poly-time constant factor approximation for hierarchical cluster agreement, that is

$$\text{HierClustAgree} = O(1)$$

The output partitions  $P^{(1)}, \dots, P^{(\ell)}$  are also returned as output to the original hierarchical correlation clustering problem.

We now provide a high level overview and the intuition behind the hierarchical cluster agreement algorithm. The algorithm can be broadly divided into two parts.

**LP cleaning.** We start by optimally solving the LP based on the weights  $\delta^{(1)}, \dots, \delta^{(\ell)}$  and partitions  $Q^{(1)}, \dots, Q^{(\ell)}$ . For each level  $t \in \{1, \dots, \ell\}$ , we naturally think of the relevant LP variables as distances, and call them LP distances. That is because a small value means that the LP wants the corresponding species to be in the same part of the output partition at level  $t$ , and vice versa, while the LP constraints also enforce the triangle inequality. The weights  $\delta^{(1)}, \dots, \delta^{(\ell)}$  impact the optimal LP variables, but will otherwise not be used in the rest of the algorithm.

Using the LP distances, we clean each set in every partition  $Q^{(t)}$  independently. The objective of this step (LP-Cleaning - Algorithm 2) is to keep only the sets whose species are very close to each other and far away from the species not in the set. All other sets are disregarded. Even though this is not a binary decision, it can be thought of as one, since the algorithm may only slightly modify each surviving set. The property that we can clean each set independently to decide whether it is important or not, without looking at any other sets makes this part of our algorithm quite simple.

Omitting exact thresholds for simplicity, the algorithm works as follows. We process each set  $C_I \in Q^{(t)}$  by keeping only those species that are at very small LP distance from at least half of the other species in  $C_I$  and at large LP distance to almost all the species outside  $C_I$ . Let us note that by triangle inequality and the pigeonhole principle, all species left in a set are at relatively small distance from each other. After this cleaning process, we only keep a set if at least 90% of its species are still intact, and we completely disregard it otherwise. The LP cleaning algorithm outputs the sequence  $L^{(*)} = (L^{(1)}, \dots, L^{(\ell)})$  where  $L^{(t)}$  is the family of surviving cleaned sets from  $Q^{(t)}$ .

**Derive hierarchy.** Taking  $L^{(*)}$  as input, in the next step the algorithm Derive-Hierarchy (Algorithm 3) computes a hierarchical partition  $P^{(*)} = (P^{(1)}, \dots, P^{(\ell)})$  of  $S$ . This algorithm works bottom-up while initializing an auxiliary bottom most level of the hierarchy with  $|S|$  sets where each set is a singleton and corresponds to a species of  $S$ . Then the algorithm performs  $\ell$  iterations where at the  $t$ -th iteration it processes all the disjoint sets in  $L^{(t)}$  and computes partition  $P^{(t)}$  while

ensuring that at the end of the iteration  $P^{(1)}, \dots, P^{(t)}$  are hierarchical. An interesting feature of our algorithm is that, once created, no further computation processing the upper levels can modify the already created partitions. Next, we discuss how to compute  $P^{(t)}$  given  $L^{(t)}$  and all the lower level sets in partitions  $P^{(1)}, \dots, P^{(t-1)}$ .

Consider a set  $C_{LP} \in L^{(t)}$ . Now if for each lower level set  $C'$ , either  $C' \cap C_{LP} = \emptyset$  or  $C' \subseteq C_{LP}$ , then introducing  $C_{LP}$  at level  $t$  does not violate the hierarchy property. Otherwise let  $C'$  be a lower level set such that  $C' \cap C_{LP} \neq \emptyset$  and  $C' \not\subseteq C_{LP}$ . Note that we already mentioned, once created,  $C'$  is never modified while processing upper level sets. Thus, to ensure the hierarchy condition, the algorithm can either extend  $C_{LP}$  so that it completely covers  $C'$  or can discard the common part from  $C_{LP}$ .

In the process of modifying  $C_{LP}$  (where we can add or discard some species from it), at any point we define the core of  $C_{LP}$  to be the part that comes from the initial set. Now to resolve the conflict between  $C_{LP}$  and  $C'$  we work as follows. If the core of  $C_{LP}$  intersects the core of  $C'$  then we extend  $C_{LP}$  so that  $C'$  becomes a subset of it. Omitting technical details, there are two main ideas here: first, we ensure that the number of species in  $C'$  (resp.  $C_{LP}$ ) that are not part of its core is negligible with respect to the size of  $C'$  (resp.  $C_{LP}$ ). Furthermore, since the cores of  $C_{LP}, C'$  have at least one common species, using triangular inequality we can claim that any pair of species from the cores of  $C', C_{LP}$  also have small LP distance; therefore, nearly all pairs of species in  $C_{LP}, C'$  have small LP distance, meaning that the extension of  $C_{LP}$  is desirable (i.e. its cost is within a constant factor from the LP cost while it ensures the hierarchy).

Here we want to emphasize the point that because of the LP-cleaning, we can ensure that for any lower level set  $C'$  at level  $t$  there exists at most one set whose core has an intersection with the core of  $C'$ . We call this the *hierarchy-friendly* property of the LP cleaned sets. This property is crucial for consistency, as it ensures that at level  $t$  there cannot exist more than one sets that are allowed to contain  $C'$  as a subset.

In the other case, where the cores of  $C_{LP}$  and  $C'$  do not intersect, the algorithm removes  $C_{LP} \cap C'$  from  $C_{LP}$ . The analysis of this part is more technical but follows the same arguments, namely using the aforementioned properties of LP-cleaning along with triangle inequality.

After processing all the sets in  $L^{(t)}$ , the algorithm naturally combines these processed sets with  $P^{(t-1)}$  to generate  $P^{(t)}$ , thus ensuring that  $P^{(1)}, \dots, P^{(t)}$  are hierarchical.

**High-level analysis** We will prove that the partitions  $P^{(1)}, \dots, P^{(\ell)}$  solves the original hierarchical clustering problem within a constant factor.

Using triangle inequality, we are going to show that the switch in Step 1, from the input edge sets  $E^{(1)}, \dots, E^{(\ell)}$  to the partitions  $Q^{(1)}, \dots, Q^{(\ell)}$  costs us no more than the approximation factor of correlation clustering used to generate each partition. This then becomes a multiplicative factor on our approximation factor for hierarchical cluster agreement, more specifically,

$$\text{HierCorrClust} < (\text{CorrClust} + 1)(\text{HierClustAgree} + 1)$$

We will even show that we can work with the LP from [3] for the original hierarchical correlation clustering problem, and get a solution demonstrating a constant factor integrality gap.

## 1.7 Organization of the paper

In Section 2 we present the LP formulation and related definitions for Hierarchical Correlation Clustering. In Section 3 we show how to reduce Hierarchical Correlation Clustering to Hierarchical Clus-



ter Agreement. In Section 4 we present the algorithm for Hierarchical Cluster Agreement, and in Section 5 we analyze it. In Section 6 we show that the LP formulation for Hierarchical Correlation Clustering has constant integrality gap. In Section 7 we show how  $L_p$ -fitting ultrametrics reduces to Hierarchical Correlation Clustering. In Section 8 we discuss the reduction from  $L_p$ -fitting tree metrics to  $L_p$ -fitting ultrametrics. In Section 9 we prove APX-Hardness of  $L_1$ -fitting ultrametrics and  $L_1$ -fitting tree metrics. We conclude in Section 10.

## 2 LP definitions for Hierarchical Correlation Clustering

In this section we present the IP/LP formulation of Hierarchical Correlation Clustering, implicit in [3, 39]. In what follows we use  $[n]$  to denote the set  $\{1, \dots, n\}$ .

**Definition 2** (IP/LP formulation of Hierarchical Correlation Clustering). *Given is a set  $S$ ,  $\ell$  positive numbers  $\delta^{(1)}, \dots, \delta^{(\ell)}$  and edge-sets  $E^{(1)}, \dots, E^{(\ell)} \subseteq \binom{S}{2}$ . The objective is:*

$$\min \sum_{t=1}^{\ell} \delta^{(t)} \left( \sum_{\{i,j\} \in E^{(t)}} x_{i,j}^{(t)} + \sum_{\{i,j\} \notin E^{(t)}} (1 - x_{i,j}^{(t)}) \right)$$

subject to the constraints

$$x_{i,j}^{(t)} \leq x_{i,k}^{(t)} + x_{j,k}^{(t)} \quad \forall \{i, j, k\} \in \binom{S}{3}, t \in [\ell] \quad (5)$$

$$x_{i,j}^{(t)} \geq x_{i,j}^{(t+1)} \quad \forall \{i, j\} \in \binom{S}{2}, t \in [\ell - 1] \quad (6)$$

$$x_{i,j}^{(t)} \in \begin{cases} \{0, 1\} & \text{if IP} \\ [0, 1] & \text{if LP} \end{cases} \quad \forall \{i, j\} \in \binom{S}{2}, t \in [\ell] \quad (7)$$

Concerning the IP, the values  $x_{i,j}^{(t)}$  encode the hierarchical partitions, with  $x_{i,j}^{(t)} = 0$  meaning that  $i, j$  are in the same part of the partition at level  $t$ , and  $x_{i,j}^{(t)} = 1$  meaning that they are not. Inequality (5) ensures that the property of being in the same part of a partition is transitive. Inequality (6) ensures that the partitions are hierarchical.

In the LP, where fractional values are allowed,  $x_{i,j}^{(t)}$  is said to be the *LP-distance* between  $i, j$  at level  $t$ . If their LP-distance is small, one should think of it as the LP suggesting that  $i, j$  should be in the same part of the output partition, while a large LP-distance suggests that they should not. Notice that for any given level  $t$ , the LP-distances satisfy the triangle inequality, by (5).

We also note that the Correlation Clustering problem directly corresponds to the case where  $\ell = \delta_1 = 1$ .

## 3 From Hierarchical Correlation Clustering to Hierarchical Cluster Agreement Problem

Our main technical contribution is proving the following theorem.

**Theorem 3.** *The Hierarchical Correlation Clustering problem can be solved in deterministic polynomial time within a constant approximation factor.*

In this section, we present a deterministic reduction from Hierarchical Correlation Clustering to Hierarchical Cluster Agreement that guarantees:

$$\text{HierCorrClust} < (\text{CorrClust} + 1)(\text{HierClustAgree} + 1) \quad (\text{C}) \text{ from Figure 1}$$

In Sections 4 and 5 we present a deterministic polynomial time constant factor approximation algorithm for Hierarchical Cluster Agreement; combined with a known deterministic polynomial time constant factor approximation algorithm for Correlation Clustering [52], it completes the proof of Theorem 3.

Assume that Correlation Clustering can be approximated within a factor  $\alpha$  and that Hierarchical Cluster Agreement can be approximated within a factor  $\beta$  (Section 4). We prove Inequality (C), by providing an algorithm to approximate Hierarchical Correlation Clustering within a factor  $(\alpha + 1)(\beta + 1) - 1$ .

Suppose we have a Hierarchical Correlation Clustering instance  $S, \delta^{(1)}, \dots, \delta^{(\ell)}, E^{(1)}, \dots, E^{(\ell)}$ . Our algorithm first solves the Correlation Clustering instance  $S, E^{(t)}$  to acquire partition  $Q^{(t)}$ , for every level  $t$ . Then, we solve the Hierarchical Cluster Agreement instance  $S, \delta^{(1)}, \dots, \delta^{(\ell)}, \mathcal{E}(Q^{(1)}), \dots, \mathcal{E}(Q^{(\ell)})$  and obtain hierarchical partitions  $P^{(1)}, \dots, P^{(\ell)}$ .

The proof that the hierarchical partitions  $P^{(1)}, \dots, P^{(\ell)}$  are a good approximation to the Hierarchical Correlation Clustering instance follows from two observations. First, by definition, the cost of Hierarchical Correlation Clustering is related to certain symmetric differences. Since the cardinality of symmetric differences satisfy the triangle inequality, we can switch between the cost of Hierarchical Correlation Clustering and Hierarchical Cluster Agreement under the same output, with only an additive term related to  $|E^{(t)} \Delta \mathcal{E}(Q^{(t)})|$  and not related to the output. Second, by definition of  $Q^{(t)}$ , the cardinality of the symmetric difference  $|E^{(t)} \Delta \mathcal{E}(Q^{(t)})|$  is minimized within a factor  $\alpha$ .

More formally, for this proof we need to define the following three concepts:

- For any  $t \in [\ell]$ ,  $OPT_{\text{CorrClust}}^{(t)}$  is an optimal solution to the Correlation Clustering instance at level  $t$ , that is a partition minimizing

$$|E^{(t)} \Delta \mathcal{E}(OPT_{\text{CorrClust}}^{(t)})|$$

- $OPT_{\text{HierCorrClust}} = (OPT_{\text{HierCorrClust}}^{(1)}, \dots, OPT_{\text{HierCorrClust}}^{(\ell)})$  is an optimal solution to the Hierarchical Correlation Clustering instance, that is a sequence of hierarchical partitions minimizing

$$\sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(OPT_{\text{HierCorrClust}}^{(t)})|$$

- $OPT_{\text{HierClustAgree}} = (OPT_{\text{HierClustAgree}}^{(1)}, \dots, OPT_{\text{HierClustAgree}}^{(\ell)})$  is an optimal solution to the Hierarchical Cluster Agreement instance, that is a sequence of hierarchical partitions minimizing

$$\sum_{t=1}^{\ell} \delta^{(t)} |\mathcal{E}(Q^{(t)}) \Delta \mathcal{E}(OPT_{\text{HierClustAgree}}^{(t)})|$$

Notice, for any  $t$ , the difference between  $OPT_{CorrClust}^{(t)}$  and  $OPT_{HierCorrClust}^{(t)}$ . The first one optimizes locally (per level), meaning that  $|E^{(t)} \Delta \mathcal{E}(OPT_{CorrClust}^{(t)})| \leq |E^{(t)} \Delta \mathcal{E}(OPT_{HierCorrClust}^{(t)})|$ , and therefore  $\sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(OPT_{CorrClust}^{(t)})| \leq \sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(OPT_{HierCorrClust}^{(t)})|$ . This does not contradict the definition of  $OPT_{HierCorrClust}$ , as the sequence  $OPT_{CorrClust}^{(1)}, \dots, OPT_{CorrClust}^{(\ell)}$  is not a sequence of hierarchical partitions.

The cost of our solution is

$$\sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(P^{(t)})| \leq \sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(Q^{(t)})| + \sum_{t=1}^{\ell} \delta^{(t)} |\mathcal{E}(Q^{(t)}) \Delta \mathcal{E}(P^{(t)})| \quad (8)$$

By definition of  $P^{(1)}, \dots, P^{(\ell)}$ , they minimize the second term of (8) within a factor  $\beta$ . Therefore, the second term is upper bounded by  $\beta \sum_{t=1}^{\ell} \delta^{(t)} |\mathcal{E}(Q^{(t)}) \Delta \mathcal{E}(OPT_{HierClustAgree}^{(t)})|$ , which, by optimality of  $OPT_{HierClustAgree}$  is upper bounded by  $\beta \sum_{t=1}^{\ell} \delta^{(t)} |\mathcal{E}(Q^{(t)}) \Delta \mathcal{E}(OPT_{HierCorrClust}^{(t)})|$ .

Using the triangle inequality again, we further upper bound the second term by:

$$\beta \sum_{t=1}^{\ell} \delta^{(t)} |\mathcal{E}(Q^{(t)}) \Delta E^{(t)}| + \beta \sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(OPT_{HierCorrClust}^{(t)})|$$

Therefore, we can rewrite (8) as:

$$\sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(P^{(t)})| \leq (\beta + 1) \sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(Q^{(t)})| + \beta \sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(OPT_{HierCorrClust}^{(t)})|$$

Since  $Q^{(t)}$  is obtained by solving Correlation Clustering at level  $t$  within a factor  $\alpha$ , we get

$$\sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(Q^{(t)})| \leq \alpha \sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(OPT_{CorrClust}^{(t)})|$$

By optimality of  $OPT_{CorrClust}^{(t)}$ , for each  $t \in [\ell]$ , we have

$$\sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(OPT_{CorrClust}^{(t)})| \leq \sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(OPT_{HierCorrClust}^{(t)})|$$

which proves that

$$\begin{aligned} \sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(P^{(t)})| &\leq ((\beta + 1)\alpha + \beta) \sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(OPT_{HierCorrClust}^{(t)})| \\ &= ((\alpha + 1)(\beta + 1) - 1) \sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta \mathcal{E}(OPT_{HierCorrClust}^{(t)})| \end{aligned}$$

## 4 Constant approximation Algorithm for Hierarchical Cluster Agreement

In this section we introduce our main algorithm, which consists of three parts: Solving the LP formulation of the problem, the LP-Cleaning subroutine and the Derive-Hierarchy subroutine.

Informally, the LP-Cleaning subroutine uses the fractional solution of the LP relaxation of Hierarchical Cluster Agreement to decide which of our input-sets are important and which are not. The decision is not a binary one, because important sets are also cleaned, in the sense that bad parts of them may be removed. However, at least a 0.9 fraction of them is left intact, while unimportant sets are completely discarded.

The Derive-Hierarchy part then receives the cleaned input-sets by LP-Cleaning, and applies a very simple combinatorial algorithm on them to compute the output.

We notice that the weights  $\delta^{(*)}$  are only used for solving the LP. Moreover, the fractional LP-solution is only used by LP-Cleaning to guide this “nearly-binary” decision for each input-set. The rest of the algorithm is combinatorial and does not take the LP-solution into account.

### 4.1 LP Definitions for Hierarchical Cluster Agreement

The IP-formulation of Hierarchical Cluster Agreement is akin to the IP-formulation of Hierarchical Correlation Clustering. Namely, the constraints are exactly the same for both problems. The only difference is in the objective function where we replace the general edge-sets  $E^{(1)}, \dots, E^{(\ell)}$  with the disjoint clique edges from  $\mathcal{E}(Q^{(1)}), \dots, \mathcal{E}(Q^{(\ell)})$ . Similarly for the LP-relaxation of Hierarchical Cluster Agreement. Here each component in  $Q^{(t)}$  is called a *level- $t$  input cluster*.

To simplify our discussion, we use  $x^{(*)}$  to denote a fractional solution to the LP-relaxation of Hierarchical Cluster Agreement, that is a vector containing all  $x_{i,j}^{(t)}, \{i, j\} \in \binom{S}{2}, t \in [\ell]$ . One can think of  $x^{(*)}$  as the optimal fractional solution, but in principle it can be any solution.

We use  $x^{(t)}$ , for some particular  $t \in [\ell]$ , to denote the vector containing all  $x_{i,j}^{(t)}, \{i, j\} \in \binom{S}{2}$ .

As previously, we use the term LP distances to refer to the entries of  $x^{(*)}$ , and notice that for any particular  $t \in [\ell]$  the LP distances even satisfy the triangle inequality, by the LP constraints.

Given  $x^{(*)}$  we define  $B_{<r}^{(t)}(i)$  to be the ball of species with LP-distance less than  $r$  from  $i$  at level  $t$ . More formally,  $B_{<r}^{(t)}(i) = \{j \in S \mid x_{i,j}^{(t)} < r\}$ . Similarly, for a subset  $S'$  of  $S$  we define the ball  $B_{<r}^{(t)}(S') = \{j \in S \mid \exists i \in S' \text{ s.t. } x_{i,j}^{(t)} < r\}$ .

We also define the LP cost of species  $i, j$  at level  $t$  as

$$\text{cost}_{i,j}^{(t)} = \begin{cases} \delta^{(t)} x_{i,j}^{(t)} & \text{if } \{i, j\} \in \mathcal{E}(Q^{(t)}) \\ \delta^{(t)} (1 - x_{i,j}^{(t)}) & \text{otherwise} \end{cases}$$

as well as the LP cost of species in a set  $S' \subseteq S$  at level  $t$  as

$$\text{cost}_{S'}^{(t)} = \sum_{\substack{\{i,j\} \in \binom{S}{2} \\ i \in S' \text{ or } j \in S'}} \text{cost}_{i,j}^{(t)}$$

and in case  $S'$  only contains a single species  $i$ , we write  $\text{cost}_i^{(t)}$  instead of  $\text{cost}_{\{i\}}^{(t)}$ .

Then the LP cost at level  $t$  is denoted as  $\text{cost}^{(t)} = \text{cost}_S^{(t)}$ .

Finally, the LP cost is simply  $\text{cost}^{(*)} = \sum_{t=1}^{\ell} \text{cost}^{(t)}$ .

## 4.2 Main Algorithm

The pseudocode for our main algorithm for Hierarchical Cluster Agreement is given in Algorithm 1.

---

### Algorithm 1 Hierarchical Cluster Agreement Algorithm

---

**Input** A set  $S$ , a sequence  $Q^{(*)} = (Q^{(1)}, \dots, Q^{(\ell)})$  of partitions of  $S$ , and weights  $\delta^{(*)} = (\delta^{(1)}, \dots, \delta^{(\ell)})$

**Returns** A sequence  $P^{(*)} = (P^{(1)}, \dots, P^{(\ell)})$  of hierarchical partitions of  $S$

- 1:  $x^{(*)} \leftarrow \text{Solve}(\text{LP-relaxation}(S, Q^{(*)}, \delta^{(*)}))$
- 2:  $L^{(*)} \leftarrow \text{LP-Cleaning}(S, Q^{(*)}, x^{(*)})$
- 3: **return**  $\text{Derive-Hierarchy}(S, L^{(*)})$

---

Our LP relaxation has size polynomial in  $S, \ell$ , and the two subroutines also run in polynomial time, as we show later. Therefore the whole algorithm runs in polynomial time.

## 4.3 LP cleaning Algorithm

In Algorithm 2 we provide the pseudocode of the LP Cleaning step of our algorithm.

Intuitively, the aim of this algorithm is to clean the input sets so that (ideally) all species remaining in a set have small LP distances to each other, and large LP distances to species not in the set.

---

### Algorithm 2 LP-Cleaning

---

**Input** A set  $S$ , a sequence  $Q^{(*)} = (Q^{(1)}, \dots, Q^{(\ell)})$  of partitions of  $S$ , and a fractional solution  $x^{(*)}$

**Returns** A sequence  $L^{(*)} = (L^{(1)}, \dots, L^{(\ell)})$  of families of disjoint subsets of  $S$

- 1: **for**  $t \leftarrow 1, \dots, \ell$  **do**
- 2:      $L^{(t)} \leftarrow \emptyset$
- 3:     **for**  $C_I \in Q^{(t)}$  **do**
- 4:          $C_{LP} \leftarrow \left\{ i \in C_I \mid \begin{array}{l} |B_{<0.1}^{(t)}(i) \cap C_I| > \frac{1}{2}|C_I|, \\ |B_{<0.6}^{(t)}(i) \setminus C_I| \leq 0.05|C_I| \end{array} \right\}$
- 5:         **if**  $|C_{LP}| \geq 0.9|C_I|$  **then**
- 6:              $L^{(t)} \leftarrow L^{(t)} \cup \{C_{LP}\}$
- 7: **return**  $L^{(*)} = (L^{(1)}, \dots, L^{(\ell)})$

---

Formally Algorithm 2 takes a sequence  $Q^{(*)} = (Q^{(1)}, \dots, Q^{(\ell)})$  of partitions of  $S$  and a fractional solution  $x^{(*)}$  containing LP distances. It outputs a sequence of families of disjoint subsets of  $S$ ,  $L^{(*)} = (L^{(1)}, \dots, L^{(\ell)})$ . Here each component of  $L^{(t)}$  is called a *level- $t$  LP-cluster*.

In the algorithm, for each input partition  $Q^{(t)}$  we process every level- $t$  input-cluster  $C_I \in Q^{(t)}$  separately. For this we remove all the species in  $C_I$  that do not have very small LP distance to at least half the species in  $C_I$  or that have small LP distance to many species not in  $C_I$ . More formally, we remove all the species in  $C_I$  with LP distance less than 0.1 to at most half the species in  $C_I$  or with LP distance less than 0.6 to more than  $0.05|C_I|$  species not in  $C_I$ .

After the cleaning step we discard  $C_I$  if less than 9/10 fraction of the species survive. Otherwise we create an LP-cluster  $C_{LP}$  containing the species in  $C_I$  that survive. Next we add the level- $t$  LP-cluster  $C_{LP}$  to  $L^{(t)}$ .

Out of several properties that we prove concerning the output of the LP-Cleaning, we briefly mention the following one: The output sequence  $L^{(*)}$  is *hierarchy-friendly* in the sense that no two LP-clusters at the same level  $t$  can be intersected by the same LP-cluster at level  $t' < t$ . We formally prove this in Lemma 7.

The LP-Cleaning subroutine trivially runs in time polynomial in  $S, \ell$ .

#### 4.4 Derive-hierarchy Algorithm

In this section, we introduce Derive-Hierarchy (Algorithm 3). It takes as input a hierarchy-friendly sequence  $L^{(*)} = (L^{(1)}, \dots, L^{(\ell)})$  of families of disjoint subsets of  $S$  and outputs a sequence  $P^{(*)} = (P^{(1)}, \dots, P^{(\ell)})$  of hierarchical partitions of  $S$ . The execution of the algorithm can be seen, via a graphical example, in Figure 2.

---

##### Algorithm 3 Derive-Hierarchy

---

**Input** A set  $S$ , and a hierarchy-friendly sequence  $L^{(*)} = (L^{(1)}, \dots, L^{(\ell)})$  of families of disjoint subsets of  $S$

**Returns** A sequence  $P^{(*)} = (P^{(1)}, \dots, P^{(\ell)})$  of hierarchical partitions of  $S$

- 1: Construct an empty forest  $\mathcal{F}$
- 2: **for**  $i \in S$  **do**
- 3:     Create a singleton tree  $T$  with a node  $u_i$  and add it to  $\mathcal{F}$
- 4:     Set  $C(u_i) \leftarrow C^+(u_i) \leftarrow \{i\}$
- 5: **for**  $t \leftarrow 1, \dots, \ell$  **do**
- 6:     **for**  $C_{LP} \in L^{(t)}$  **do**
- 7:         Create a node  $u$  and set  $C(u) \leftarrow C_{LP}$
- 8:         **for** all roots  $v \in \mathcal{F}$  s.t.  $C(v) \cap C(u) = \emptyset$  **do**
- 9:              $C(u) \leftarrow C(u) \setminus C^+(v)$
- 10:          $C^+(u) \leftarrow C(u)$
- 11:         **for** all roots  $v \in \mathcal{F}$  s.t.  $C(v) \cap C(u) \neq \emptyset$  **do**
- 12:              $C^+(u) \leftarrow C^+(u) \cup C^+(v)$
- 13:         Make  $v$  a child of  $u$  in  $\mathcal{F}$
- 14:     Set  $P^{(t)}$  to contain the extended-clusters  $C^+(v)$  of all roots  $v \in \mathcal{F}$
- 15: **return**  $P^{(*)} = (P^{(1)}, \dots, P^{(\ell)})$

---

The algorithm works bottom-up while performing  $\ell$  iterations for  $t = 1, \dots, \ell$ . In the process it incrementally builds a forest  $\mathcal{F}$ . Throughout the algorithm each non leaf node  $u$  in  $\mathcal{F}$  can be identified by an LP-cluster in  $L^{(*)}$ . Moreover for each node  $u$  the algorithm maintains two sets  $C(u)$  and  $C^+(u) \subseteq S$ .

The algorithm starts by initializing  $\mathcal{F}$  with  $|S|$  trees where each tree contains a single node  $u_i$  identified by a species  $i \in S$ . Also it initializes both sets  $C(u_i), C^+(u_i)$  with  $\{i\}$ . Next in iteration  $t$  the algorithm processes the LP-clusters in  $L^{(t)}$  and at the end of the iteration, the  $C^+(\cdot)$  sets associated with the root nodes in  $\mathcal{F}$  define the partition  $P^{(t)}$ . Precisely here, the  $C^+(\cdot)$  set of a root node contains all the species descending from the respective root.

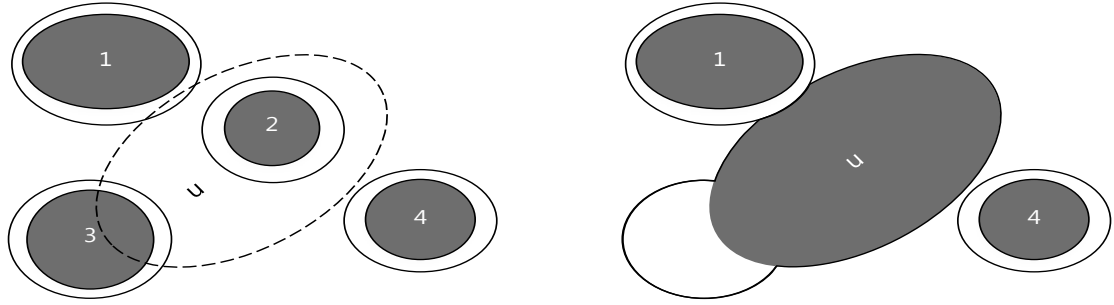


Figure 2: Example of Derive-Hierarchy (Algorithm 3). Nodes 1, 2, 3, 4 (left) are the roots of the forest before inserting the new LP-cluster  $L(u)$  (dashed line). Each node is described by its extended-cluster, with the shaded part being the core-cluster. Core-cluster of nodes 2 and 3 intersect  $L(u)$ ; thus they become children of  $u$  and the extended-cluster of  $u$  covers the extended-clusters of 2, 3 (right). Notice that the core-cluster of  $u$  is reduced due to node 1.

In the  $t$ -th iteration, for each cluster  $C_{LP} \in L^{(t)}$  the algorithm adds a root node  $u$  to forest  $\mathcal{F}$  while initializing the set  $C(u)$  with  $C_{LP}$ . Next for the root node  $u$  the algorithm decides on its children by processing the pre-existing roots in the following way. For consistency first it detects all the pre-existing root nodes  $v$  such that  $C(u)$  does not intersect  $C(v)$ . Then it removes from  $C(u)$  all the species that are descending from  $v$ ; i.e. sets  $C(u) \leftarrow C(u) \setminus C^+(v)$ . Lastly it sets  $u$  as a parent of all other pre-existing root nodes  $v$  such that  $C(u)$  intersects  $C(v)$ . Also accordingly it modifies the set of leaf nodes of the subtree rooted at  $u$  by setting  $C^+(u) \leftarrow C^+(u) \cup C^+(v)$ . Notice here that some of the root-nodes may correspond to sets from levels lower than  $t$ , in case no parent was assigned to them.

At the end of iteration  $t$  the algorithm completes processing all the LP-clusters in  $L^{(1)}, \dots, L^{(t)}$  and constructs partitions  $P^{(1)}, \dots, P^{(t)}$ . At the end of the  $\ell$  iterations it outputs the  $\ell$  partitions  $P^{(*)} = (P^{(1)}, \dots, P^{(\ell)})$ .

The Derive-Hierarchy subroutine trivially runs in time polynomial in  $S, \ell$ .

## 5 Analysis of Hierarchical Cluster Agreement Algorithm

In this section, we proceed with our analysis. We first lay out some terminology, then provide some results related to the LP Cleaning, then some structural results, and finally prove that our algorithm is a constant factor approximation for Hierarchical Cluster Agreement.

### 5.1 Terminology

Notice that throughout the execution of the algorithm,  $\mathcal{F}$  is an incrementally updated graph (that is, no deletions occur). In fact it is always a forest, as we start with  $|S|$  isolated nodes and only introduce new nodes as parents of roots of some of the existing trees. Moreover, this process implies that the subtree rooted at any specific node is never modified.

From now on we use  $\mathcal{F}$  to refer to the final instance of the incrementally updated forest. We use  $\mathcal{F}(u)$  to refer to the state of this incrementally updated forest after introducing  $u$ ; therefore

$\mathcal{F}(u) \setminus \{u\}$  denotes the state of the forest exactly before introducing node  $u$ . We naturally identify the leaves of  $\mathcal{F}$  with the species of  $S$ .

For any node  $u$  in the forest  $\mathcal{F}$ , the Derive-Hierarchy algorithm defines  $C(u)$ , which we call the core-cluster of  $u$ , and  $C^+(u)$ , which we call the extended-cluster of  $u$ . Furthermore, notice that each core-cluster  $C(u)$  is a subset of some LP-cluster  $C_{LP}$  (Line 7 of Algorithm 3); we call this the LP-cluster of  $u$  and denote it by  $L(u)$ . Moreover, each LP-cluster  $L(u)$  is a subset of an input-cluster  $C_I$  (Line 3 of Algorithm 2); we call this the input-cluster of  $u$  and denote it by  $I(u)$ . These concepts are well defined for any new node  $u$  and never change throughout the algorithm. We remind the reader that LP-Cleaning discards some of the input clusters, in the sense that they have no corresponding LP-cluster, and therefore they do not match  $I(u)$ , for any node  $u$ .

Directly from the algorithm we get that

$$\begin{aligned} C(u) &\subseteq L(u) \subseteq I(u) \\ C(u) &\subseteq C^+(u) \end{aligned}$$

To help with our discussion, we also define the following variables related to the Derive-Hierarchy algorithm (Algorithm 3):

$$\begin{aligned} \Delta^-(u) &= L(u) \setminus C(u) \\ \Delta^+(u) &= C^+(u) \setminus C(u) \end{aligned}$$

For a node  $u \in \mathcal{F}$ , we define its level  $t(u)$  to be the value of iteration  $t$  in Algorithm 3 when internal node  $u$  was introduced, and 0 when  $u$  is a leaf node.

## 5.2 LP-Cleaning Results (Algorithm 2)

We start with some observations that are heavily used in proving structural results regarding the core and the extended-clusters. These are in turn used for lower-bounding the LP cost.

The most important reason we are using the LP-Cleaning subroutine is so that any two species belonging in the same LP-Cluster at level  $t$  have small LP-distance.

**Lemma 4.** *Given a node  $u \in \mathcal{F}$  and a species  $i$  in  $u$ 's LP-cluster  $L(u)$ , it holds that the LP-distance from  $i$  to any other species in  $L(u)$  is less than 0.2 for all levels  $t \geq t(u)$ , that is  $B_{<0.2}^{(t)}(i) \supseteq L(u)$ .*

*Proof.* It suffices to prove that  $x_{i,j}^{(t)} < 0.2$  for all  $j \in L(u)$  only for level  $t = t(u)$ , as the LP constraints enforce  $x_{i,j}^{(t+1)} \leq x_{i,j}^{(t)}$ .

By pigeonhole principle, since both  $B_{<0.1}^{(t)}(i) \cap I(u)$  and  $B_{<0.1}^{(t)}(j) \cap I(u)$  have size more than  $|I(u)|/2$  (Line 4 of Algorithm 2), there exists a node  $k \in I(u)$  for which both  $x_{i,k}^{(t)}$  and  $x_{j,k}^{(t)}$  are less than 0.1. Since the LP-distances in  $x^{(t)}$  satisfy the triangle inequality, it follows that  $x_{i,j}^{(t)} < 0.2$  (enforced by the LP constraints).  $\square$

For the analysis, it is convenient that our relations involve the LP-clusters instead of the input-clusters. Therefore, we rephrase Line 4 of Algorithm 2 in terms of LP-clusters, effectively proving that few species outside of an LP-cluster  $L(u)$  have small LP-distances to  $L(u)$ .

**Lemma 5.** *For any node  $u \in \mathcal{F}$  it holds that  $|B_{<0.4}^{(t(u))}(L(u))| \leq (1 + \frac{1}{6})|L(u)|$ . In particular,  $|B_{<0.4}^{(t(u))}(L(u)) \setminus L(u)| \leq \frac{1}{6}|L(u)|$ .*



*Proof.* Let  $t = t(u)$ . We claim that species close to some species in  $L(u)$  are close to all species in  $L(u)$ . Formally, we claim that for any  $i \in L(u)$

$$B_{<0.4}^{(t)}(L(u)) \subseteq B_{<0.6}^{(t)}(i)$$

Let  $j \in B_{<0.4}^{(t)}(L(u))$ . We bound the LP-distance between  $i, j$  by finding an intermediate  $i'$  that is close to both and applying the triangle inequality forced by the LP constraints. By definition of  $j$ , there exists a species  $i' \in L(u)$  with LP-distance less than 0.4 from  $j$ . By Lemma 4, the LP-distance between  $i$  and  $i'$  is less than 0.2, and thus by triangle inequality  $x_{i,j}^{(t)} < 0.6$ .

Line 4 of Algorithm 2 gives that

$$|B_{<0.6}^{(t)}(i) \setminus I(u)| \leq 0.05|I(u)| \implies |B_{<0.6}^{(t)}(i)| \leq 0.05|I(u)| + |I(u)|$$

Combining these two relations, and by  $|L(u)| \geq 0.9|I(u)|$  (Line 5 of Algorithm 2):

$$|B_{<0.4}^{(t)}(L(u))| \leq |B_{<0.6}^{(t)}(i)| \leq \frac{(1 + 0.05)}{0.9}|L(u)| = (1 + \frac{1}{6})|L(u)|$$

□

The following lemma is just a convenient application of the triangle inequality of our LP, that is heavily used in subsequent proofs. Informally, it states that, under certain mild conditions, the LP-distance is small not only if  $i, j$  belong in the same LP-cluster (or core-cluster), but even if they happen to be in different clusters that are both intersected by the same third cluster.

**Lemma 6.** *Let  $u, v, w \in \mathcal{F}$  be three arbitrary nodes. Assume that the LP-cluster of  $v$  intersects the LP-clusters of  $u$  and  $w$  and  $t_{max} = \max\{t(u), t(v), t(w)\}$ . Then for any  $i, j \in \{L(u) \cup L(v) \cup L(w)\}$  their LP-distance at level  $t_{max}$  is less than 0.6, and  $B_{<0.4}^{(t_{max})}(L(u)) \supseteq L(u) \cup L(v) \cup L(w)$ .*

*Proof.* If both  $i, j$  are either in  $L(u)$  or  $L(v)$  or  $L(w)$  then the claim follows trivially from Lemma 4. Otherwise we use triangle inequality twice, with species in the intersections of the clusters as intermediates. More formally, let  $k \in L(u) \cap L(v)$ ,  $k' \in L(v) \cap L(w)$ . Lemma 4 implies three things:

- (1)  $x_{i,k}^{(t_{max})} < 0.2$ , for any  $i \in L(u) \cup L(v)$
- (2)  $x_{k,k'}^{(t_{max})} < 0.2$ , as both  $k, k' \in L(v)$
- (2)  $x_{k',j}^{(t_{max})} < 0.2$ , for any node  $j \in L(v) \cup L(w)$

Since the LP-distances  $x^{(t_{max})}$  respect the triangle inequality it holds that  $x_{i,j}^{(t_{max})} < 0.6$ . The claim about the ball of  $L(u)$  follows by taking the distance from  $k$  to  $j$ . □

We are now ready to prove the hierarchy-friendly property of the output of LP-Cleaning, as we informally claimed when introducing the algorithm. We claim that two LP-clusters of the same level cannot be intersected by the same lower level LP-cluster.

**Lemma 7.** *Given two nodes  $v, w \in \mathcal{F}$  on the same level, there is no lower level node  $u$  such that  $L(u)$  intersects both  $L(v)$  and  $L(w)$ .*

*In particular, there is also no  $C(u)$  intersecting both  $L(v)$  and  $L(w)$ .*

*Proof.* The intuition is that  $L(v), L(w)$  are close and thus Algorithm 2 would discard at least one of them.

Without loss of generality, let  $|L(v)| \geq |L(w)|$ .  $L(v), L(w)$  are disjoint as they are subsets of different parts of the partition  $Q^{(t(v))}$ , by Algorithm 2.

By Lemma 6  $|B_{<0.4}^{(t(w))}(L(w))| \geq |L(v)| + |L(w)| \geq 2|L(w)|$ , which contradicts Lemma 5.  $\square$

We finally present a simple lower bound on the LP cost.

**Lemma 8.** *Let  $C_I \in Q^{(t)}$  be an input-cluster at level  $t$ , and  $C_{LP}$  be the respective LP-cluster from Algorithm 2. Fix a species  $i \in C_I \setminus C_{LP}$ . Then the fractional LP cost  $cost_i^{(t)} = \Omega(\delta^{(t)}|C_I|)$ .*

*Proof.* There are two reasons for  $i$  to be in  $C_I \setminus C_{LP}$ , by Line 4 of Algorithm 2. Either half the species in  $C_I$  are at distance at least 0.1 from  $i$ , or more than  $0.05|C_I|$  species not in  $C_I$  are at distance at most 0.6 from  $i$ .

In the first case  $cost_i^{(t)} \geq 0.1 \cdot (\frac{1}{2}|C_I|)\delta^{(t)}$ , and in the second case  $cost_i^{(t)} \geq (1 - 0.6) \cdot 0.05|C_I|\delta^{(t)}$ .  $\square$

### 5.3 Derive-Hierarchy results (Algorithm 3)

In this section we present several structural results related to our algorithm.

We start with pointing out that our algorithm ends up with the same output, no matter the order in which we process LP-clusters of the same level. This is due to the input sequence  $L^{(*)}$  being hierarchy-friendly.

**Remark 9.** *The output of Algorithm 3 is the same, irrespective of the order in which LP-clusters of the same level are processed in Line 6.*

*Proof.* For each level, fix any ordering in which LP-clusters of the same level are processed, and run the algorithm. For any  $t \in [\ell]$ , let  $\mathcal{F}_{t-1}$  be the state of the forest just before processing the first node of level  $t$ . We show that for any level- $t$  LP-cluster  $C_{LP}$  with corresponding node  $u$  (that is  $t(u) = t$  and  $L(u) = C_{LP}$ ), no matter when it was actually processed due to the ordering we fixed, the effect is the same as if it was the first level- $t$  LP-cluster processed. More formally, let  $N(u)$  be the set of children of  $u$ , and  $C_{t-1}(u), C_{t-1}^+(u), N_{t-1}(u)$  be the core-cluster, the extended-cluster and the set of children of  $u$  in the case where  $C_{LP}$  was the first LP-cluster of level- $t$  to be processed. Then  $C(u) = C_{t-1}(u), C^+(u) = C_{t-1}^+(u), N(u) = N_{t-1}(u)$ .

The main idea is that if a root  $v \in \mathcal{F}_{t-1}$  has a core-cluster intersecting  $L(u)$ , then it is still a root just before inserting  $u$ ; else  $v$  would have another parent  $w$  of level  $t$ , meaning  $C(v) \subseteq L(v)$  would also intersect  $C(w) \subseteq L(w)$  (Line 11), which contradicts that  $L^{(*)}$  is hierarchy-friendly.

For  $u$ 's children, we first show that  $N(u) \subseteq N_{t-1}(u)$ . Suppose this was not true, then there would exist a level- $t$  node  $v \in N(u) \setminus N_{t-1}(u)$ . That would imply that  $u$ 's and  $v$ 's core clusters intersect (Line 11). But, core-clusters are always subsets of their corresponding LP-clusters, and LP-clusters of the same level are disjoint.

Before proving  $N_{t-1}(u) \subseteq N(u)$ , we need to show that  $C(u) = C_{t-1}(u)$ . We show it by proving that  $L(u) \setminus C(u) = L(u) \setminus C_{t-1}(u)$ . If a species  $i$  is in  $L(u) \setminus C(u)$ , then it is in the extended-cluster of some node  $v$  processed before  $u$  such that their core-clusters do not intersect (Line 8). If  $t(v) = t$ , then  $i$  is either in  $C(v)$  (contradiction as it would then not be in  $L(u)$ ), or in the extended-cluster of one of its children  $w$ , which we proved are of lower-level. Thus  $w$  was a root in  $\mathcal{F}_{t-1}$ . Again by  $L^{(*)}$  being hierarchy-friendly,  $C(w) \subseteq L(w)$  does not intersect  $L(u)$ , meaning it does not intersect

$C_{t-1}(u) \subseteq L(u)$  and so  $i$  would also be in  $L(u) \setminus C_{t-1}(u)$  (Line 8). If  $t(v) < t$ , then  $v$  itself was a root in  $\mathcal{F}_{t-1}$ . The same argument in reverse order is used to prove that if  $i$  is in  $L(u) \setminus C_{t-1}(u)$  then it is in  $L(u) \setminus C(u)$ .

We now see that  $N_{t-1}(u) \subseteq N(u)$ ; that is because if  $v \in N_{t-1}(u)$ , then  $v$  is a root in  $\mathcal{F}_{t-1}$  with a core-cluster intersecting  $L(u)$ , and by  $L^{(*)}$  being hierarchy-friendly it is also a root in  $\mathcal{F}(u) \setminus \{u\}$ . As  $C(v)$  intersects  $C_{t-1}(u) = C(u)$ , we get  $v \in N(u)$  (Line 11).

Finally, a species  $i$  in  $C_{t-1}^+(u) \setminus C_{t-1}(u)$  is part of the extended cluster of a node in  $N_{t-1}(u)$ ; this child is still a root in  $\mathcal{F}(u) \setminus \{u\}$  by  $L^{(*)}$  being hierarchy-friendly, therefore  $i \in C^+(u) \setminus C(u)$ . The other way around, a species  $i$  in  $C^+(u) \setminus C(u)$  is part of the extended cluster of a node in  $N(u)$ ; this child is a root in  $\mathcal{F}_{t-1}$  as  $u$  has no children of level  $t$ , therefore  $i \in C^+(u) \setminus C_{t-1}(u)$ .  $\square$

Next, we prove two claims that we have already mentioned informally while introducing Algorithm 3 (Derive-Hierarchy). First we claim that the incrementally built graph is always a forest and also for any node  $u$ , its extended-cluster contains exactly the species descending from  $u$ . We notice that the previously stated results do not require these properties.

**Lemma 10.** *For any  $u \in \mathcal{F}$ ,  $\mathcal{F}(u)$  is a forest of rooted trees with  $|S|$  leaves identified with the species of  $S$ , and for each  $u \in \mathcal{F}$ ,  $C^+(u)$  is the set of  $u$ 's descending species.*

*Proof.* We prove this inductively based on the order in which the nodes are added in  $\mathcal{F}$ . The base case for both the claims follows by the initialization of the forest with  $|S|$  leaves identified with the species of  $S$ .

When we insert a node, it becomes the parent of some of the existing roots, therefore the forest structure is preserved.

Next let  $u$  be some node in  $\mathcal{F}$  and let  $v_1, \dots, v_k$  be the children of  $u$ . Then by construction all these children nodes are added to  $\mathcal{F}$  before  $u$ , and thus by induction argument for each  $v_m$  the set of descending species of  $v_m$  is exactly the set  $C^+(v_m)$ . Now we need to prove the same for node  $u$ . Note that the set of descending species of  $u$  is precisely the set  $\bigcup_{m \in [k]} C^+(v_m)$ . Moreover by construction  $C^+(u) = C(u) \cup (\bigcup_{m \in [k]} C^+(v_m))$ . Hence to prove the claim we need to show that  $C(u) \subseteq \bigcup_{m \in [k]} C^+(v_m)$ . For the sake of contradiction let  $w \in C(u)$  be a species such that  $w \notin \bigcup_{m \in [k]} C^+(v_m)$ . As  $\mathcal{F}(u) \setminus \{u\}$  is a forest, there exists a unique node  $r$  which is the root node of the tree of  $\mathcal{F}(u) \setminus \{u\}$  that contains  $w$ . Hence again by induction argument  $w \in C^+(r)$ . By our assumption, as  $r$  is not a child of  $u$ ,  $C(u) \cap C(r) = \emptyset$ . Thus Algorithm 3 (Line 8) sets  $C(u) \leftarrow C(u) \setminus C^+(r)$  and hence  $w \notin C(u)$ , which is a contradiction.  $\square$

This simple lemma alone is enough to prove the following corollaries:

**Corollary 11.** *For any  $u \in \mathcal{F}$ , the extended-clusters of the root nodes in  $\mathcal{F}(u)$  form a partition of  $S$ .*

*Proof.* As  $\mathcal{F}(u)$  is a forest, each species is a descendant of exactly one such root and thus belongs in exactly one such extended cluster.  $\square$

**Corollary 12.** *The output of our algorithm is a sequence of hierarchical partitions of  $S$ .*

*Proof.* By Corollary 11 the output of the algorithm is a sequence of partitions of  $S$ . To see that the output partitions are hierarchical, notice that if two species are in the same rooted tree at some point in the algorithm, then they are never separated as we only add nodes in the forest.  $\square$

**Corollary 13.** *For any node  $u \in \mathcal{F}$ , the species removed from its LP-cluster and the species inserted in its core cluster are disjoint,  $\Delta^-(u) \cap \Delta^+(u) = \emptyset$ .*

*Proof.* For the sake of contradiction let  $i \in \Delta^-(u) \cap \Delta^+(u)$ . Since the extended clusters of root nodes in  $\mathcal{F}(u) \setminus \{u\}$  form a partition, let  $v$  be the unique such root for which  $i \in C^+(v)$ . Now as  $i \in \Delta^-(u)$ ,  $C(v) \cap C(u) = \emptyset$  (Line 8 of Algorithm 3). But again  $i \in \Delta^+(u)$  implies  $C(v) \cap C(u) \neq \emptyset$  (Line 11 of Algorithm 3) and both these can never be satisfied together.  $\square$

**Corollary 14.** *If two nodes  $u, v \in \mathcal{F}$  do not have an ancestry-relationship, then their extended clusters do not intersect.*

*Proof.* If their extended clusters intersected, then they would have a descending species in common, which implies an ancestry-relationship.  $\square$

We also need the following result.

**Lemma 15.** *For any node  $u \in \mathcal{F}$ , its extended-cluster is equal to the union of the core clusters of all descendant nodes  $v$  of  $u$ .*

*Proof.* We prove this inductively. As a base-case, the claim trivially holds for the  $|S|$  initial leaves. For an internal node  $u$ , let  $v_1, \dots, v_k$  be the children of  $u$  and let  $D(u)$  be the descendant nodes of  $u$ . Then  $D(u) = \cup_{m \in [k]} D(v_m)$ . Also, by induction, for each  $v_m$ ,  $C^+(v_m) = \cup_{w \in D(v_m)} C(w)$ . Now as  $C^+(u) = C(u) \cup (\cup_{m \in [k]} C^+(v_m))$  we have  $C^+(u) = C(u) \cup (\cup_{w \in D(u)} C(w))$ . which proves our claim.  $\square$

## 5.4 Managing removals and extensions

Using the developed toolkit of structural results, we are ready to show that for any node  $u \in \mathcal{F}$ , all three of the LP-cluster  $L(u)$ , the core-cluster  $C(u)$  and the extended-cluster  $C^+(u)$  are similar; more than that, we show lower bounds of the LP cost related to  $\Delta^-(u) = L(u) \setminus C(u)$  and  $\Delta^+(u) = C^+(u) \setminus C(u)$ .

In particular, we claim that the following inequality holds for every  $u \in \mathcal{F}$ .

$$|\Delta^+(u)| \leq 0.3|C(u)| \tag{9}$$

We prove this claim inductively, based on the order in which nodes are added in  $\mathcal{F}$ . As a base case, we initially create a node  $u_i$  for each species  $i \in S$  with  $C(u_i) = C^+(u_i) = \{i\}$ , meaning that  $\Delta^+(u_i) = \emptyset$ .

For any other node  $u$ , we argue about the size of its extended-cluster  $C^+(u)$  in relation with the core-clusters of its descendants, as suggested by Lemma 15. We now partition the descendants of  $u$  in three parts and argue about each one of them.

Informally, for a node  $u$  we define its top-non-intersecting descendants  $J$  as the set of highest level descendant nodes in  $\mathcal{F}$  whose core-clusters do not intersect  $C(u)$  (the reader is encouraged to consult Figure 3 before proceeding). More formally, using  $v \prec_{\mathcal{F}} u$  to denote that  $v$  is a descendant of  $u$  in forest  $\mathcal{F}$ , we have:

$$J = \left\{ v \in \mathcal{F} \left| \begin{array}{l} v \prec_{\mathcal{F}} u \\ C(u) \cap C(v) = \emptyset \\ C(u) \cap C(w) \neq \emptyset, \forall w \text{ s.t. } v \prec_{\mathcal{F}} w \prec_{\mathcal{F}} u \end{array} \right. \right\}$$

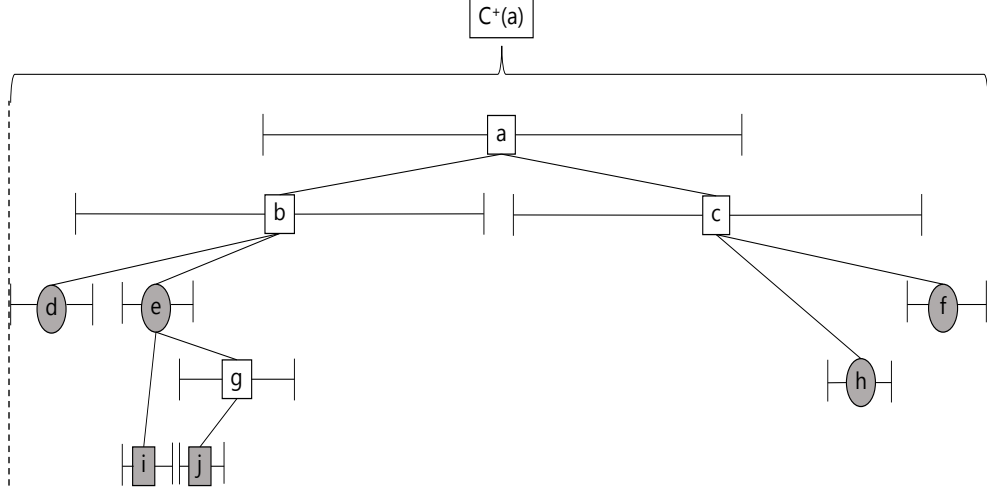


Figure 3: Part of the forest  $\mathcal{F}(a)$ . Intervals around nodes denote core-clusters (two core-clusters intersect if a vertical line intersects both); colored nodes  $\{d, e, f, h, i, j\}$  have core-clusters not intersecting the core-cluster of  $a$ . In particular, the circle-shaped colored nodes are  $a$ 's top-non-intersecting descendants (denoted by  $J = \{d, e, f, h\}$ ). Their proper descendants define  $J^+ = \{g, i, j\}$ .  $R = \{b, c\}$  contains all other proper descendants of  $a$ .

Notice that, by definition, if two nodes  $v, w$  belong in  $u$ 's top-non-intersecting descendants, then none is an ancestor of the other. Therefore  $u$ 's top-non-intersecting descendants  $J$  naturally partitions the proper descendants of  $u$  in three parts:  $J$  itself, the set  $J^+$  of proper descendants of nodes in  $J$ , and  $R$  containing the rest of the proper descendants of  $u$  (i.e., the proper descendants of  $u$  that are not descendants of any node in  $J$ ). We also define sets of species related to these sets:

$$\begin{aligned}
 S_J &= \bigcup_{v \in J} C(v) & (10) \\
 S_{J^+} &= \bigcup_{v \in J^+} C(v) \setminus (C(u) \cup S_J) \\
 S_R &= \bigcup_{v \in R} C(v) \setminus (C(u) \cup S_J \cup S_{J^+})
 \end{aligned}$$

The apparent asymmetry of not excluding  $C(u)$  from  $S_J$  follows from the definition of  $u$ 's top-non-intersecting descendants  $J$ ; the core-clusters of nodes in  $J$  are disjoint from  $C(u)$ , meaning  $S_J$  would be the same even if we excluded species in  $C(u)$ . Note that this is not the case for the core-clusters in  $J^+$  as proper descendants of nodes in  $J$  might still intersect  $C(u)$ , as in Figure 3.

Notice that by Lemma 15 we have  $C^+(u) = C(u) \cup (S_J \cup S_{J^+} \cup S_R)$ , thus

$$\Delta^+(u) = S_J \cup S_{J^+} \cup S_R \quad (11)$$

If  $v \in J \cup J^+ \cup R$ , and its core-cluster does not intersect the core-cluster of  $u$ , then by definition of  $J$  we have that  $v$  is in descendants (not necessarily proper) of  $J$ . Therefore  $v \in J \cup J^+$ , meaning that nodes in  $R$  have core-clusters that intersect  $C(u)$ . Furthermore, by definition, each node  $v$  in

$u$ 's top-non-intersecting descendants has a parent whose core-cluster intersects  $C(u)$ . Therefore, for any species  $i \in C(u)$  and any species  $j \in S_J \cup S_R$ , by Lemma 6 we have that their LP-distance is small, that is

$$x_{i,j}^{(t(u))} < 0.6, \forall i \in C(u), j \in S_J \cup S_R \quad (12)$$

Species in  $S_J \cup S_R$  are not in  $C(u)$ , and so by Corollary 13 they are not in  $L(u)$  as they belong in  $\Delta^+(u)$ . Then Lemma 5 gives

$$|S_J \cup S_R| < \frac{1}{6}|L(u)| \quad (13)$$

We are left to argue about species in  $J^+$ , that is in core-clusters of the descendants of  $u$ 's top-non-intersecting descendants. By Lemma 15, these species all belong in the extended-clusters of  $u$ 's top-non-intersecting descendants,  $\bigcup_{v \in J} C^+(v) \supseteq S_J \cup S_{J^+}$ . By Corollary 14 these extended-clusters are disjoint, thus  $S_{J^+} \subseteq \bigcup_{v \in J} \Delta^+(v)$ . By the inductive hypothesis (9) we get

$$|S_{J^+}| \leq 0.3|S_J| \quad (14)$$

Therefore, by Inequality (13) we get that

$$|S_{J^+}| < \frac{1}{6} \cdot 0.3|L(u)| = \frac{1}{6} \cdot 0.3|C(u) \cup \Delta^-(u)| \quad (15)$$

By (13) and (15) we bound the size of  $\Delta^+(u)$ :

$$|\Delta^+(u)| < 1.3 \cdot \frac{1}{6}|L(u)| \quad (16)$$

We are only left with bounding  $|L(u)|$ . For this we prove that

$$|\Delta^-(u)| < 0.1|C(u)| \quad (17)$$

which, combined with (16), proves our initial claim.

Before proving (17) we make some definitions (see Figure 4). Roughly speaking, we want to identify an appropriate set  $K$  of nodes such that the union of their extended-clusters both contains  $\Delta^-(u)$  and its cardinality is reasonably boundable. In fact, the nodes in  $K$  are descendants of roots of  $\mathcal{F}(u) \setminus \{u\}$  that satisfy the condition in Line 8 of Algorithm 3 (i.e., nodes  $v$  such that  $C(v) \cap C(u) = \emptyset$ , and  $C^+(v) \cap L(u) \neq \emptyset$ ).

We now give a formal constructive definition of the set  $K$ . Let  $M$  be the set containing all the non-descendants of  $u$  at level at most  $t(u)$  whose core-clusters intersect  $L(u)$ . We define  $K'$  to be the set of parents of the nodes in  $M$ . Finally,  $K$  is obtained from  $K'$  by removing the nodes who have a proper ancestor in  $K'$ . Notice by Corollary 14, their extended-clusters are disjoint. We also define sets of species associated with  $K$  as follows.

$$\begin{aligned} S_K &= \bigcup_{v \in K} C(v) \\ S_{K^+} &= \bigcup_{v \in K} C^+(v) \end{aligned} \quad (18)$$

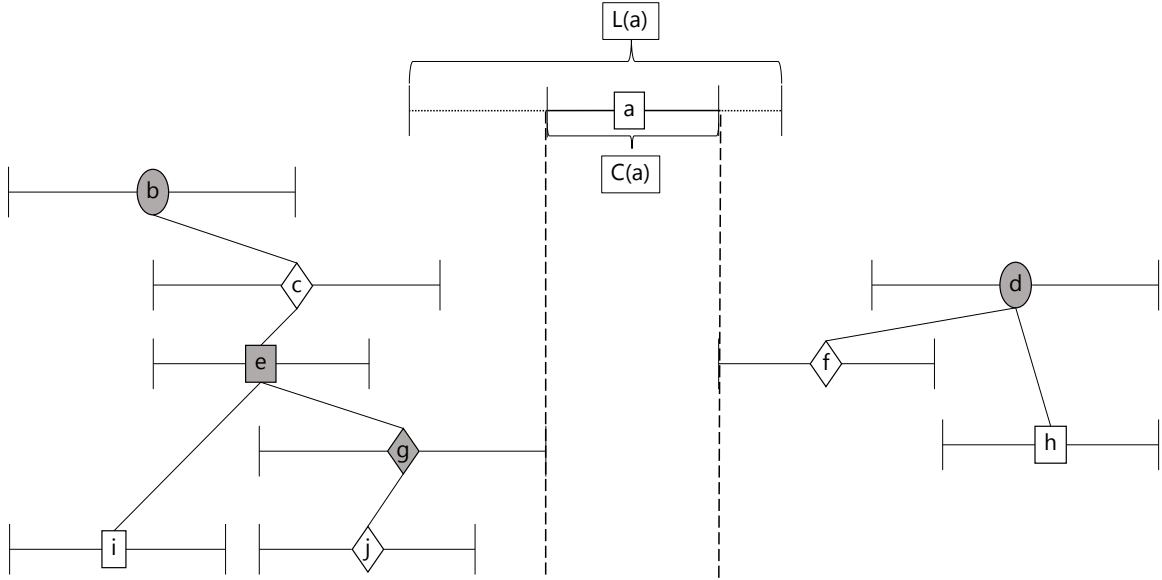


Figure 4: Part of the forest  $\mathcal{F}(a)$ . Intervals around nodes denote core-clusters (two core-clusters intersect if a vertical line intersects both); for node  $a$  we also denote its LP-cluster by horizontal dotted lines. All other depicted nodes are not descendants of  $a$ . The diamond-shaped nodes  $\{c, f, g, j\}$  are contained in  $M$ , colored nodes  $\{b, d, e, g\}$  are contained in  $K'$ , and circle-shaped nodes  $\{b, d\}$  are contained in  $K \subseteq K'$ .

Note  $\Delta^-(u) \subseteq S_{K^+}$ . Next we claim for each node  $v \in K$ ,  $C(v) \cap L(u) = \emptyset$ . Now if we can prove this claim then it implies  $S_K \cap L(u) = \emptyset$  and thus we can write  $\Delta^-(u) \subseteq S_{K^+} \setminus S_K$ . Next we prove the claim. Notice that for any node  $v \in M$ ,  $v$  is not a descendant of  $u$  but  $C(v) \cap L(u) \neq \emptyset$ ; thus there always exists a node  $w \in K$  such that  $w$  is an ancestor of  $v$  and  $C(w) \cap L(u) = \emptyset$ .

Now, for the sake of contradiction, assume there exists a node  $w \in K$  such that  $C(w) \cap L(u) \neq \emptyset$ . But then  $w \in M$ , and following the previous argument there exists a node  $w' \in K$  such that  $w'$  is an ancestor of  $w$  and  $C(w') \cap L(u) = \emptyset$ . This is a contradiction, as by construction both  $w$  and  $w'$  cannot be present in  $K$ .

Furthermore, notice that no node  $w \in K$  is at level  $t(w) = t(u)$ , as that would imply a child  $w' \in M$  of  $w$ ; but  $C(w')$  intersects  $C(w)$  (and therefore  $L(w)$ ) as  $w'$  is a child of  $w$ , and  $C(w')$  intersects  $L(u)$  since  $w \in M$ . This is a contradiction, by Lemma 7.

We conclude that  $K$  contains nodes at level at most  $t(u) - 1$ , which allows us to apply the inductive hypothesis  $|\Delta^+(w)| \leq 0.3|C(w)|$  for nodes  $w \in K$ . Thus, from  $\Delta^-(u) \subseteq S_{K^+} \setminus S_K$  we get

$$|\Delta^-(u)| \leq 0.3|S_K| \quad (19)$$

Furthermore, all nodes in  $K$  have a child whose core-cluster intersects  $C(u)$ , and so by Lemma 6 the LP-distance between a species  $i \in L(u)$  and a species  $j \in S_K$  is small,  $x_{i,j}^{t(u)} < 0.6$ . By Lemma 5 we get  $|S_K| < \frac{1}{6}|L(u)|$ , which gives us  $|\Delta^-(u)| \leq \frac{1}{6} \cdot 0.3|L(u)|$ .

By the definition of  $\Delta^-(u) = L(u) \setminus C(u)$  we get  $|C(u)| \geq (1 - \frac{1}{6} \cdot 0.3)|L(u)|$ , by which

$$|\Delta^-(u)| \leq \frac{\frac{1}{6} \cdot 0.3}{1 - \frac{1}{6} \cdot 0.3} |C(u)| \quad (20)$$

which concludes the proof of claim (17), and as previously argued, the proof of claim (9).

As a byproduct of this analysis, we can also give some lower bounds on the LP cost.

**Lemma 16.** *Given a node  $u \in \mathcal{F}$ ,  $\text{cost}_{I(u)}^{(t(u))} = \Omega(\delta^{(t(u))}|L(u)||\Delta^-(u)|)$ .*

*Proof.* Fix a  $j \in S_K$ , as defined in (18). We have shown that for all  $i \in L(u)$ , the LP-distance with  $j$  is small,  $x_{i,j}^{(t(u))} < 0.6$ . If  $j \in S_K$  is not in the input-cluster  $I(u)$ , then  $\text{cost}_{i,j}^{(t(u))} = \delta^{(t(u))}(1 - x_{i,j}^{(t(u))}) > \delta^{(t(u))}(1 - 0.6)$  for each  $\{i, j\}$  pair with  $i \in L(u)$ . Else, it was removed from the input-cluster in the LP-Cleaning step,  $\text{cost}_j^{(t(u))} = \Omega(\delta^{(t(u))}|I(u)|)$  by Lemma 8.

By the algorithm,  $I(u) \supseteq L(u)$ , so summing up these costs gives  $\text{cost}_{I(u)}^{(t(u))} = \Omega(\delta^{(t(u))}|L(u)||S_K|)$  which is  $\Omega(\delta^{(t(u))}|L(u)||\Delta^-(u)|)$  by (19).  $\square$

**Lemma 17.** *Given a node  $u \in \mathcal{F}$ ,  $\text{cost}_{I(u)}^{(t(u))} = \Omega(\delta^{(t(u))}|C(u)||\Delta^+(u)|)$ .*

*Proof.* Let  $S_J, S_{J+}, S_R$  be defined as in (10). It holds that  $\Delta^+(u) = S_J \cup S_{J+} \cup S_R$  by (11) and these three sets are pairwise disjoint by definition. By (14), the size of  $S_{J+}$  is small compared to  $|S_J|$  which implies that  $|\Delta^+(u)| = \Omega(|S_J \cup S_R|)$ . Furthermore, by (12), for any  $i \in C(u), j \in S_J \cup S_R$ , we have that their LP-distance is small, that is  $x_{i,j}^{(t)} < 0.6$ .

We fix such a  $j \in S_J \cup S_R$ , therefore  $j \notin C(u)$ . If  $j \in S_J \cup S_R$  is not in the input-cluster  $I(u)$ , then  $\text{cost}_{i,j}^{(t(u))} = \delta^{(t(u))}(1 - x_{i,j}^{(t(u))}) > \delta^{(t(u))}(1 - 0.6)$  for each  $\{i, j\}$  pair with  $i \in C(u)$ . Else,  $j \in I(u)$ , but  $j \notin L(u)$ . That is because,  $j$  is not in  $C(u)$ , and if it was in  $L(u)$  then it would contradict Corollary 13. Therefore  $j$  was removed from the input-cluster in the LP-Cleaning step (Line 4 of Algorithm 2), and  $\text{cost}_j^{(t(u))} = \Omega(\delta^{(t(u))}|I(u)|)$  by Lemma 8. Summing these costs proves our claim.  $\square$

## 5.5 Approximation factor

In this section we prove that Algorithm 1 is an  $O(1)$  approximation of the LP cost.

We first make some definitions. Let  $t \in [\ell]$ . An input-cluster  $C_I \in Q^{(t)}$  is strong if there exists a level- $t$  node  $u \in \mathcal{F}$  such that  $I(u) = C_I$ . Similarly, a part  $P$  of the output partition  $P^{(t)}$  is strong if there exists a level- $t$  node  $u \in \mathcal{F}$  such that  $C^+(u) = P$ . In both cases we say that  $u$  is the corresponding node. We characterize an input-cluster as weak if it is not strong, and similarly a part of the output partition  $P^{(t)}$  as weak if it is not strong.

We start with upper bounding the cost of Algorithm 1. The upper bound is related to the input-clusters (distinguishing between strong and weak) and the parts of the output partitions (again distinguishing between strong and weak). Informally, for weak input-clusters and weak parts, the cost of our algorithm is proportional to the sum of squares of their size. For a strong input-cluster with corresponding node  $u$ , the cost of our algorithm is proportional to its size times the number of species of the input-cluster that did not end up in  $u$ 's core-cluster. For a strong part of the output partitions, the cost of our algorithm is proportional to its size times the number of its species that did not end up in  $u$ 's core-cluster.

**Lemma 18.** *Suppose we are given a Hierarchical Cluster Agreement instance  $S, Q^{(*)}, \delta^{(*)}$  and LP-distances  $x^{(*)}$ . Then the cost of the output of Algorithm 1 at level  $t$  is at most*

$$\delta^{(t)} \left( \sum_{\substack{C_I \in Q^{(t)} \\ C_I \text{ is weak}}} \binom{|C_I|}{2} + \sum_{\substack{P \in P^{(t)} \\ P \text{ is weak}}} \binom{|P|}{2} + \sum_{\substack{u \in \mathcal{F} \\ t(u)=t}} (|I(u) \setminus C(u)||I(u)| + |\Delta^+(u)||C^+(u)|) \right)$$



*Proof.* The cost at level  $t$  is  $\delta^{(t)}$  times the number of pairs  $\{i, j\}$  that do not end up in the same part of the output partition  $P^{(t)}$  but  $\{i, j\} \in \mathcal{E}(Q^{(t)})$ , plus the number of pairs  $\{i, j\}$  that end up in the same part of the output partition but  $\{i, j\} \notin \mathcal{E}(Q^{(t)})$ . This is

$$\begin{aligned}
& \delta^{(t)} |\mathcal{E}(Q^{(t)}) \setminus \mathcal{E}(P^{(t)})| + \delta^{(t)} |\mathcal{E}(P^{(t)}) \setminus \mathcal{E}(Q^{(t)})| \\
&= \delta^{(t)} \sum_{C_I \in Q^{(t)}} \left| \binom{C_I}{2} \setminus \mathcal{E}(P^{(t)}) \right| + \delta^{(t)} \sum_{P \in P^{(t)}} \left| \binom{P}{2} \setminus \mathcal{E}(Q^{(t)}) \right| \\
&= \delta^{(t)} \sum_{\substack{C_I \in Q^{(t)} \\ C_I \text{ is weak}}} \left| \binom{C_I}{2} \setminus \mathcal{E}(P^{(t)}) \right| + \delta^{(t)} \sum_{\substack{C_I \in Q^{(t)} \\ C_I \text{ is strong}}} \left| \binom{C_I}{2} \setminus \mathcal{E}(P^{(t)}) \right| \\
&+ \delta^{(t)} \sum_{\substack{P \in P^{(t)} \\ P \text{ is weak}}} \left| \binom{P}{2} \setminus \mathcal{E}(Q^{(t)}) \right| + \delta^{(t)} \sum_{\substack{P \in P^{(t)} \\ P \text{ is strong}}} \left| \binom{P}{2} \setminus \mathcal{E}(Q^{(t)}) \right|
\end{aligned}$$

Notice that if  $i, j$  are in the same core-cluster  $C(u)$  of some node  $u$  at level  $t(u) = t$ , then  $\{i, j\} \in \mathcal{E}(Q^{(t)}) \cap \mathcal{E}(P^{(t)})$ . Also, for each strong input-cluster there exists a corresponding node  $u$ , and vice-versa (similarly for strong parts of the output-partitions). Therefore:

$$\begin{aligned}
\delta^{(t)} \sum_{\substack{C_I \in Q^{(t)} \\ C_I \text{ is strong}}} \left| \binom{C_I}{2} \setminus \mathcal{E}(P^{(t)}) \right| &\leq \delta^{(t)} \sum_{\substack{u \in \mathcal{F} \\ t(u)=t}} \left( \binom{|I(u)|}{2} - \binom{|C(u)|}{2} \right) \\
\delta^{(t)} \sum_{\substack{P \in P^{(t)} \\ P \text{ is strong}}} \left| \binom{P}{2} \setminus \mathcal{E}(Q^{(t)}) \right| &\leq \delta^{(t)} \sum_{\substack{u \in \mathcal{F} \\ t(u)=t}} \left( \binom{|C^+(u)|}{2} - \binom{|C(u)|}{2} \right)
\end{aligned}$$

For an input cluster  $C_I$  with a corresponding node  $u$  at level  $t$  such that  $I(u) = C_I$ , and since always  $C(u) \subseteq I(u)$

$$\binom{|I(u)|}{2} - \binom{|C(u)|}{2} = |I(u) \setminus C(u)| |I(u)| - \binom{|I(u) \setminus C(u)|}{2} \leq |I(u) \setminus C(u)| |I(u)|$$

Notice that subtraction is needed since  $|I(u) \setminus C(u)| |I(u)|$  double-counts the pairs in  $\binom{I(u) \setminus C(u)}{2}$ .

Similarly, for a part  $P$  of the output partition  $P^{(t)}$  with a corresponding node  $u$  at level  $t$  such that  $C^+(u) = P$ , and since always  $C(u) \subseteq C^+(u)$

$$\begin{aligned}
\binom{|C^+(u)|}{2} - \binom{|C(u)|}{2} &= |C^+(u) \setminus C(u)| |C^+(u)| - \binom{|C^+(u) \setminus C(u)|}{2} \leq |C^+(u) \setminus C(u)| |C^+(u)| \\
&= |\Delta^+(u)| |C^+(u)|
\end{aligned}$$

□

For each term of Lemma 18, we show a matching lower bound for the LP cost. First, we give a lower bound of the LP cost related to the weak input-clusters.

**Lemma 19.** *The LP cost at level  $t$   $cost^{(t)}$  is*

$$\Omega \left( \delta^{(t)} \sum_{\substack{C_I \in Q^{(t)} \\ C_I \text{ is weak}}} \binom{|C_I|}{2} \right)$$

*Proof.* Let  $C_I \in Q^{(t)}$  be an input-cluster and  $C_{LP}$  be the corresponding LP-cluster by Algorithm 2. By Lemma 8,  $cost_{C_I \setminus C_{LP}}^{(t)} = \Omega(\delta^{(t)} |C_I \setminus C_{LP}| |C_I|)$ .

If  $C_I$  has no corresponding node  $u$  with  $I(u) = C_I, t(u) = t$ , this means that  $|C_{LP}| < 0.9|C_I|$ , therefore  $|C_I \setminus C_{LP}| = \Omega(|C_I|)$ , which makes the aforementioned cost  $\Omega(\delta^{(t)} |C_I|^2) = \Omega(\delta^{(t)} \binom{|C_I|}{2})$ .

Summing over all these input-clusters may only double-count each pair, which completes the proof.  $\square$

Next, we give a lower bound of the LP cost related to the strong input-clusters.

**Lemma 20.** *The LP cost at level  $t$   $cost^{(t)}$  is*

$$\Omega \left( \delta^{(t)} \sum_{\substack{u \in \mathcal{F} \\ t(u)=t}} (|I(u) \setminus C(u)| |I(u)|) \right)$$

*Proof.* Summing the cost of Lemma 16 over all nodes  $u$  at level  $t(u) = t$  gives a cost of

$$\Omega \left( \delta^{(t)} \sum_{\substack{u \in \mathcal{F} \\ t(u)=t}} (|L(u) \setminus C(u)| |L(u)|) \right)$$

since we may only double-count some pairs. Similarly, summing the cost of Lemma 8 over all species in such nodes gives a cost of

$$\Omega \left( \delta^{(t)} \sum_{\substack{u \in \mathcal{F} \\ t(u)=t}} (|I(u) \setminus L(u)| |I(u)|) \right)$$

We prove our claim by summing these two, and noticing that  $|L(u)| \geq 0.9|I(u)|$  by Line 5 of Algorithm 2.  $\square$

The following lemma lower bounds the LP cost in relation to the strong parts of the output partition  $P^{(t)}$ .

**Lemma 21.** *The LP cost at level  $t$   $cost^{(t)}$  is*

$$\Omega \left( \delta^{(t)} \sum_{\substack{u \in \mathcal{F} \\ t(u)=t}} (|\Delta^+(u)| |C^+(u)|) \right)$$

*Proof.* Summing the cost of Lemma 17 over all nodes  $u$  at level  $t(u) = t$  proves our claim, since we may only double-count some pairs.  $\square$

The following lemma lower bounds the LP cost in relation to the weak parts of the output partition  $P^{(t)}$ .

**Lemma 22.** *The LP cost at level  $t$   $cost^{(t)}$  is*

$$\Omega \left( \delta^{(t)} \sum_{\substack{P \in P^{(t)} \\ P \text{ is weak}}} \binom{|P|}{2} \right)$$

*Proof.* Fix any such part  $P$ . By Algorithm 3, each part of the output-partition  $P^{(t)}$  corresponds to the extended-cluster of some node  $u \in \mathcal{F}$ . We use  $C_P$  to refer to the core-cluster of this node corresponding to  $P$ , and notice that  $|C_P| = \Omega(|P|)$  by (9). Furthermore, by Lemma 4 any two species  $i, j \in C_P \subseteq P$  have  $x_{i,j}^{(t)} < 0.2$ .

We take two cases based on whether there exists an input-cluster  $C_I \in Q^{(t)}$  such that  $|C_I \cap C_P| > |C_P|/2$ . Since  $Q^{(t)}$  is a partition, there may be at most one such  $C_I$  for each part  $P$ . If none exists, then there exist  $\Omega(\binom{|C_P|}{2})$  pairs of species in  $C_P$  which belong in different input-clusters, and thus  $cost_{C_P}^{(t)} = \Omega(\delta^{(t)} \binom{|C_P|}{2}) = \Omega(\delta^{(t)} \binom{|P|}{2})$  by (9).

For the remaining parts, we first partition them based on parts that have the same corresponding input-cluster. Let  $P_1, \dots, P_k$  be such a maximal group with the same corresponding input cluster  $C_I$ , meaning that  $|C_I| = \Omega(\sum_{r=1}^k |C_{P_r}|) = \Omega(\sum_{r=1}^k |P_r|)$ . If  $C_I$  does not correspond to any node  $u$  at level  $t$  such that  $I(u) = C_I$ , then  $cost_{C_I}^{(t)} = \Omega(\delta^{(t)} \binom{|C_I|}{2})$  by Lemma 8, which is  $\Omega(\delta^{(t)} \sum_{r=1}^k \binom{|P_r|}{2})$ .

Else there exists such a node  $u$  with  $I(u) = C_I$ , while by the statement of our Lemma there is no  $v$  such that  $C^+(v) = P_r$  for  $r \in [k]$ . Therefore all these parts are disjoint from  $C^+(u)$  (Corollary 12), and thus disjoint from  $C(u)$ . This implies

$$\bigcup_{r=1}^k C_{P_r} \cap I(u) \subseteq I(u) \setminus C(u)$$

Then

$$|I(u) \setminus C(u)| \geq \left| \bigcup_{r=1}^k C_{P_r} \cap I(u) \right| > \sum_{r=1}^k |C_{P_r}|/2 = \Omega\left(\sum_{r=1}^k |P_r|\right)$$

By Lemma 20 the LP cost at level  $t$  is  $\Omega(\delta^{(t)} |I(u) \setminus C(u)| |I(u)|) = \Omega(\delta^{(t)} \sum_{r=1}^k \binom{|P_r|}{2})$ .  $\square$

We combine all our lower bounds in the following corollary.

**Corollary 23.** *The LP cost at level  $t$  is*

$$\Omega \left( \delta^{(t)} \left( \sum_{\substack{C_I \in Q^{(t)} \\ C_I \text{ is weak}}} \binom{|C_I|}{2} + \sum_{\substack{P \in P^{(t)} \\ P \text{ is weak}}} \binom{|P|}{2} + \sum_{\substack{u \in \mathcal{F} \\ t(u)=t}} (|I(u) \setminus C(u)| |I(u)| + |\Delta^+(u)| |C^+(u)|) \right) \right)$$

*Proof.* Follows by summing the LP cost at level  $t$  by Lemmas 19, 20, 21, 22.  $\square$

We are now ready to combine all the aforementioned results:

**Lemma 24.** *Given a Hierarchical Cluster Agreement instance  $S, Q^{(*)}, \delta^{(*)}$  and LP-distances  $x^{(*)}$ , the output of  $\text{Derive-Hierarchy}(S, \text{LP-Cleaning}(S, Q^{(*)}, x^{(*)}))$  is a sequence of hierarchical partitions  $P^{(*)}$  with cost  $O(\text{cost}^{(*)})$ .*

*Proof.* By Corollary 12, the output is a sequence of hierarchical partitions of  $S$ .

For any level  $t$ , by Lemma 18 and Corollary 23 the cost of the output of  $\text{Derive-Hierarchy}(S, \text{LP-Cleaning}(S, Q^{(*)}, x^{(*)}))$  is within a constant factor from the LP cost. Summing over all levels  $t$  proves our lemma.  $\square$

Lemma 24 directly proves:

$$\text{HierClustAgree} = O(1) \quad (\text{E}) \text{ from Figure 1}$$

as Algorithm 1 simply picks  $x^{(*)}$  to be an optimal fractional solution to the LP relaxation.

Combining this with Inequality (C) concludes the proof of Theorem 3.  $\square$

## 6 Constant integrality gap

In this section, we prove that the LP formulation for Hierarchical Correlation Clustering (Section 2) has constant integrality gap. This directly extends to the integrality gap of the LP formulation used by Ailon and Charikar for ultrametrics [3], as the LP formulation for Hierarchical Correlation Clustering is a generalization of the one for ultrametrics (implicit in [3, 39], discussed in Section 7).

Notice that this is not direct from our algorithm, as for Hierarchical Correlation Clustering we do not directly work with the LP from Section 2; we rather reduce our problem to an instance of Hierarchical Cluster Agreement, and then round the LP of this instance.

We start with some definitions. Suppose we have an instance of Hierarchical Correlation Clustering  $S, \delta^{(*)} = (\delta^{(1)}, \dots, \delta^{(\ell)}), E^{(*)} = (E^{(1)}, \dots, E^{(\ell)})$ . We say that  $x$  is an LP vector if it consists of LP distances  $x_{i,j}$  satisfying the triangle inequality and being in the interval  $[0, 1]$ , for all species  $i, j \in S$ . For any  $E \subseteq \binom{S}{2}$ , we extend the previously used notion of LP cost as:

$$\text{cost}^{(t)}(E, x) = \sum_{\{i,j\} \in E} (x_{i,j}) + \sum_{\{i,j\} \notin E} (1 - x_{i,j})$$

Notice that for any LP vector  $x$  and edge-sets  $E_1, E_2 \subseteq \binom{S}{2}$ , we have that

$$\text{cost}^{(t)}(E_1, x) \leq \text{cost}^{(t)}(E_2, x) + \delta^{(t)} |E_1 \Delta E_2| \quad (21)$$

That is because only pairs in their symmetric difference may be charged differently by  $\text{cost}(E_1, x)$  and  $\text{cost}(E_2, x)$ , and the maximum such difference is  $\delta^{(t)}$ , as the LP-distances are between 0 and 1.

The LP formulation of Correlation Clustering, which is a special case of the formulation of Hierarchical Correlation Clustering, has constant integrality gap [17]. Therefore, for a Correlation Clustering instance  $S, E$ , integral solution  $Q$  whose cost is within a constant factor from the optimal integral solution  $OPT$ , and any  $t$  and LP vector  $x$ , it holds that

$$\delta^{(t)} |E \Delta Q| = O(\delta^{(t)} |E \Delta OPT|) = O(\text{cost}(E, x)) \quad (22)$$

Finally, let  $Q^{(*)} = (Q^{(1)}, \dots, Q^{(\ell)})$  be partitions of  $S$  such that for each  $t \in [\ell]$ ,  $Q^{(t)}$  is a solution to Correlation Clustering with input  $S, E^{(t)}$  whose cost is within a constant factor from the optimal. Let  $x^{(*)} = (x^{(1)}, \dots, x^{(\ell)})$  be  $\ell$  LP vectors satisfying (6) that are an optimal fractional solution to Hierarchical Correlation Clustering.

We need to prove that some integral solution  $P^{(*)} = (P^{(1)}, \dots, P^{(\ell)})$  to Hierarchical Correlation Clustering has cost within a constant factor of the optimal fractional solution. We pick  $P^{(*)} = \text{Derive-Hierarchy}(S, \text{LP-Cleaning}(S, Q^{(*)}, x^{(*)}))$ , that is the integral solution suggested by Lemma 24. Formally, we prove

$$\sum_{t=1}^{\ell} \delta^{(t)} |P^{(t)} \Delta E^{(t)}| = O \left( \sum_{t=1}^{\ell} \text{cost}(E^{(t)}, x^{(*)}) \right)$$

It holds that

$$\sum_{t=1}^{\ell} \delta^{(t)} |P^{(t)} \Delta E^{(t)}| \leq \sum_{t=1}^{\ell} \delta^{(t)} (|P^{(t)} \Delta \mathcal{E}(Q^{(t)})| + |\mathcal{E}(Q^{(t)}) \Delta E^{(t)}|)$$

By Lemma 24 we have that

$$\begin{aligned} \sum_{t=1}^{\ell} \delta^{(t)} |P^{(t)} \Delta \mathcal{E}(Q^{(t)})| &= O \left( \sum_{t=1}^{\ell} \text{cost}(\mathcal{E}(Q^{(t)}), x^{(*)}) \right) \\ &\leq O \left( \sum_{t=1}^{\ell} (\text{cost}(E^{(t)}, x^{(*)}) + |E^{(t)} \cap \mathcal{E}(Q^{(t)})|) \right) \end{aligned}$$

with the later following by (21). Therefore we bound  $\sum_{t=1}^{\ell} \delta^{(t)} |P^{(t)} \Delta E^{(t)}|$ :

$$\begin{aligned} \sum_{t=1}^{\ell} \delta^{(t)} |P^{(t)} \Delta E^{(t)}| &\leq \sum_{t=1}^{\ell} \delta^{(t)} (|P^{(t)} \Delta \mathcal{E}(Q^{(t)})| + |\mathcal{E}(Q^{(t)}) \Delta E^{(t)}|) \\ &= O \left( \sum_{t=1}^{\ell} (\text{cost}(E^{(t)}, x^{(*)}) + |E^{(t)} \cap \mathcal{E}(Q^{(t)})|) \right) \\ &= O \left( \sum_{t=1}^{\ell} \text{cost}(E^{(t)}, x^{(*)}) \right) \end{aligned}$$

with the last step following from (22).

## 7 From $L_1$ -fitting ultrametrics to hierarchical correlation clustering

For completeness, we here review the reduction from ultrametrics to hierarchical correlation clustering implicit in previous work [3, 39].

Given an  $L_1$ -fitting ultrametrics instance with input  $\mathcal{D} : \binom{S}{2} \rightarrow \mathbb{R}_{>0}$ , we construct an input to the Hierarchical Correlation Clustering instance as follows. Let  $D^{(1)} < \dots < D^{(\ell+1)}$  be the distances that appear in the input distance function  $\mathcal{D}$ . For  $t = 1, \dots, \ell$ , define

$$\delta^{(t)} = D^{(t+1)} - D^{(t)} \text{ and } E^{(t)} = \left\{ \{i, j\} \in \binom{S}{2} \mid \mathcal{D}(i, j) \leq D^{(t)} \right\} \quad (23)$$

Now given the solution to this hierarchical correlation clustering problem, to construct a corresponding ultrametric tree, we first complete the partition hierarchy with  $P^{(0)}$  partitioning  $S$  into singletons and  $P^{(\ell+1)}$  consisting of the single set  $S$ . Moreover, we set  $\delta^{(0)} = D^{(1)}$ .

To get the ultrametric tree  $U$ , we create a node for each set in the hierarchical partitioning. Next, for  $t = 0, \dots, \ell$ , the parent of a level  $t$  node  $u$  is the node on level  $t + 1$  whose set contains the set of  $u$ , and the length of the parent edge is  $\delta^{(t)}/2$ . Then nodes on level  $t$  are of height  $\sum_{i=0}^{t-1} \delta^{(i)}/2 = D^{(t)}/2$  and if two species have their lowest common ancestor on level  $t$ , then their distance is exactly  $D^{(t)}$ .

The construction is reversible in a manner that given any ultrametric tree  $U$  with leaf set  $S$  and all distances from  $\{D^{(1)}, \dots, D^{(\ell+1)}\}$ , we get the partitions  $P^{(1)}, \dots, P^{(\ell)}$  as follows. First, possibly by introducing nodes with only one child, for each species  $i$  we make sure it has ancestors of heights  $D^{(t)}/2$  for  $t = 1, \dots, \ell + 1$ . Then, for  $t = 1, \dots, \ell$ , we let partitions  $P^{(t)}$  consist of the sets of descendants for each level  $t$  node in  $U$ .

With this relation between  $U$  and  $P^{(1)}, \dots, P^{(\ell)}$ , it follows easily that they have the same cost relative to  $\mathcal{D}$  in the sense that

$$\sum_{t=1}^{\ell} \delta^{(t)} |E^{(t)} \Delta E(P^{(t)})| = \sum_{\{i,j\} \in \binom{S}{2}} |\text{dist}_U(i, j) - \mathcal{D}(i, j)|.$$

Thus, with (23), the hierarchical correlation clustering is equal to  $L_1$ -fitting ultrametrics with ultrametric distances from the set of different distances in  $\mathcal{D}$ .

Finally, from Lemma 1(a) in [39], we have that among all ultrametrics minimizing the  $L_1$  distance to  $\mathcal{D}$ , there is at least one using only distances from  $\mathcal{D}$ . This implies that an  $\alpha$ -approximation algorithm for hierarchical correlation clustering implies an  $\alpha$ -approximation algorithm for  $L_1$ -fitting ultrametrics, that is

$$\text{UltraMetric} \leq \text{HierCorrClust} \quad (\text{B}) \text{ from Figure 1}$$

Combining this with Theorem 3 concludes the second part of Theorem 1, namely that the  $L_1$ -fitting ultrametrics problem can be solved in deterministic polynomial time within a constant approximation factor.  $\square$

## 8 Tree metric to ultrametric

Agarwala et al. [2] reduced tree metrics to certain restricted ultrametrics. In fact, their reduction may make certain species have distance 0 in the final tree, which means that it is actually a reduction from tree pseudometrics<sup>8</sup> to certain restricted ultrametrics. In this section we show that the restrictions are not needed, and that the reduction can be made in a way that does not introduce zero-distances.

<sup>8</sup>Pseudometrics are a generalization of metrics that allow distance 0 between distinct species.

## 8.1 Tree pseudometric to (unrestricted) ultrametric

The claim from [2] is that approximating a certain restricted ultrametric within a factor  $\alpha$  can be used to approximate tree pseudometric within a factor  $3\alpha$ . Here we completely lift these restrictions for  $L_1$ , and show that they can be lifted for all  $L_p$  with  $p \in \{2, 3, \dots\}$  with an extra factor of at most 2.

We will need the well-known characterization of ultrametrics discussed in the introduction, that  $U$  is an ultrametric iff it is a metric and  $\forall \{i, j, k\} \in \binom{S}{3} : U(i, j) \leq \max\{U(i, k), U(k, j)\}$ .

For simplicity, we prove the theorem only for  $p < \infty$ , as for  $L_\infty$  the 3 approximation [2] cannot be improved by our theorem.

**Theorem 25.** *For any integer  $1 \leq p < \infty$ , a factor  $\alpha \geq 1$  approximation for  $L_p$ -fitting ultrametrics implies a factor  $3 \cdot 2^{(p-1)/p} \cdot \alpha$  approximation for  $L_p$ -fitting tree pseudometrics. In particular, for  $L_1$  it implies a factor  $3\alpha$ .*

*Proof (Extending proof from [2]).* The restriction from Agarwala et al. [2] is as follows. For every species  $i \in S$ , we have a “lower bound”  $\beta_i$ . Moreover, we have a distinguished species  $k \in S$  with an upper bound  $\gamma_k$ .

We want an ultrametric  $U$  such that

$$\begin{aligned} \gamma_k \geq U(i, j) &\geq \max\{\beta_i, \beta_j\} & \forall \{i, j\} \in \binom{S}{2} \\ \gamma_k = U(k, i) & & \forall i \in S \setminus \{k\}. \end{aligned}$$

We note that the conditions can only be satisfied if  $\gamma_k \geq \beta_i$  for all  $i \in S$ , so we assume this is the case. We can even have  $\beta_k = \gamma_k$ .

The result from [2] states that for any  $p$  and  $\mathcal{D} : \binom{S}{2} \rightarrow \mathbb{R}_{>0}$ , if we can minimize the restricted ultrametric  $L_p$  error within a factor  $\alpha$  in polynomial-time, then there is a polynomial-time algorithm that minimizes the tree pseudometric  $L_p$  error within a factor  $3\alpha$ .

We start with creating a new distance function  $\mathcal{D}'$ .

$$\mathcal{D}'(i, j) = \min\{\gamma_k, \max\{\mathcal{D}(i, j), \beta_i, \beta_j\}\}.$$

Intuitively, we squeeze  $\mathcal{D}'$  to satisfy the restrictions. For any restricted ultrametric  $U$ , the error between  $U$  and  $\mathcal{D}'$  can never be larger than the error between  $U$  and  $\mathcal{D}$ , no matter the norm  $L_p$ . Formally, since  $U$  is restricted, we have  $\max\{\beta_i, \beta_j\} \leq U(i, j) \leq \gamma_k$ .

- If  $\mathcal{D}(i, j) > \gamma_k$ , then  $\mathcal{D}'(i, j) = \gamma_k \geq U(i, j)$  and  $|U(i, j) - \mathcal{D}'(i, j)|^p < |U(i, j) - \mathcal{D}(i, j)|^p$ .
- If  $\mathcal{D}(i, j) < \max\{\beta_i, \beta_j\}$ , then  $\mathcal{D}'(i, j) = \max\{\beta_i, \beta_j\} \leq U(i, j)$  and  $|U(i, j) - \mathcal{D}'(i, j)|^p < |U(i, j) - \mathcal{D}(i, j)|^p$ .
- If  $\max\{\beta_i, \beta_j\} \leq \mathcal{D}(i, j) \leq \gamma_k$ , then  $\mathcal{D}'(i, j) = \mathcal{D}(i, j)$  and  $|U(i, j) - \mathcal{D}'(i, j)|^p = |U(i, j) - \mathcal{D}(i, j)|^p$ .

We now ask for an arbitrary ultrametric fit  $U'$  for  $\mathcal{D}'$ . With exactly the same reasoning, we can only improve the cost if we replace  $U'$  with

$$U(i, j) = \min\{\gamma_k, \max\{U'(i, j), \beta_i, \beta_j\}\}.$$

Clearly  $U$  now satisfies the restrictions (in the end of the proof we show that it is an ultrametric).

Our solution to  $L_p$ -fitting tree pseudometrics is to first create  $\mathcal{D}'$  from  $\mathcal{D}$ , obtain ultrametric  $U'$  by an  $\alpha$  approximation to  $L_p$ -fitting ultrametrics, and then obtain the restricted ultrametric  $U$  from  $U'$ . Finally we apply the result from [2] to get the tree pseudometric.

Let  $OPT_{\mathcal{D},R}$  be the closest restricted ultrametric to  $\mathcal{D}$ , and  $OPT_{\mathcal{D}'}$  be the closest ultrametric to  $\mathcal{D}'$ . It suffices to show that  $\|U - \mathcal{D}\|_p \leq 2^{(p-1)/p}\alpha\|OPT_{\mathcal{D},R} - \mathcal{D}\|_p$  (equivalently  $\|U - \mathcal{D}\|_p^p \leq 2^{p-1}\alpha^p\|OPT_{\mathcal{D},R} - \mathcal{D}\|_p^p$ ), and that  $U$  is indeed an ultrametric.

By the above observations, it holds that

$$\begin{aligned}\|\mathcal{D}' - U\|_p &\leq \|\mathcal{D}' - U'\|_p \leq \alpha\|\mathcal{D}' - OPT_{\mathcal{D}'}\|_p \leq \alpha\|\mathcal{D}' - OPT_{\mathcal{D},R}\|_p \implies \\ \|\mathcal{D}' - U\|_p^p &\leq \alpha^p\|\mathcal{D}' - OPT_{\mathcal{D},R}\|_p^p\end{aligned}$$

By definition of  $\mathcal{D}'$ , and since  $U$  is restricted, for any species  $i, j$  it holds  $\min\{\mathcal{D}(i, j), U(i, j)\} \leq \mathcal{D}'(i, j) \leq \max\{\mathcal{D}(i, j), U(i, j)\}$ . The proof follows by a direct case study of the 3 cases  $\mathcal{D}(i, j) \leq \max\{\beta_i, \beta_j\}$ ,  $\max\{\beta_i, \beta_j\} < \mathcal{D}(i, j) \leq \gamma_k$ ,  $\gamma_k < \mathcal{D}(i, j)$ . Therefore

$$|\mathcal{D}(i, j) - U(i, j)| = |\mathcal{D}(i, j) - \mathcal{D}'(i, j)| + |\mathcal{D}'(i, j) - U(i, j)|$$

For  $p \geq 1$  we have  $|x|^p + |y|^p \leq (|x| + |y|)^p$ , meaning  $|\mathcal{D}(i, j) - \mathcal{D}'(i, j)|^p + |\mathcal{D}'(i, j) - U(i, j)|^p \leq |\mathcal{D}(i, j) - U(i, j)|^p$ .

Moreover, by the convexity of  $|x|^p$  for real  $x$ , we get  $((x + y)/2)^p \leq (|x|^p + |y|^p)/2$ , meaning  $|\mathcal{D}(i, j) - U(i, j)|^p \leq 2^{p-1}(|\mathcal{D}(i, j) - \mathcal{D}'(i, j)|^p + |\mathcal{D}'(i, j) - U(i, j)|^p)$ . Therefore

$$|\mathcal{D}(i, j) - \mathcal{D}'(i, j)|^p + |\mathcal{D}'(i, j) - U(i, j)|^p \leq |\mathcal{D}(i, j) - U(i, j)|^p \leq 2^{p-1}(|\mathcal{D}(i, j) - \mathcal{D}'(i, j)|^p + |\mathcal{D}'(i, j) - U(i, j)|^p)$$

The same holds if we replace  $U$  with  $OPT_{\mathcal{D},R}$ , as we only used that  $U$  is restricted. We now have



$$\begin{aligned}
\|\mathcal{D} - U\|_p^p &= \sum_{\{i,j\} \in \binom{S}{2}} |\mathcal{D}(i,j) - U(i,j)|^p \\
&\leq \sum_{\{i,j\} \in \binom{S}{2}} 2^{p-1} (|\mathcal{D}(i,j) - \mathcal{D}'(i,j)|^p + |\mathcal{D}'(i,j) - U(i,j)|^p) \\
&= 2^{p-1} \left( \sum_{\{i,j\} \in \binom{S}{2}} |\mathcal{D}(i,j) - \mathcal{D}'(i,j)|^p + \|\mathcal{D}' - U\|_p^p \right) \\
&\leq 2^{p-1} \left( \sum_{\{i,j\} \in \binom{S}{2}} |\mathcal{D}(i,j) - \mathcal{D}'(i,j)|^p + \alpha^p \|\mathcal{D}' - OPT_{\mathcal{D},R}\|_p^p \right) \\
&\leq 2^{p-1} \alpha^p \left( \sum_{\{i,j\} \in \binom{S}{2}} |\mathcal{D}(i,j) - \mathcal{D}'(i,j)|^p + \|\mathcal{D}' - OPT_{\mathcal{D},R}\|_p^p \right) \\
&= 2^{p-1} \alpha^p \sum_{\{i,j\} \in \binom{S}{2}} (|\mathcal{D}(i,j) - \mathcal{D}'(i,j)|^p + |\mathcal{D}'(i,j) - OPT_{\mathcal{D},R}(i,j)|^p) \\
&\leq 2^{p-1} \alpha^p \sum_{\{i,j\} \in \binom{S}{2}} (|\mathcal{D}(i,j) - OPT_{\mathcal{D},R}(i,j)|^p) \\
&= 2^{p-1} \alpha^p \|\mathcal{D} - OPT_{\mathcal{D},R}\|_p^p
\end{aligned}$$

Finally, we need to prove that  $U$  inherits that it is an ultrametric. This is clear if we proceed in rounds; each round we construct a new ultrametric, and the last one will coincide with  $U$ .

More formally, let  $U_0 = U'$ . In the first  $|S|$  rounds, we take out a different  $i' \in S$  at a time, and let

$$U_r(i, j) = \max\{U_{r-1}(i, j), \beta_{i'}\}.$$

Suppose  $r > 0$  is the first round where  $U_r$  is not an ultrametric. Then there exists a triple  $\{i, j, k\}$  such that  $U_r(i, j) > \max\{U_{r-1}(i, k), U_{r-1}(k, j)\}$ . As we only increase distances, this may only happen if  $U_r(i, j) > U_{r-1}(i, j)$ . But this means that  $U_r(i, j) = \max\{\beta_{i'}, \beta_j\}$ , which is a lower bound on  $U_r(i, k)$  and  $U_r(k, j)$  by construction.

Finally,  $U$  is simply

$$U(i, j) = \min\{\gamma_k, U_{|S|}(i, j)\}.$$

Suppose there exists a triple  $\{i, j, k\}$  that now violates the ultrametric property, then it holds that

$$U(i, j) > \max\{U_{|S|}(i, k), U_{|S|}(k, j)\}$$

As we did not increase any distance, this means that both  $U(i, k) < U_{|S|}(i, k)$  and  $U(k, j) < U_{|S|}(k, j)$ ; but distances can only reduce to  $\gamma_k$  which is an upper bound on  $U(i, j)$  by construction.  $\square$

## 8.2 From tree metric to tree pseudometric

In this section we prove that in order to find a good tree metric, it suffices to find a good tree pseudometric. This is a minor detail that we add for completeness. Informally, the construction

simply replaces 0 distances with some parameter  $\epsilon$ , and accordingly adapts the whole metric. By making the parameter  $\epsilon$  very small, the cost is not significantly changed.

Technically, our main lemma is the following.

**Lemma 26.** *Given is a set  $S$ , a distance function  $\mathcal{D} : \binom{S}{2} \rightarrow \mathbb{R}_{>0}$ , a tree  $T$  with non-negative edge weights describing a tree pseudometric on  $S$ , and a parameter  $\alpha \in (0, 1]$ . In time polynomial in the size of  $T$  we can construct a tree  $T'$  with positive edge weights describing a tree metric on  $S$ , such that for any  $p \geq 1$ , it holds that  $\|T' - \mathcal{D}\|_p \leq (1 + \alpha)\|T - \mathcal{D}\|_p$ .*

*Proof.* We construct  $T'$  from  $T$  as follows. First, we contract all edges with weight 0. This may result in several species from  $S$  coinciding in the same node. For each such node  $u$  and species  $i$  coinciding with some other species in  $u$ , we create a new leaf-node  $u_i$  connected only with  $u$  with edge-weight  $\epsilon > 0$  (to be specified later). We identify  $i$  with  $u_i$ , instead of  $u$ .

$T'$  describes a tree metric on  $S$ , as by construction each species  $i \in S$  is identified with a distinct node in  $T'$ , and  $T'$  only contains positive edge-weights.

If  $T$  matches  $\mathcal{D}$  exactly, that is  $\|T - \mathcal{D}\|_p = 0$ , then no pair of species  $i, j \in S$  have  $\text{dist}_T(i, j) = 0$ , as  $\mathcal{D}(i, j) > 0$ . But then no species coincided in the same node due to the contractions, meaning that no distances changed, which proves our claim. From here on we assume that at least one pair has  $\text{dist}_T(i, j) \neq \mathcal{D}(i, j)$ .

To specify the parameter  $\epsilon$  we first make some definitions. Let  $Y$  be the set containing all species  $i \in S$  for which we created a new leaf node in  $T'$ . Moreover, let  $d_{min}$  be the smallest positive  $|\text{dist}_T(i, j) - \mathcal{D}(i, j)|$  among all  $i, j \in S$ . Then

$$\epsilon = \alpha d_{min} / (8|S|)$$

For any two species  $i, j$ , their distance stays the same, increases by  $\epsilon$ , or increases by  $2\epsilon$ . Therefore, for  $p = \infty$  we directly get  $\|T' - \mathcal{D}\|_p \leq \|T - \mathcal{D}\|_p + 2\epsilon$ . By definition of  $d_{min}$  we have also have  $\|T - \mathcal{D}\|_p \geq d_{min} \implies 2\epsilon \leq \alpha \|T - \mathcal{D}\|_p / (4|S|) < \alpha \|T - \mathcal{D}\|_p$ , which proves our claim. Therefore we can assume that  $p < \infty$ .

We start with a lower bound related to  $\|T - \mathcal{D}\|_p$ . By definition of  $Y$ , for any  $i \in Y$  there exists a  $j \in Y$  such that  $\text{dist}_T(i, j) = 0$ , meaning that  $|\text{dist}_T(i, j) - \mathcal{D}(i, j)| = |\mathcal{D}(i, j)| \geq d_{min}$ . Therefore

$$\|T - \mathcal{D}\|_p^p \geq \frac{|Y|}{2} d_{min}^p$$

We now upper bound  $\|T' - \mathcal{D}\|_p$ . If  $\text{dist}_T(i, j) \neq \mathcal{D}(i, j)$ , then  $|\text{dist}_T(i, j) - \mathcal{D}(i, j)| \geq d_{min}$  by definition of  $d_{min}$ . For the rest of the pairs  $i, j$ , if their distance increased then either  $i \in Y$  or  $j \in Y$ , by construction; thus there are at most  $|Y||S|$  such pairs. Using these observations, we take the following three cases:

$$\begin{aligned}
\sum_{\substack{\{i,j\} \in \binom{S}{2} \\ \text{dist}_{T'}(i,j) \neq \mathcal{D}(i,j)}} |dist_{T'}(i,j) - \mathcal{D}(i,j)|^p &\leq \sum_{\substack{\{i,j\} \in \binom{S}{2} \\ \text{dist}_T(i,j) \neq \mathcal{D}(i,j)}} (|dist_T(i,j) - \mathcal{D}(i,j)|^p + |2\epsilon|^p) \\
\sum_{\substack{\{i,j\} \in \binom{S}{2} \\ \text{dist}_T(i,j) = \mathcal{D}(i,j) \\ \text{dist}_T(i,j) = \text{dist}_{T'}(i,j)}} |dist_{T'}(i,j) - \mathcal{D}(i,j)|^p &= 0 \\
\sum_{\substack{\{i,j\} \in \binom{S}{2} \\ \text{dist}_T(i,j) = \mathcal{D}(i,j) \\ \text{dist}_T(i,j) < \text{dist}_{T'}(i,j)}} |dist_{T'}(i,j) - \mathcal{D}(i,j)|^p &\leq \sum_{\substack{\{i,j\} \in \binom{S}{2} \\ \text{dist}_T(i,j) = \mathcal{D}(i,j) \\ \text{dist}_T(i,j) < \text{dist}_{T'}(i,j)}} |2\epsilon|^p \leq |Y||S||2\epsilon|^p
\end{aligned}$$

Adding these 3 upper bounds  $\|T' - \mathcal{D}\|_p^p$  by

$$\sum_{\substack{\{i,j\} \in \binom{S}{2} \\ \text{dist}_T(i,j) \neq \mathcal{D}(i,j)}} (|dist_T(i,j) - \mathcal{D}(i,j)|^p + |2\epsilon|^p) + |Y||S||2\epsilon|^p$$

Using our lower bound and the definition of  $\epsilon$

$$|Y||S||2\epsilon|^p \leq 2^{p+1}|S||\epsilon|^p \|T - \mathcal{D}\|_p^p / d_{min}^p \leq (\alpha/2) \|T - \mathcal{D}\|_p^p$$

By the definition of  $\epsilon$  and  $d_{min}$ , for  $i, j$  such that  $\text{dist}_T(i, j) \neq \mathcal{D}(i, j)$  it holds that  $(|dist_T(i, j) - \mathcal{D}(i, j)|^p + |2\epsilon|^p) \leq (\alpha/2)|dist_T(i, j) - \mathcal{D}(i, j)|^p$ . Therefore, we get

$$\begin{aligned}
\|T' - \mathcal{D}\|_p^p &\leq \sum_{\substack{\{i,j\} \in \binom{S}{2} \\ \text{dist}_T(i,j) \neq \mathcal{D}(i,j)}} (1 + \alpha/2)|dist_T(i, j) - \mathcal{D}(i, j)|^p + (\alpha/2)\|T - \mathcal{D}\|_p^p \\
&= (1 + \alpha/2)\|T - \mathcal{D}\|_p^p + (\alpha/2)\|T - \mathcal{D}\|_p^p = (1 + \alpha)\|T - \mathcal{D}\|_p^p
\end{aligned}$$

□

Therefore, for any  $p \geq 1$ , we can approximate  $L_p$ -fitting tree metrics by using an approximation to  $L_p$ -fitting tree pseudometrics. The error is at most  $(1 + \alpha)$  times the approximation factor of the tree pseudometric, as any tree metric is also a tree pseudometric.

Setting  $\alpha = \frac{1}{|S|}$  and using the result from [2], we conclude that

$$\text{TreeMetric} \leq (3 + o(1)) \cdot \text{UltraMetric} \quad (\text{A}) \text{ from Figure 1}$$

This concludes the proof of Theorem 1. □

As a final note, in the case of  $L_0$  (that is, we count the number of disagreements between  $\mathcal{D}$  and  $T'$ ) one cannot hope for a similar result. To see this, let  $S$  be a set of species, and let  $c_1, c_2 \in S$  be two special species. The distance between any pair of species is 2, except if the pair contains either  $c_1$  or  $c_2$ , in which case the distance is 1. The optimal tree pseudometric simply sets the distance between  $c_1$  and  $c_2$  to 0, and preserves everything else (1 disagreement).

Any tree metric requires at least  $|S| - 3$  disagreements: we say that a non-special species is good if it has tree-distance 1 to both  $c_1$  and  $c_2$ , and bad otherwise. Bad species have distance different than 1 to at least one special species, while good species have distance less than 2 with each other; the disagreements minimize at  $|S| - 3$ , when there is either one or two good species.

## 9 APX-Hardness

The problems of  $L_1$ -fitting tree metrics and  $L_1$ -fitting ultrametrics are regarded as APX-Hard in the literature [3, 39]. However, we decided to include our own versions of these proofs for a multitude of reasons: First and foremost, [3] attributes the APX-hardness to [53], which is an unpublished Master thesis that is non-trivial to read. Also [39] claims that APX-Hardness of  $L_1$ -fitting ultrametrics follows directly by the APX-Hardness of Correlation Clustering [17]; but this is only true if all the distances in the ultrametric are in  $\{1, 2\}$ . Second, we think that our proofs are considerably simpler and more direct. Finally, our constant factor approximation algorithms for these problems make it important to have formal proofs of their APX-Hardness, since the combination settles that a constant factor approximation is best possible in polynomial time unless  $P=NP$ .

### 9.1 $L_1$ -fitting ultrametrics

The correlation clustering problem has been shown to be APX-Hard in [17]. As noted in [3, 39] correlation clustering is the same as the  $L_1$ -fitting ultrametrics in case both the input and the output are only allowed to have distances in  $\{1, 2\}$ . We refer to this problem as  $L_1$ -fitting  $\{1, 2\}$ -ultrametrics. Therefore the  $L_1$ -fitting  $\{1, 2\}$ -ultrametrics is also APX-Hard.

For completeness, we sketch this relation here. Let  $E \subseteq \binom{S}{2}$  be an instance of correlation clustering, then  $\mathcal{D}(i, j)$  is an instance to  $L_1$ -fitting  $\{1, 2\}$ -ultrametrics, where  $\mathcal{D}(i, j) = 1$  if  $\{i, j\} \in E$ , and  $\mathcal{D}(i, j) = 2$  otherwise. Similarly, given  $\mathcal{D}$  we can obtain  $E$  by setting  $\{i, j\} \in S$  iff  $\mathcal{D}(i, j) = 1$ . Given any solution to correlation clustering (permutation  $P$  of  $S$ ), we get a solution  $T$  to  $L_1$ -fitting  $\{1, 2\}$ -ultrametrics with  $T(i, j) = 1$  if  $i, j$  are in the same part of  $P$ , and  $T(i, j) = 2$  otherwise. As  $T$  is an ultrametric, we are guaranteed that  $T(i, j) \leq \max\{T(i, k), T(j, k)\}$ , therefore if  $T(i, k) = T(j, k) = 1$ , then  $T(i, j) = 1$  as only distances in  $\{1, 2\}$  are allowed. Thus distance-1 is a transitive relation and  $P$  can be obtained by the equivalence classes of species with distance 1 in  $T$ . The observation from [3] is that  $|E \Delta \mathcal{E}(P)| = \|T - \mathcal{D}\|_1$ , which follows by trivial calculations.

The bird's eye view of our approach for showing APX-Hardness of  $L_1$ -fitting ultrametrics is the following. For the sake of contradiction, we assume that  $L_1$ -fitting ultrametrics is not APX-Hard. We then show how to solve the  $L_1$ -fitting  $\{1, 2\}$ -ultrametrics problem in polynomial time within any constant factor greater than 1, contradicting the fact that it is APX-Hard. The main idea is that we first solve the general  $L_1$ -fitting ultrametrics problem. Then we apply a sequence of local transformations that converts the general ultrametric to an ultrametric with distances in  $\{1, 2\}$  without increasing the error. To achieve this, we first eliminate distances smaller than 1, then eliminate distances larger than 2, and then eliminate distances in  $(1, 2)$ .

We first prove the following result concerning the local transformations. We remind the reader that an ultrametric  $T$  is defined as a metric with the property that for  $i, j, k \in S$  we have  $T(i, j) \leq \max\{T(i, k), T(j, k)\}$ .

**Lemma 27.** *Let  $S$  be a set of species,  $\mathcal{D} : \binom{S}{2} \rightarrow \{1, 2\}$  be a distance function with distances only in  $\{1, 2\}$ , and  $T$  be a rooted tree such that each species  $i \in S$  corresponds to a leaf in  $T$  (more than one species may correspond to the same leaf) and all leaves are at the same depth. Then, in polynomial time, we can create a tree  $T_{1,2}$  describing an ultrametric with distances only in  $\{1, 2\}$  such that  $\|T_{1,2} - \mathcal{D}\|_1 \leq \|T - \mathcal{D}\|_1$ .*

*Proof.* We set  $T' = T$  and apply the following local transformation to  $T'$ . If  $T(i, j) < 1$ , we set  $T'(i, j) = 1$ . It holds that  $\|T' - \mathcal{D}\|_1 \leq \|T - \mathcal{D}\|_1$  as  $\mathcal{D}(i, j) \geq 1$  and  $T(i, j) < 1$  implies

$|1 - \mathcal{D}(i, j)| < |T(i, j) - \mathcal{D}(i, j)|$ . Furthermore  $T'$  still describes an ultrametric. To see this, notice that  $\max\{T'(i, k), T'(j, k)\} \geq \max\{T(i, k), T(j, k)\} \geq T(i, j)$  as we do not decrease distances and  $T$  is an ultrametric. Therefore if  $T'(i, j) > \max\{T'(i, k), T'(j, k)\}$ , this means that  $T'(i, j) > T(i, j)$ . But this only happens if  $T'(i, j) = 1$ , which is a lower bound on  $T'(i, k), T'(j, k)$  by construction. This contradicts that  $T'(i, j) > \max\{T'(i, k), T'(j, k)\}$ , therefore  $T'$  describes an ultrametric. Notice that no two species in  $S$  coincide in the same node in  $T'$  as the minimum distance between any two distinct species is 1.

Similarly, we set  $T'' = T'$  and apply the following local transformation to  $T''$ . If  $T''(i, j) > 2$ , we set  $T''(i, j) = 2$ . It holds that  $\|T'' - \mathcal{D}\|_1 \leq \|T' - \mathcal{D}\|_1$  as  $\mathcal{D}(i, j) \leq 2$  and  $T'(i, j) > 2$  implies  $|2 - \mathcal{D}(i, j)| < |T'(i, j) - \mathcal{D}(i, j)|$ . Furthermore  $T''$  still describes an ultrametric. To see this, notice that  $T''(i, j) \leq T'(i, j) \leq \max\{T'(i, k), T'(j, k)\}$ . If  $T''(i, j) > \max\{T''(i, k), T''(j, k)\}$  then  $\max\{T''(i, k), T''(j, k)\} < \max\{T'(i, k), T'(j, k)\}$  which only happens if either of  $T'(i, k)$  or  $T'(j, k)$  dropped to 2, meaning that  $\max\{T''(i, k), T''(j, k)\} = 2$ . But 2 is an upper bound on  $T''(i, j)$ . This contradicts that  $T''(i, j) > \max\{T''(i, k), T''(j, k)\}$ , therefore  $T''$  describes an ultrametric.

Now, by construction, the ultrametric tree describing  $T''$  has leaves at depth 1 (the maximum distance is 2) and internal nodes at depth between 0 and 0.5 (the minimum distance is 1). If an internal node  $u$  has depth  $d_u \in (0, 0.5)$ , let  $x_1$  be the number of pairs  $\{i, j\} \subseteq \binom{S}{2}$  whose nearest common ancestor is  $u$  and  $\mathcal{D}(i, j) = 1$ , and  $x_2$  be the number of pairs  $\{i, j\} \subseteq \binom{S}{2}$  whose nearest common ancestor is  $u$  and  $\mathcal{D}(i, j) = 2$ . If  $x_2 \geq x_1$ , we remove  $u$  and connect the children of  $u$  directly with the parent of  $u$ . We still have an ultrametric as we have an ultrametric tree describing the metric. The  $L_1$  error is not larger, as the error of  $x_2$  pairs drops by twice the absolute difference in depths between  $u$  and its parent (their distance increases but does not exceed 2), and the error of  $x_1 \leq x_2$  pairs increases by the same amount. Otherwise  $x_2 < x_1$ . In this case we increase the depth of  $u$  until it coincides with the depth of some of its children, and merge these children with  $u$ . Similarly with the previous argument, we still have an ultrametric with smaller  $L_1$  error.

Each time we apply the above step, we remove at least one node from our tree. Therefore when we can no longer apply this step, we spent polynomial time and acquired an ultrametric  $T_{1,2}$  with distances only in  $\{1, 2\}$  whose  $L_1$  error from  $\mathcal{D}$  is  $\|T_{1,2} - \mathcal{D}\|_1 \leq \|T'' - \mathcal{D}\|_1 \leq \|T' - \mathcal{D}\|_1 \leq \|T - \mathcal{D}\|_1$ .  $\square$

**Theorem 28.**  *$L_1$ -fitting ultrametrics is APX-Hard. In particular,  $L_1$ -fitting ultrametrics where the input only contains distances in  $\{1, 2\}$  is APX-Hard.*

*Proof.* Let  $\mathcal{D} : \binom{S}{2} \rightarrow \{1, 2\}$  be a distance function,  $OPT$  be the optimal ultrametric for the  $L_1$ -fitting ultrametrics problem, and  $OPT_{1,2}$  be the optimal ultrametric for the  $L_1$ -fitting  $\{1, 2\}$ -ultrametrics. We solve this  $L_1$ -fitting ultrametrics instance in polynomial time and obtain  $T$  such that  $\|T - \mathcal{D}\|_1 \leq (1 + \epsilon)OPT$  for a sufficiently small constant  $\epsilon$ , as we assumed that  $L_1$ -fitting ultrametrics is not APX-Hard. Notice that any solution to the  $L_1$ -fitting  $\{1, 2\}$ -ultrametrics is also a solution to the  $L_1$ -fitting ultrametrics, meaning that  $\|T - \mathcal{D}\|_1 \leq (1 + \epsilon)OPT \leq (1 + \epsilon)OPT_{1,2}$ .

Let  $T_{1,2}$  be the ultrametric we get from  $T$  by applying Lemma 27. Then  $T_{1,2}$  is a solution to the  $L_1$ -fitting  $\{1, 2\}$ -ultrametrics instance, and  $\|T_{1,2} - \mathcal{D}\|_1 \leq \|T - \mathcal{D}\|_1 \leq (1 + \epsilon)OPT_{1,2}$ . This contradicts the fact that  $L_1$ -fitting  $\{1, 2\}$ -ultrametrics is APX-Hard.  $\square$

## 9.2 $L_1$ -fitting tree metrics

In this section, we show that  $L_1$ -fitting tree metrics is APX-Hard. Our reduction is based on the techniques used in [27] to prove NP-Hardness of the same problem. The bird's eye view of our

approach is that we solve  $L_1$ -fitting ultrametrics by solving  $L_1$ -fitting tree metrics on a modified instance. In this instance we introduce new species having small distance to each other and large distance to the original species. Through a sequence of local transformations, we show that we can modify the tree describing the obtained tree metric so as to consist of a star connecting the new species, and an ultrametric tree connecting the original species (the center of the star and the root of the ultrametric tree are connected by a large edge). This ultrametric would refute APX-Hardness of  $L_1$ -fitting ultrametrics, in case  $L_1$ -fitting tree metrics was not APX-Hard.

**Theorem 29.**  *$L_1$ -fitting tree metrics is APX-Hard.*

*Proof.* Let  $\mathcal{D} : \binom{S}{2} \rightarrow \{1, 2\}$  be an input to  $L_1$ -fitting ultrametrics, such that all distances in  $\mathcal{D}$  are in  $\{1, 2\}$ . Moreover, let  $n = |S|$  and  $OPT_{\mathcal{D}, U}$  be the ultrametric minimizing  $\|OPT_{\mathcal{D}, U} - \mathcal{D}\|_1$ . By Theorem 28, this problem is APX-Hard. For the sake of contradiction, assume  $L_1$ -fitting tree metrics is not APX-Hard.

Let  $\epsilon \in (0, 1)$  be a sufficiently small constant, and  $M = 2(1 + \epsilon)\binom{n}{2} + 1$  be a large value. We extend  $S$  to  $S' \supseteq S$  such that  $|S'| = 2n$ . For  $\{i, j\} \in \binom{S}{2}$  we set  $\mathcal{D}'(i, j) = \mathcal{D}(i, j)$ . For  $i, j \in \binom{S' \setminus S}{2}$  we set  $\mathcal{D}'(i, j) = 2$ . For all other  $i, j$  we set  $\mathcal{D}'(i, j) = M$ . As we assumed  $L_1$ -fitting tree metrics not to be APX-Hard, in polynomial time we can compute  $T$ , a tree metric such that for any other tree metric  $T_0$  it holds that  $\|T - \mathcal{D}'\|_1 \leq (1 + \epsilon)\|T_0 - \mathcal{D}'\|_1$ , for sufficiently small  $\epsilon$  such that  $0 < \epsilon < 1$ .

We first show that each species  $k \in S' \setminus S$  has an incident edge contained in all paths from this species to any species in  $S$ . To do so, we need to upper bound  $\|T - \mathcal{D}'\|_1$ . If we make a star whose leaves are the species in  $S$  with distance 1 from the center, a second star whose leaves are the species in  $S' \setminus S$  with distance 1 from the center, and connect the two centers with an edge of weight  $M - 2$  then only pairs with both species in  $S$  may have the wrong distance, and the error for each such pair is at most one. Therefore  $\|T - \mathcal{D}'\|_1 \leq (1 + \epsilon)\binom{n}{2}$ . This means that if  $k \in S' \setminus S$ , then in the tree describing  $T$  there exists a path  $\Pi_k$  starting from  $k$  and having weight larger than 1, such that the path from  $k$  to any species  $i \in S$  has  $\Pi_k$  as a prefix. To see why this is true, notice that otherwise two species  $i, j$  would exist such that the paths from  $k$  to  $i$  and from  $k$  to  $j$  only share a prefix  $\Pi_{i,j}$  of weight  $w_{\Pi_{i,j}} \leq 1$ . But  $T(i, k) > M/2$  as otherwise we would have  $\|T - \mathcal{D}'\|_1 \geq |T(i, k) - \mathcal{D}'(i, k)| \geq M/2 > (1 + \epsilon)\binom{n}{2}$ , and similarly  $T(j, k) > M/2$ . Then  $T(i, j) = T(i, k) + T(j, k) - 2 \cdot w_{\Pi_{i,j}} > M - 2$ , meaning again  $\|T - \mathcal{D}'\|_1 > |T(i, j) - \mathcal{D}'(i, j)| > (1 + \epsilon)\binom{n}{2}$ .

Using the aforementioned structural property, we show how to modify our tree so that all species in  $S$  are close to each other, all species in  $S' \setminus S$  are close to each other, but species in  $S$  are far from species in  $S' \setminus S$ . Let  $k \in S' \setminus S$  be the species minimizing  $\sum_{i \in S} |T(i, k) - \mathcal{D}'(i, k)|$ . We transform the tree describing  $T$  by inserting a node  $u$  in the path  $\Pi_k$  at distance 1 from  $k$ , and creating a star with  $u$  as its center and all species in  $S' \setminus S$  as leaves at distance 1. Let  $T'$  be the resulting tree metric and notice that  $\|T' - \mathcal{D}'\|_1 \leq \|T - \mathcal{D}'\|_1$  because the errors from species in  $S' \setminus S$  to species in  $S$  did not increase (by definition of  $k$ ), the errors between species in  $S' \setminus S$  are exactly zero, and the errors between species in  $S$  stay exactly the same (we did not modify the part of the tree formed by the union of paths between species in  $S$ ).

Then, we modify the tree describing  $T'$  to obtain  $T''$  so that the distance from any species in  $S$  to any species in  $S' \setminus S$  is  $M$ . If for any  $i \in S$  we have  $T'(i, k) \neq M$ , we move  $i$  in the tree so as to make its distance with  $k$  equal to  $M$ : if  $T'(i, k) < M$ , we create a new leaf node connected with  $i$  with distance  $M - T'(i, k)$ , and move  $i$  to this new leaf node. Else if  $T'(i, k) > M$  there exists an  $i'$  (possibly by subdividing an edge) in the path from  $k$  to  $i$  having distance  $M$  from  $k$  and we move  $i$  to this node. Notice that  $\|T'' - \mathcal{D}'\|_1 \leq \|T' - \mathcal{D}'\|_1$  because we move each  $i \in S$

by  $|M - T'(i, k)|$  so that it has zero error with each  $k' \in S' \setminus S$ , meaning that the error drops by  $|S' \setminus S| |M - T'(i, k)| = n |M - T'(i, k)|$  ( $|M - T'(i, k)|$  for each  $k' \in S' \setminus S$ ), and increases by at most  $(n - 1) |M - T'(i, k)|$  ( $|M - T'(i, k)|$  for each  $i' \in S \setminus \{i\}$ ).

If we remove all nodes not in a path from  $k$  to any  $i \in S$  in the tree describing  $T''$ , then by construction we have a tree  $T_{\mathcal{D},U}$  rooted at  $k$ , having leaves identified with the species in  $S$ , and all leaves having depth  $M$ . By the above discussion its error is  $\|T_{\mathcal{D},U} - \mathcal{D}\|_1 = \|T'' - \mathcal{D}\|_1$ . As some species may coincide in the same nodes, we get an ultrametric  $T'_{\mathcal{D},U}$  of  $S$  having the aforementioned properties so that no two species coincide in the same node, using Lemma 27.

Notice that  $OPT_{\mathcal{D},U}$  has maximum distance between species less than  $M$ ; otherwise its error would be at least  $M - 2$ , which is a contradiction to the fact that an ultrametric where all species have distance 1 has error at most  $\binom{n}{2} < M - 2$ . But then we can take the tree describing this optimal ultrametric, connect its root with a node  $u$  so that  $u$  has distance  $M - 1$  to all species in  $S$ , and identify each species  $k' \in S' \setminus S$  with a leaf  $u'_{k'}$  connected with  $u$  with an edge of weight 1. If the resulting tree metric is  $T_1$ , then  $\|OPT_{\mathcal{D},U} - \mathcal{D}\|_1 = \|T_1 - \mathcal{D}\|_1$ . We conclude that  $\|T'_{\mathcal{D},U} - \mathcal{D}\|_1 = \|T_{\mathcal{D},U} - \mathcal{D}\|_1 = \|T'' - \mathcal{D}\|_1 \leq \|T' - \mathcal{D}\|_1 \leq \|T - \mathcal{D}\|_1 \leq (1 + \epsilon) \|T_1 - \mathcal{D}\|_1 = (1 + \epsilon) \|OPT_{\mathcal{D},U} - \mathcal{D}\|_1$ . This contradicts Theorem 28.  $\square$

## 10 Conclusion

We have given the first constant factor approximation for  $L_1$ -fitting tree metrics, the first improvement on the problem for the last 16 years. This problem was one of the relatively few remaining problems for which obtaining a constant factor approximation or showing hardness was open. Breaking through the best known  $O((\log n)(\log \log n))$ -approximation had thus been stated as a fascinating open problem.

Interestingly, our journey brought us to the study of a natural definition of hierarchical cluster agreement that may be of broader interest, in particular to the data mining community where correlation clustering has been a successful objective function and where hierarchical clustering is often desired in practice.

Finding a polynomial time constant factor approximation (or showing that this is hard, e.g., by reduction to unique games) for  $L_2$ -fitting tree metrics is a great open problem. Recall from Section 8 that it suffices to focus on approximating the problem of fitting into an arbitrary ultrametric (no need for restricted versions). Finally, the  $O((\log n)(\log \log n))$ -approximation algorithm of Ailon and Charikar for the weighted case (where the cost of an edge is weighted by an input edge weight) could potentially be improved to  $O(\log n)$  without improving multicut, and it would be interesting to do so. Going even further would require improving the best known bounds for multicut, a notoriously hard problem.

## References

- [1] Amir Abboud, Vincent Cohen-Addad, and Hussein Houdrouge. Subquadratic high-dimensional hierarchical clustering. In *NeurIPS*, pages 11576–11586, 2019.
- [2] Richa Agarwala, Vineet Bafna, Martin Farach, Mike Paterson, and Mikkal Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM J. Comput.*, 28(3):1073–1085, 1999. Announced at SODA 1996.
- [3] Nir Ailon and Moses Charikar. Fitting tree metrics: Hierarchical clustering and phylogeny. *SIAM J. Comput.*, 40(5):1275–1291, 2011. Announced at FOCS 2005.
- [4] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. Announced at STOC 2005.
- [5] Noga Alon, Yossi Azar, and Danny Vainstein. Hierarchical clustering: A 0.585 revenue approximation. In *COLT*, volume 125, pages 153–162, 2020.
- [6] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. A discriminative framework for clustering via similarity functions. In *STOC*, pages 671–680, 2008.
- [7] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, 2004. Announced at FOCS 2002.
- [8] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS*, pages 184–193, 1996.
- [9] Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *SODA*, pages 737–756, 2014.
- [10] Gunnar E Carlsson and Facundo Mémoli. Characterization, stability and convergence of hierarchical clustering methods. *J. Mach. Learn. Res.*, 11(Apr):1425–1470, 2010.
- [11] L. L. Cavalli-Sforza and A. W. F. Edwards. Phylogenetic analysis models and estimation procedures. *The American Journal of Human Genetics*, 19:233–257, 1967.
- [12] James A. Cavender. Taxonomy with confidence. *Mathematical Biosciences*, 40(3):271–280, 1978.
- [13] Ines Chami, Albert Gu, Vaggos Chatziafratis, and Christopher Ré. From trees to continuous embeddings and back: Hyperbolic hierarchical clustering. In *NeurIPS*, 2020.
- [14] Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *SODA*, pages 841–854, 2017.
- [15] Moses Charikar, Vaggos Chatziafratis, and Rad Niazadeh. Hierarchical clustering better than average-linkage. In *SODA*, pages 2291–2304, 2019.
- [16] Moses Charikar, Vaggos Chatziafratis, Rad Niazadeh, and Grigory Yaroslavtsev. Hierarchical clustering for euclidean data. In *AISTATS*, pages 2721–2730, 2019.



- [17] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005. Announced at FOCS 2003.
- [18] Vaggos Chatziafratis, Grigory Yaroslavtsev, Euiwoong Lee, Konstantin Makarychev, Sara Ahmadian, Alessandro Epasto, and Mohammad Mahdian. Bisect and conquer: Hierarchical clustering via max-uncut bisection. In *AISTATS*, volume 108, pages 3121–3132, 2020.
- [19] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Comput. Complex.*, 15(2):94–114, 2006. Announced at CCC 2005.
- [20] Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In *STOC*, pages 219–228, 2015.
- [21] Michael Cochez and Hao Mou. Twister tries: Approximate hierarchical agglomerative clustering for average distance in linear time. In *SIGMOD*, pages 505–517, 2015.
- [22] Vincent Cohen-Addad, Rémi de Joannis de Verclos, and Guillaume Lagarde. Improving ultrametrics embeddings through coresets. In *ICML*, 2021.
- [23] Vincent Cohen-Addad, Varun Kanade, and Frederik Mallmann-Trenn. Hierarchical clustering beyond the worst-case. In *NeurIPS*, pages 6201–6209, 2017.
- [24] Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. *J. ACM*, 66(4):26:1–26:42, 2019. Announced at SODA 2018.
- [25] Vincent Cohen-Addad, Karthik C. S., and Guillaume Lagarde. On efficient low distortion ultrametric embedding. In *ICML*, volume 119, pages 2078–2088, 2020.
- [26] Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *STOC*, pages 118–127, 2016.
- [27] William H.E. Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49(4):461–467, 1987.
- [28] Kedar Dhamdhere. Approximating additive distortion of embeddings into line metrics. In *APPROX-RANDOM*, pages 96–104, 2004.
- [29] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004. Announced at STOC 2003.
- [30] Chenglin Fan, Anna C. Gilbert, Benjamin Raichel, Rishi Sonthalia, and Gregory Van Buskirk. Generalized metric repair on graphs. In *SWAT*, volume 162, pages 25:1–25:22, 2020.
- [31] Chenglin Fan, Benjamin Raichel, and Gregory Van Buskirk. Metric violation distance: Hardness and approximation. In *SODA*, pages 196–209. SIAM, 2018.

- [32] Martin Farach and Sampath Kannan. Efficient algorithms for inverting evolution. *J. ACM*, 46(4):437–449, 1999. Announced at STOC 1996.
- [33] Martin Farach, Sampath Kannan, and Tandy J. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13(1/2):155–179, 1995. Announced at STOC 1993.
- [34] James S. Farris. Estimating phylogenetic trees from distance matrices. *The American Naturalist*, 106(951):645–688, 1972.
- [35] Anna C. Gilbert and Lalit Jain. If it ain’t broke, don’t fix it: Sparse metric repair. In *55th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2017*, pages 612–619. IEEE, 2017.
- [36] Anna C. Gilbert and Rishi Sonthalia. Unsupervised metric learning in presence of missing data. In *56th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2018*, pages 313–321. IEEE, 2018.
- [37] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.
- [38] Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of algorithms*, 31(1):228–248, 1999. Announced at SODA 1998.
- [39] Boulos Harb, Sampath Kannan, and Andrew McGregor. Approximating the best-fit tree under  $l_p$  norms. In *APPROX-RANDOM*, pages 123–133, 2005.
- [40] Monika Rauch Henzinger, Valerie King, and Tandy J. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24(1):1–13, 1999. Announced at SODA 1996.
- [41] Jon M. Kleinberg and Éva Tardos. *Algorithm design*. Addison-Wesley, 2006.
- [42] Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.
- [43] Bin Ma, Lusheng Wang, and Louxin Zhang. Fitting distances by tree metrics with increment error. *J. Comb. Optim.*, 3(2-3):213–225, 1999.
- [44] Benjamin Moseley and Joshua Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *NeurIPS*, pages 3094–3103, 2017.
- [45] Benjamin Moseley and Joshua R. Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *NeurIPS*, pages 3094–3103, 2017.
- [46] Elchanan Mossel and Sébastien Roch. Learning nonsingular phylogenies and hidden markov models. In *STOC*, pages 366–375, 2005.
- [47] Aurko Roy and Sebastian Pokutta. Hierarchical clustering via spreading metrics. In *NeurIPS*, pages 2316–2324, 2016.
- [48] Anastasios Sidiropoulos, Dingkang Wang, and Yusu Wang. Metric embeddings with outliers. In *SODA*, pages 670–689, 2017.

- [49] Peter H.A. Sneath and Robert R. Sokal. Numerical taxonomy. *Nature*, 193(4818):855–860, 1962.
- [50] Peter H.A. Sneath and Robert R. Sokal. *Numerical Taxonomy. The Principles and Practice of Numerical Classification*. Freeman, 1963.
- [51] Rishi Sonthalia and Anna C. Gilbert. Tree! I am no tree! I am a low dimensional hyperbolic embedding. In *NeurIPS*, 2020.
- [52] Anke van Zuylen and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Math. Oper. Res.*, 34(3):594–620, 2009. Announced at SODA 2007.
- [53] Harold Todd Wareham. On the computational complexity of inferring evolutionary trees. Master’s thesis, Memorial University of of Newfoundland, 1993.
- [54] M.S. Waterman, T.F. Smith, M. Singh, and W.A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64(2):199–213, 1977.

## Appendix B

# Unpublished: Constrained Correlation Clustering: Deterministically and Combinatorially

# Constrained Correlation Clustering: Deterministically and Combinatorially

**Evangelos Kipouridis** ✉ 

Basic Algorithms Research Copenhagen (BARC), Denmark  
Department of Computer Science, University of Copenhagen, Denmark

**Jonas Klausen** ✉

Basic Algorithms Research Copenhagen (BARC), Denmark  
Department of Computer Science, University of Copenhagen, Denmark

**Mikkel Thorup** ✉ 

Basic Algorithms Research Copenhagen (BARC), Denmark  
Department of Computer Science, University of Copenhagen, Denmark

---

## Abstract


Correlation Clustering is one of the most successful clustering objectives. For each pair of nodes the input contains a preference related to whether we prefer the two endpoints to be in the same cluster or not. The output should be a clustering inducing the minimum number of violated preferences. In certain cases, however, the preference between some particular pairs may be too important to be violated. The constrained version of this problem also specifies pairs of nodes that must be in the same cluster and pairs of nodes that must not be in the same cluster (hard constraints). The output should satisfy all hard constraints and minimize the number of violated preferences.


Constrained Correlation Clustering is APX-Hard and has been approximated within a constant factor by van Zuylen et al. [SODA '07]. Their algorithm is based on modifying the preferences graph and applying a pivoting algorithm. Both the modification of the graph and the pivoting algorithm are based on rounding an LP. In this work, we show that the problem is also solvable combinatorially. A key ingredient in our approach is the first deterministic combinatorial algorithm for Correlation Clustering that is based on pivoting. In fact this algorithm achieves the best known approximation among all known deterministic combinatorial algorithms for Correlation Clustering, not just pivoting ones.

Furthermore, we introduce a natural node-weighted version of Correlation Clustering, which can be approximated within a constant factor using our results. As the general weighted version of Correlation Clustering would require a major breakthrough to approximate within a factor  $o(\log n)$ , Node-Weighted Correlation Clustering may be of independent interest.

**2012 ACM Subject Classification** Theory of computation → Facility location and clustering

**Keywords and phrases** Clustering, Constrained Correlation Clustering, Approximation

**Funding** This research is supported by Thorup's Investigator Grant from the Villum Foundation under Grant No. 16582, Basic Algorithms Research Copenhagen (BARC), Denmark. 

*Evangelos Kipouridis:* Evangelos Kipouridis has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 801199. 

## 1 Introduction

Clustering is a fundamental task related to unsupervised learning, with many applications in machine learning and data mining. The goal of clustering is to partition a set of objects into disjoint clusters, such that (ideally) all objects within a cluster are similar, and objects in different clusters are dissimilar. As no single definition best captures this high-level goal, a lot of different clustering objectives have been suggested.

Correlation Clustering is one of the most well studied such formulations for a multitude of reasons: its definition is simple and natural, it does not need the number of clusters to be part of the input, and it has found success in many applications. Some few examples include automated labeling [1, 8], clustering ensembles [6], community detection [11], disambiguation tasks [15], duplicate detection [3] and image segmentation [16, 22].

In Correlation Clustering, we are given a graph  $G = (V, E)$ , and the output is a partition  $C = \{C_1, \dots, C_k\}$  of the vertex-set  $V$ . We refer to the sets  $C_i$  of  $C$  as clusters. We define  $E_C = \bigcup_{i=1}^k \binom{C_i}{2}$ , that is we think of each cluster as a clique over its nodes. The goal is to minimize  $|E \Delta E_C|$ . In other words the goal is to approximate the input graph by a collection of cliques.

The intuition is the following: the input graph describes preferences, where an edge implies that we prefer its two endpoints to be clustered together, and a non-edge implies that we prefer them not to be clustered together. The output is a clustering (partition) of  $V$ , which defines a graph  $(V, E_C)$  such that two nodes share an edge if and only if they are in the same cluster. The definition of  $E_C$  makes the cost  $|E \Delta E_C|$  equal to the number of violated preferences (edges of  $G$  with endpoints in different clusters, and non-edges of  $G$  with endpoints in the same cluster).

## 1.1 Previous Results

The problem was initially introduced by Bansal et al. [5], who proved that it is NP-Hard, and provided a deterministic combinatorial<sup>1</sup> algorithm with an  $O(1)$  approximation factor, the constant being larger than 15,000. The improvements that followed were all based on rounding the natural LP: Charikar et al. gave a deterministic 4 approximation [9], while Ailon et al. gave a randomized 2.5 approximation and proved that the problem is APX-Hard [2]. Finally a deterministic 2.06 approximation was given by Chawla et al. [10]. The last result is near-optimal among algorithms rounding the natural LP, as its integrality gap is at least 2.

Given the importance of Correlation Clustering, research does not only focus on improving its approximation factor. Another important goal is the design of combinatorial algorithms; in the same paper with their 2.5 approximation [2], Charikar et al. design a randomized 3 approximation algorithm, which, despite its worse approximation, enjoys the benefit of being combinatorial. Similarly, much later than the 2.06 approximation [10], we have a combinatorial 6 approximation [20]. Deterministic algorithms is also an important direction. It is explicitly pursued by [19] and [20], and is also a significant part of the technical contribution of [10].

An interesting class of algorithms for Correlation Clustering is pivoting algorithms. These are algorithms that, based on some criterion, pick an object (the pivot) and set the first cluster to be the pivot along with all the objects that prefer to be with the pivot. Then these objects are removed and we recurse on the remaining ones. Such algorithms are very simple and enjoy properties that are crucial in certain applications [19]. In [2] a randomized combinatorial pivoting algorithm with a 3 approximation was given. In [19] a deterministic pivoting algorithm also achieved 3 approximation, and avoided randomization by rounding an LP. However, if we insist on both deterministic and combinatorial algorithms, no constant factor approximation based on pivoting is known.

Correlation Clustering has also been studied in different settings such as parameterized

---

<sup>1</sup> Throughout this work, we use the term combinatorial to refer to algorithms that do not use LP.

algorithms [14], sublinear and streaming algorithms [4], massively parallel computation (MPC) algorithms [12], and differentially private algorithms [7].

In the weighted version of Correlation Clustering, we are also given a weight for each preference. The final cost is then the sum of weights of the violated preferences. An  $O(\log n)$  approximation for weighted Correlation Clustering is known by Demaine et al. [13]. In the same paper they show that the problem is equivalent to Multicut, meaning that an  $o(\log n)$  approximation would require a major breakthrough. As efficiently approximating the general weighted version is out of reach, research has focused on special cases for which constant factor approximations are possible [17, 18].

Constrained Correlation Clustering is an interesting variant of Correlation Clustering capturing the idea of critical pairs of nodes. To address these situations, Constrained Correlation Clustering also introduces hard constraints. A clustering is valid if it satisfies all hard constraints, and the goal is to find a valid clustering of minimal cost.

Constrained Correlation Clustering can be phrased as a restrictive weighting of Correlation Clustering as well: we simply give infinite weight to pairs associated with a hard constraint, and weight 1 to all other pairs.

Our paper is mostly related to the work of van Zuylen et al. on Constrained Correlation Clustering, who designed a deterministic 3 approximation [19]. Their algorithm works in two steps.

1. The hard constraints are used to create an edge-set  $E'$  from input  $(V, E)$  such that any pivoting algorithm applied to  $(V, E')$  would result in a valid clustering of  $V$ .
2. It then treats  $(V, E')$  as an instance of (unconstrained) Correlation Clustering, ignoring the actual hard constraints. On this graph it applies a pivoting algorithm.

We note that the algorithm itself is not pivoting, as it first modifies the preferences graph; in fact no pivoting algorithm can exist for Constrained Correlation Clustering. It is however crucial that a pivoting algorithm is used on the *unconstrained* instance (step 2), as this is what ensures that no hard constraint is violated.

Throughout [19], the authors give deterministic algorithms for several problems, and whenever possible, combinatorial algorithms are preferred. Their solution for Constrained Correlation Clustering is, however, not combinatorial; in fact both the modification of the graph and the pivoting algorithm need to use the values of the LP variables.

## 1.2 Our contribution

In this work we solve Constrained Correlation Clustering deterministically and combinatorially. In order to achieve this, we give deterministic combinatorial counterparts for both steps of the algorithm in [19].

For the first step, our algorithm carefully modifies the input graph so that on one hand the optimal cost is not significantly changed, and on the other hand any pivoting algorithm on the transformed graph returns a clustering that respects all hard constraints. As we cannot use LP to lower bound the optimal value, we rely on subgraphs of the original graph for which we argue that any valid clustering needs to pay.

Concerning the second step, we design a deterministic combinatorial algorithm for (unconstrained) Correlation Clustering based on pivoting. No such algorithm was known before. In fact, the approximation factor of our algorithm is the best known among all deterministic combinatorial algorithms for Correlation Clustering, not just pivoting ones.

► **Theorem 1.** *There exists a deterministic combinatorial algorithm based on pivoting that solves Correlation Clustering in  $O(|V|^6)$  time with a 5.8 approximation factor. There exists a faster such algorithm running in  $O(|V|^3)$  time, with a 9 approximation.*

Exactly as in [19], our overall algorithm for Constrained Correlation Clustering is not pivoting as no pivoting algorithm can exist for Constrained Correlation Clustering; only the subroutine for unconstrained Correlation Clustering (step 2) is pivoting.

Concerning the effect of modifying the graph, roughly speaking we get that the final approximation factor is  $(2 + \sqrt{5}) \cdot \alpha + 3$ , where  $\alpha$  is the approximation factor of the pivoting algorithm we use. Along with two combinatorial pivoting algorithms (randomized 3 approximation from [2] and deterministic 5.8 and 9 approximation from Theorem 1) we get the first combinatorial algorithms for Constrained Correlation Clustering:

► **Theorem 2.** *There exists a randomized combinatorial algorithm that solves Constrained Correlation Clustering in  $O(|V|(|V| + |E|))$  time with an expected approximation factor less than 16.*

► **Theorem 3.** *There exists a deterministic combinatorial algorithm that solves Constrained Correlation Clustering in  $O(|V|^6)$  time with an approximation factor less than 29. There exists a faster such algorithm running in  $O(|V|^3)$  time, with a 42 approximation.*

A result of independent interest concerning the unconstrained Correlation Clustering is a lower bound on the approximation factor of any pivoting algorithm:

► **Theorem 4.** *No pivoting algorithm for Correlation Clustering has  $3 - \Omega(1)$  approximation factor.*

Notice that if we are satisfied with randomized pivoting algorithms, then Theorem 4 implies that the algorithm from [2] is optimal. Similarly, if we are satisfied with non-combinatorial pivoting algorithms, then the algorithm from [19] is optimal.

In this work we also introduce the Node-Weighted Correlation Clustering problem. As weighted Correlation Clustering is equivalent to Multicut, improving over the current  $\Theta(\log n)$  approximation seems out of reach. The advantage of our alternative type of weighted Correlation Clustering is that it is natural and approximable within a constant factor.

In Node-Weighted Correlation Clustering we assign weights to the nodes, rather than on pairs of nodes. Violating a preference between nodes  $u, v$  with weights  $w(u), w(v)$  costs  $w(u) \cdot w(v)$ . We provide a randomized combinatorial algorithm for Node-Weighted Correlation Clustering with a 3 approximation factor.

► **Theorem 5.** *There exists a randomized combinatorial algorithm for Node-Weighted Correlation Clustering with a 3 approximation factor. Its running time is  $O(|V| + |E|)$ .*

In order to solve Node-Weighted Correlation Clustering, we reduce it to a Constrained Correlation Clustering instance. Even though the size of the new instance may be exponentially larger, we show that its special structure allows us to solve it efficiently.

In fact, the final algorithm boils down to a modification of the pivoting algorithm from [2], where instead of sampling uniformly at random, we sample with probabilities proportional to the weights. However, it is not clear to us whether a modification of the original analysis could work, as it relies on the fact that the instance is unweighted<sup>2</sup>.

---

<sup>2</sup> For the more knowledgeable reader, in the original analysis it is enough that any clustering pays at least 1 for a bad triplet, no matter which of the 3 implied preferences is violated. In our weighted case however this is not enough, as the 3 preferences may have significantly different weights.



## 2 Preliminaries

We denote the set  $\{1, \dots, n\}$  by  $[n]$ . We denote all subsets of size  $k$  of a set  $S$  by  $\binom{S}{k}$ . The symmetric difference between two sets  $A, B$  is denoted by  $A \Delta B$ .

Given a graph  $G = (V, E)$  and two disjoint subsets  $U_1, U_2 \subseteq V$ , we denote the set of edges with one endpoint in  $U_1$  and the other in  $U_2$  by  $E(U_1, U_2)$ . The subgraph of  $G$  induced by vertex-set  $U_1$  is denoted by  $G[U_1]$ . We denote by  $K_{n_1, n_2}$  the complete bipartite graph on  $n_1$  vertices on one set of the bipartition and  $n_2$  on the other.

We use the terms clustering and partition interchangeably. We say that the edge-set of a clustering  $C = \{C_1, \dots, C_k\}$  of  $V$  is the set of pairs with both endpoints in the same set in  $C$ . More formally, the edge-set of  $C$  is  $\bigcup_{i=1}^k \binom{C_i}{2}$ .

We now formally define the problems of interest. We start with the simplest problem, (unweighted) Correlation Clustering.

► **Definition 6.** *Correlation Clustering:* The input is a set of nodes  $V$  and a set of edges  $E \subseteq \binom{V}{2}$ . The output is a clustering  $C = \{C_1, C_2, \dots, C_k\}$  of  $V$  with edge-set  $E_C$  minimizing  $|E \Delta E_C|$ .

An algorithm for Correlation Clustering is said to be *pivoting* if based on some criterion it picks an unclustered node  $u$  (which we call the pivot), creates a cluster containing  $u$  and all its unclustered neighbors in  $(V, E)$ , and repeats the same process until all nodes are clustered.

The constrained version of Correlation Clustering is defined as:

► **Definition 7.** *Constrained Correlation Clustering:* The input is a set of nodes  $V$ , a set of edges  $E \subseteq \binom{V}{2}$ , a set of friendly pairs  $F \subseteq \binom{V}{2}$  and a set of hostile pairs  $H \subseteq \binom{V}{2}$ . The output is a clustering  $C = \{C_1, C_2, \dots, C_k\}$  of  $V$  with edge-set  $E_C$  such that no pair  $\{u, v\} \in F$  has  $u, v$  in different clusters and no pair  $\{u, v\} \in H$  has  $u, v$  in the same cluster. The clustering  $C$  shall minimize  $|E \Delta E_C|$ .

We refer to the set of nodes of a connected component of  $(V, F)$  as a supernode. The set of supernodes partition  $V$  and is denoted by  $SN$ . Given a node  $u$ , we let  $s(u)$  be the unique supernode containing  $u$ . Two supernodes  $U, W$  are hostile if there exists a hostile pair  $\{u, w\}$  with  $u \in U, w \in W$ . Two supernodes  $U, W$  are connected if  $|E(U, W)| \geq 1$ . Two supernodes  $U, W$  are  $\beta$ -connected if  $|E(U, W)| \geq \beta|U||W|$ .

We also introduce Node-Weighted Correlation Clustering, a new related problem that may be of independent interest. It is defined as follows:

► **Definition 8.** *Node-Weighted Correlation Clustering:* The input is a set of nodes  $V$ , a set of edges  $E \subseteq \binom{V}{2}$ , and a weight function  $w : V \rightarrow \mathbb{N}_{>0}$ . The output is a clustering  $C = \{C_1, C_2, \dots, C_k\}$  of  $V$  with edge-set  $E_C$  minimizing

$$\sum_{\{u,v\} \in E \Delta E_C} w(u) \cdot w(v)$$

This is similar to Correlation Clustering, but in the end we pay  $w(u) \cdot w(v)$  (instead of 1) for each pair  $\{u, v\}$  violating a preference.

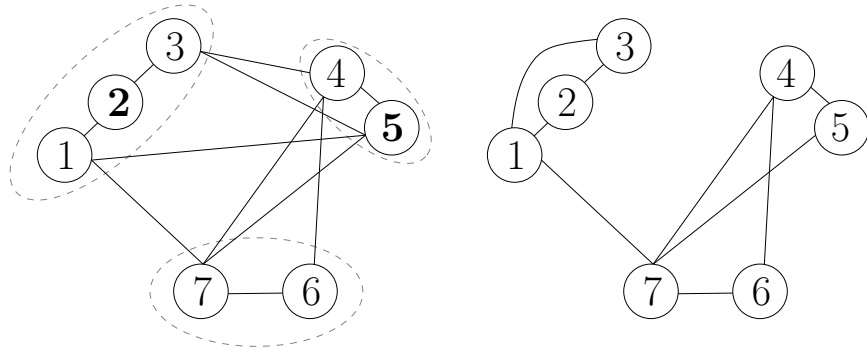
The model of computation for all problems considered is the standard word RAM, with word size  $w = \Theta(\log |V|)$ .

### 3 Combinatorial Algorithms for Constrained Correlation Clustering

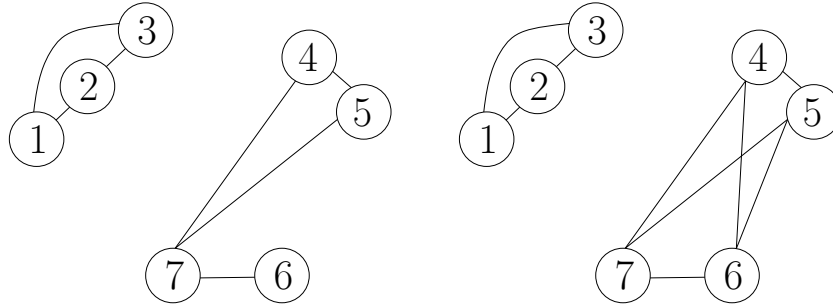
The first step of our combinatorial approach is to transform the graph  $G$  in a more manageable form  $G'$ . The high-level idea is that in  $G'$ :

1. if  $\{u, v\}$  is a friendly pair, then  $u$  and  $v$  are connected and have the same neighborhood.
2. if  $\{u, v\}$  is a hostile pair, then  $u$  and  $v$  are not connected and have no common neighbor.
3. an  $O(1)$  approximation of the  $G'$  instance is also an  $O(1)$  approximation of the  $G$  instance.

As was already noticed in [19], Properties 1 and 2 imply that a pivoting algorithm on  $G'$  gives a clustering satisfying the hard constraints. Along with Property 3 and the randomized pivoting  $O(1)$  approximation from [2], we prove Theorem 2. Similarly, using our deterministic pivoting  $O(1)$  approximation from Section 4, we prove Theorem 3.



(a) The original graph. The set of friendly pairs is  $F = \{\{1, 2\}, \{2, 3\}, \{4, 5\}, \{6, 7\}\}$ , and the only hostile pair in  $H$  is  $\{2, 5\}$ . (b) Line 3 introduces edge  $\{1, 3\}$ , and Line 4 disconnects the supernodes containing 2 and 5.



(c) Line 6 removes the pair of edges  $\{1, 7\}$  and  $\{4, 6\}$  because 1, 4 are in hostile supernodes while 6, 7 are in the same supernode. (d) Line 9 introduces all edges connecting supernodes  $\{4, 5\}$  and  $\{6, 7\}$  because there were enough edges between them already.

■ **Figure 1** Procedure TRANSFORM (Algorithm 1). For any two supernodes  $U_1, U_2$ , either all pairs with an endpoint in  $U_1$  and an endpoint in  $U_2$  share an edge, or none of them does. Furthermore all pairs within a supernode are connected and no hostile supernodes are connected.

Our algorithm works as follows. If some supernode is hostile to itself, then it outputs that no clustering satisfies the hard constraints. Else, starting from edge-set  $E$ , it adds all edges within a supernode. Then it drops all edges between hostile supernodes. Subsequently, it repeatedly detects hostile supernodes that are connected with the same supernode, and drops one edge from each such connection. Finally, for each  $\beta$ -connected pair of supernodes,

■ **Algorithm 1** The procedure MAIN is given a vertex-set  $V$ , an edge-set  $E$  describing the preferences, a set of friendly pairs  $F$ , a set of hostile pairs  $H$ , and a PIVOTING\_ALGORITHM for Correlation Clustering. It creates a new  $G'$  using procedure TRANSFORM and uses PIVOTING\_ALGORITHM on  $G'$  to return a clustering.

---

```

Procedure TRANSFORM( $G = (V, E), F, H$ )
1  |   Compute the connected components of  $(V, F)$ 
   |
   |   // Impossible iff some pair must both be and not be in the same
   |   cluster.
2  |   if  $\exists U \in SN$  hostile to itself then return  $G' = (\emptyset, \emptyset)$ 
   |
   |   // Connect nodes in the same supernode.
3  |    $E_1 \leftarrow E \cup \{\{u, v\} \in \binom{V}{2} \mid s(u) = s(v)\}$ 
   |
   |   // Disconnect pairs in hostile supernodes.
4  |    $E_2 \leftarrow E_1 - \{\{u, v\} \in \binom{V}{2} \mid s(u) \text{ and } s(v) \text{ are hostile}\}$ 
   |
   |   // While hostile supernodes  $U_1, U_2$  are both connected with supernode
   |    $U_3$ , drop an edge between  $U_1, U_3$  and an edge between  $U_2, U_3$ 
5  |    $E_3 \leftarrow E_2$ 
6  |   while  $\exists U_1, U_2, U_3 \in \binom{SN}{3}$  such that  $U_1, U_2$  are hostile and
   |    $\exists u_1 \in U_1, u_2 \in U_2, u_3 \in U_3, u'_3 \in U_3$  such that  $\{u_1, u_3\} \in E_3, \{u_2, u'_3\} \in E_3$  do
7  |   |    $E_3 \leftarrow E_3 - \{\{u_1, u_3\}, \{u_2, u'_3\}\}$ 
   |
   |   // Round connections between pairs of supernodes
8  |    $E_4 \leftarrow E_3$ 
9  |   foreach  $\{U_1, U_2\} \in \binom{SN}{2}$  do
10 |   |    $E_{U_1, U_2} \leftarrow \{\{u_1, u_2\} \mid u_1 \in U_1, u_2 \in U_2\}$ 
11 |   |   if  $|E_{U_1, U_2} \cap E_4| > \frac{3-\sqrt{5}}{2}|U_1||U_2|$  then  $E_4 \leftarrow E_4 \cup E_{U_1, U_2}$ 
12 |   |   else  $E_4 \leftarrow E_4 - E_{U_1, U_2}$ 
13 |   return  $G' = (V, E_4)$ 

Procedure MAIN( $G = (V, E), F, H, PIVOTING\_ALGORITHM$ )
14 |    $G' \leftarrow$  TRANSFORM( $G=(V, E), F, H$ )
15 |   if  $G' = (\emptyset, \emptyset)$  then return "Impossible"
16 |   return PIVOTING_ALGORITHM( $G'$ )

```

---

it connects all their nodes if  $\beta > \frac{3-\sqrt{5}}{2}$ , and disconnects them otherwise<sup>3</sup>.

From a high-level view, the first two modifications are directly related to the hard constraints: if  $u_1, u_2$  are friendly and  $u_2, u_3$  are friendly, then any valid clustering has  $u_1, u_3$  in the same cluster, even if a preference discourages it. Similarly, if  $u_1, u_2$  are friendly,  $u_3, u_4$

<sup>3</sup> The constant  $\frac{3-\sqrt{5}}{2}$  optimizes the approximation factor. The natural choice of 0.5 would still give a constant approximation factor, although slightly worse.

are friendly, but  $u_1, u_3$  are hostile, then any valid clustering has  $u_2, u_4$  in different clusters, even if a preference discourages it. Our first two modifications simply make the preferences consistent with the hard constraints.

The third modification guarantees that hostile supernodes share no common neighbor. Intuitively, a pivoting algorithm will thus never put their nodes in the same cluster, as the hostility constraints require. Concerning the cost, notice that if hostile supernodes  $U_1, U_2$  are connected with supernode  $U_3$ , then no valid clustering can put all three of them in the same cluster. Therefore we always need to pay either for the connections between  $U_1$  and  $U_3$ , or for the connections between  $U_2$  and  $U_3$ .

Finally, after rounding, for each pair of supernodes  $U_1, U_2$ , the edge-set  $E(U_1, U_2)$  is either empty or the full set of size  $|U_1||U_2|$ . This ensures that a pivoting algorithm always puts all nodes of a supernode in the same cluster, thus also obeying the friendliness constraints. Concerning the cost of the rounded instance, a case analysis shows that it is always within a constant factor of the cost of the instance before rounding.

Before proceeding with the approximation ratio and the correctness, we first prove the running time of our algorithm.

► **Lemma 9.** *The running time of Algorithm 1 is  $O(|V|(|V| + |E|) + T(|V|, \binom{|V|}{2}))$ , where  $T(n, m)$  is an upper bound on the running time of the pivoting algorithm we use on a graph with  $n$  nodes and  $m$  edges.*

**Proof.** Computing the connected components of  $(V, F)$  takes  $O(|V| + |F|)$  time. Adding all the edges between supernodes takes  $O(|V|^2)$  time. Then we can contract the supernodes (allowing parallel edges) in  $O(|V|^2)$  time. Removing the edges between hostile supernodes takes  $O(|E| + |H|)$  time.

For the steps in the loop of Line 6, notice that there are at most  $|E|$  edges connecting distinct supernodes, as the only edges we added were internal in supernodes. We can iterate over all these edges  $\{u, v\}$ , and over all supernodes  $W$ . If  $W$  is connected with  $s(u)$  and hostile with  $s(v)$ , then we remove  $\{u, v\}$  and an arbitrary edge connecting  $W$  with  $s(u)$ , and similarly if  $W$  is connected with  $s(v)$  and hostile with  $s(u)$ . This takes  $O(|V||E|)$  time. Each pair of edges removed trivially satisfies the requirements of Line 6. As we do not add edges in this step, it is impossible that when finishing there is still a pair of edges  $\{e_1, e_2\}$  that needed to be removed; when processing  $e_1$ , we would remove  $e_1$  along with some other edge (possibly different from  $e_2$ ).

Rounding the connections between pairs of supernodes is done in  $O(|V|^2)$  time.

The final graph may have at most  $\binom{|V|}{2}$  edges (even if  $|E|$  was much smaller, e.g. in the case where all nodes belong in the same supernode), therefore the time spent by the pivoting algorithm is at most  $T(|V|, \binom{|V|}{2})$ .

The claimed bound follows by both  $|F|$  and  $|H|$  being  $O(|V|^2)$ . ◀

### 3.1 Approximation factor

In this section we prove the approximation factor of Algorithm 1. We assume that there exists at least one clustering satisfying the hard constraints.

We will use the following terminology. Let  $E'$  be the edge-set of the transformed graph  $G'$ ,  $E_{\frac{1}{2}}$  be the edge-set at Line 8 (exactly before the rounding step),  $OPT$  be the edge-set of an optimal clustering for  $E$  satisfying the hard constraints described by  $F$  and  $H$ ,  $OPT'$  be the edge-set of an optimal clustering for the preferences defined by  $E'$ , and  $C$  be the edge-set of the clustering returned by our algorithm. Finally, let  $\alpha$  be the approximation factor of the PIVOTING\_ALGORITHM we use.

We start with a simple observation about supernodes.

► **Lemma 10.** *Given an instance  $(V, E, F, H)$  of Constrained Correlation Clustering, if two nodes  $u_1, u_2$  are in the same supernode, then they must be in the same cluster.*

**Proof.** The proof follows by “in the same cluster” being a transitive property.

More formally,  $u_1, u_2$  are in the same connected component in  $(V, F)$ , as  $s(u_1) = s(u_2)$ . Thus there exists a path from  $u_1$  to  $u_2$ . We claim that all nodes in a path must be in the same cluster. This is trivial if the path is of length 0 ( $u_1 = u_2$ ) or of length 1 ( $\{u_1, u_2\} \in F$ ). Else, the path is  $u_1, w_1, \dots, w_k, u_2$  for some  $k \geq 1$ . We inductively have that all of  $w_1, \dots, w_k, u_2$  must be in the same cluster, and  $u_1$  must be in the same cluster with  $w_1$  because  $\{u_1, w_1\} \in F$ . Therefore all nodes in the path must be in the same cluster with  $w_1$ . ◀

We now show that it is enough to bound the symmetric difference between  $E$  and  $E'$ , or in other words changes in the graph created by our algorithm. The high level idea is that for any fixed clustering, the difference between the cost of the clustering with respect to  $E$  and the cost of the clustering with respect to  $E'$  cannot be larger than  $|E \Delta E'|$ . That is because we can flip edges from  $E$  to reach  $E'$  and then from  $E'$  to reach the edge-set of the desired clustering (triangle inequality). The proof uses manipulations of triangle inequality and the definition of  $C$ .

► **Lemma 11.** *The cost of our clustering is  $|E \Delta C| \leq (\alpha + 1)|E \Delta E'| + \alpha|E \Delta OPT|$ .*

**Proof.** Symmetric difference satisfies the triangle inequality, therefore we have

$$|E \Delta C| \leq |E \Delta E'| + |E' \Delta C|$$

As  $C$  is an  $\alpha$  approximation of the optimal clustering  $OPT'$  for  $E'$ , it is also an  $\alpha$  approximation of any clustering for  $E'$ , including  $OPT$ . Therefore:

$$|E' \Delta C| \leq |E' \Delta E| + \alpha|E' \Delta OPT|$$

Using the triangle inequality again, we get

$$|E \Delta C| \leq |E \Delta E'| + \alpha|E' \Delta E| + \alpha|E \Delta OPT|$$

which proves our claim. ◀

In order to upper bound  $|E \Delta E'|$  related to the cost of the optimal clustering  $|E \Delta OPT|$ , we first need to lower bound the cost of the optimal clustering.

► **Lemma 12.** *Let  $S$  be the set of all pairs of distinct supernodes  $U, W$  that are in the same cluster in  $OPT$ . Then  $|E \Delta OPT| \geq \sum_{\{U, W\} \in S} |E(U, W) \Delta E_{\frac{1}{2}}(U, W)|$ .*

**Proof.** The high level idea is that when a node is connected to two hostile nodes, then any valid clustering needs to pay for at least one of these edges. Extending this fact to supernodes, we construct an edge-set of size  $\sum_{\{U, W\} \in S} |E(U, W) \Delta E_{\frac{1}{2}}(U, W)|$  such that the optimal clustering needs to pay for each edge in this set.

First, for any  $\{U, w\} \in S$  it holds that  $E(U, W) \Delta E_{\frac{1}{2}}(U, W) = E(U, W) \setminus E_{\frac{1}{2}}(U, W)$  because Line 3 does not modify edges between pairs of distinct supernodes, and Lines 4 and 6 only removes edges. Each edge of  $E(U, W) \setminus E_{\frac{1}{2}}(U, W)$  is the result of applying Line 6, seeing as Line 4 only removes edges from hostile pairs of supernodes.

Thus each edge  $\{u, w\} \in E(U, W) \setminus E_{\frac{1}{2}}(U, W)$  can be paired up with a unique edge  $\{x, y\} \in E$  which is removed together with  $\{u, w\}$ . Without loss of generality it holds that

$x \in U, y \in Z$  for some supernode  $Z$  different from  $U$  and  $W$ . Due to the way Line 6 chooses edges it must be the case that  $Z$  and  $W$  are hostile, hence  $\{x, y\} \in E\Delta OPT$ .

Summing over all pairs of clustered supernodes gives the result stated in the lemma. ◀

We are now ready to bound  $|E\Delta E'|$ .

► **Lemma 13.**  $|E\Delta E'| \leq (1 + \sqrt{5})|E\Delta OPT|$

**Proof.** To prove this, we first charge each pair of nodes in a way such that the total charge is at most  $2|E\Delta OPT|$ . Then we partition the pairs of nodes into 5 different sets, and show that the size of the intersection between  $E\Delta E'$  and each of the 5 sets is at most  $\frac{(1+\sqrt{5})}{2}$  times the total charge given to the pairs in the given set.

The first three sets contain the pairs across non-hostile supernodes; out of them the first one is the most technically challenging, requiring a combination of Lemma 12 (related to Line 6) and a direct analysis on  $E\Delta OPT$ , as neither of them would suffice on its own. The analysis of the second and third sets relate to the rounding in Line 9. The fourth set contains pairs across hostile supernodes, while the fifth set contains pairs within supernodes. Their analysis is directly based on the hard constraints.

Concerning our charging scheme: first, each pair of nodes is charged 1 if the optimal clustering pays for it, i.e. if this pair is in  $E\Delta OPT$ . We also put charge 1 on certain pairs not in  $E\Delta OPT$ , namely on edges  $e \in E$  connecting supernodes that are put together in the optimal clustering, and  $e \in E\Delta E_{\frac{1}{2}}$ . Notice that the number of such edges is a lower bound on  $|E\Delta OPT|$ , by Lemma 12. Therefore the total charge in all pairs of nodes is at most  $2|E\Delta OPT|$  and no pair is charged twice.

Case 1: Consider two distinct supernodes  $U, W$  that are not hostile, have more than  $\frac{3-\sqrt{5}}{2}|U||W|$  edges between them in  $E$ , and have at most  $\frac{3-\sqrt{5}}{2}|U||W|$  edges in  $E_{\frac{1}{2}}$ . Then the rounding of Line 9 removes all edges between them. Therefore  $|E(U, W)\Delta E'(U, W)| = |E(U, W)| \leq |U||W|$ . If  $OPT$  separates  $U$  and  $W$ , then the pairs are charged  $|E(U, W)|$ ; else they are charged  $|U||W| - |E(U, W)|$  due to the part of the charging scheme related to  $E\Delta OPT$ . In the latter case, they are also charged  $|E(U, W)| - |E_{\frac{1}{2}}(U, W)|$  due to the part of the charging scheme related to Lemma 12. Therefore they are charged at least

$$\begin{aligned} |U||W| - |E(U, W)| + |E(U, W)| - |E_{\frac{1}{2}}(U, W)| &= |U||W| - |E_{\frac{1}{2}}(U, W)| \\ &\geq |U||W| - \frac{3-\sqrt{5}}{2}|U||W| \end{aligned}$$

Thus, in the worst case, these pairs contribute  $\max\left\{\frac{|E(U, W)|}{|E(U, W)|}, \frac{|E(U, W)|}{|U||W| - \frac{3-\sqrt{5}}{2}|U||W|}\right\} \leq \frac{1}{1 - \frac{3-\sqrt{5}}{2}} = \frac{(1+\sqrt{5})}{2}$  times more in  $|E\Delta E'|$  compared to their charge.

Case 2: Consider two distinct supernodes  $U, W$  that are not hostile, have more than  $\frac{3-\sqrt{5}}{2}|U||W|$  edges between them in  $E$ , and more than  $\frac{3-\sqrt{5}}{2}|U||W|$  edges in  $E_{\frac{1}{2}}$ . Then the rounding of Line 9 will include all  $|U||W|$  edges between them. Thus we have  $|E(U, W)\Delta E'(U, W)| = |U||W| - |E(U, W)| < (1 - \frac{3-\sqrt{5}}{2})|U||W|$ . If  $OPT$  separates  $U, W$ , then it pays for  $|E(U, W)| > \frac{3-\sqrt{5}}{2}|U||W|$  pairs, else it pays for  $|U||W| - |E(U, W)|$ . Thus, in the worst case, these pairs contribute  $\frac{(1 - \frac{3-\sqrt{5}}{2})}{\frac{3-\sqrt{5}}{2}} = \frac{(1+\sqrt{5})}{2}$  times more in  $|E\Delta E'|$  compared to their charge.

Case 3: If two distinct supernodes  $U, W$  are not hostile and have at most  $\frac{3-\sqrt{5}}{2}|U||W|$  edges between them in  $E$ , then they also have at most that many edges in  $E_{\frac{1}{2}}$  as we only remove edges between such supernodes. Therefore after the rounding step there is no edge

between them in  $E'$ , meaning that  $|E(U, W) \Delta E'(U, W)| = |E(U, W)| \leq \frac{3-\sqrt{5}}{2}|U||W|$ . If  $OPT$  separates  $U, W$  then it pays for  $|E(U, W)|$  pairs related to the connection between  $U, W$ ; else it pays for  $|U||W| - |E(U, W)| \geq (1 - \frac{3-\sqrt{5}}{2})|U||W| > \frac{3-\sqrt{5}}{2}|U||W|$ . Thus, these pairs' contribution in  $|E \Delta E'|$  is at most as much as their charge.

Case 4: Pairs  $\{u, v\}$  with  $s(u) \neq s(v)$  and  $s(u)$  hostile with  $s(v)$  are not present in  $E'$ . That is because by Line 4 no pair of hostile supernodes is connected; then Line 6 only removes edges, and Line 9 does not add any edge between  $s(u)$  and  $s(v)$  as they had  $0 \leq \frac{3-\sqrt{5}}{2}|s(u)||s(v)|$  edges between them. Edge  $\{u, v\}$  is also not present in  $OPT$  as  $s(u)$  and  $s(v)$  are not in the same cluster because they are hostile. These pairs' contribution in  $|E \Delta E'|$  is exactly equal to their charge.

Case 5: Pairs  $\{u, v\}$  with  $s(u) = s(v)$  are present in  $E'$  by Line 3 and the fact that all subsequent steps only modify edges whose endpoints are in different supernodes. Pair  $\{u, v\}$  is also present in  $OPT$ , by Lemma 10. Therefore these pairs' contribution in  $|E \Delta E'|$  is exactly equal to their charge.

In the worst case, the pairs of each of the five sets contribute at most  $\frac{(1+\sqrt{5})}{2}$  times more in  $|E \Delta E'|$  compared to their charge, which proves our lemma.  $\blacktriangleleft$

### 3.2 Correctness

Having proven the approximation factor of Algorithm 1, in this section we prove its correctness. That is, we prove that either the final algorithm satisfies all hard constraints, or no clustering can satisfy the hard constraints and the algorithm outputs “Impossible”.

We start with showing that our algorithm correctly detects all cases where the hard constraints are impossible to satisfy.

► **Lemma 14.** *Given an instance  $(V, E, F, H)$  of Constrained Correlation Clustering, the graph  $G' \leftarrow \text{TRANSFORM}(V, E, F, H)$  is equal to  $(\emptyset, \emptyset)$  if and only if the Constrained Correlation Clustering instance is impossible to satisfy.*

**Proof.** We show that if two hostile nodes are in the same supernode, then the instance is not satisfiable and the algorithm correctly determines it; on the other hand, if no such hostile nodes exist, then there exists at least one valid clustering.

It holds that  $G' = (\emptyset, \emptyset)$  if there exist nodes  $u_1, u_2$  such that  $u_1, u_2$  are in the same connected component of  $(V, F)$  and  $\{u_1, u_2\} \in H$ . Then  $u_1, u_2$  must be in the same cluster (Lemma 10) and not be in the same cluster (because  $\{u_1, u_2\} \in H$ ). Therefore the instance is impossible to satisfy.

Else no  $u_1, u_2$  in the same supernode are hostile. Creating a cluster for each supernode is a valid clustering. To see this, notice that no hostility constraint is violated, by hypothesis. All friendliness constraints are satisfied because any two nodes that must be linked belong in the same supernode, and thus in the same cluster. Therefore such an instance is satisfiable.  $\blacktriangleleft$

In the following we can thus assume that we have a satisfiable instance with no supernode being hostile to itself.

The next lemma shows that friendly nodes have the same neighborhood and are connected.

► **Lemma 15.** *Given a satisfiable instance  $(V, E, F, H)$  of Constrained Correlation Clustering, let  $G' \leftarrow \text{TRANSFORM}(V, E, F, H)$ . For any  $\{u, v\} \in F$ , it holds that  $u, v$  are connected in  $G'$  and their neighborhoods are the same.*

**Proof.** The idea is that all nodes in the same supernode are explicitly connected by the algorithm, in Line 3. Then all nodes of the same supernode connect to the exact same nodes due to the rounding step in Line 9.



More formally, as  $\{u, v\} \in F$ , they are trivially both in the same connected component of  $(V, F)$ . Thus they are in the same supernode.

As  $s(u) = s(v)$ ,  $u$  and  $v$  get connected in Line 3. All subsequent steps only modify edges  $\{u', v'\}$  where  $s(u') \neq s(v')$ , therefore  $u, v$  remain connected in  $G'$ . Similarly both  $u$  and  $v$  are connected with all other nodes in  $s(u)$ .

For nodes  $w \notin s(u)$ , when  $\{s(u), s(w)\}$  is processed in the loop of Line 9, either both  $u$  and  $v$  get connected to  $w$  or both get disconnected by  $w$ . ◀

Similarly, hostile nodes are disconnected and do not share any common neighbor.

► **Lemma 16.** *Given a satisfiable instance  $(V, E, F, H)$  of Constrained Correlation Clustering, let  $G' \leftarrow \text{TRANSFORM}(V, E, F, H)$ . For any  $\{u, v\} \in H$  it holds that  $u, v$  are not connected in  $G'$  and they have no common neighbor.*

**Proof.** The idea is that all nodes in hostile supernodes are explicitly disconnected by the algorithm, in Line 4. Then if two hostile nodes share a common neighbor, we drop both edges in Line 6.

More formally, as the instance is satisfiable, we have that  $s(u) \neq s(v)$  by Lemma 14. Therefore no node in  $s(u)$  is connected with a node in  $s(v)$  after Line 4. In Line 6 we only remove edges, meaning that when we process  $\{s(u), s(v)\}$  in the loop of Line 9, the two supernodes are not connected, and they stay like that. Thus  $u, v$  (and even  $s(u), s(v)$ ) are not connected in  $G'$ .

After Line 6, for any supernode  $W$  we have that at least one from  $s(u), s(v)$  are not connected with  $W$ , or else the loop would not terminate. Assume without loss of generality that  $s(u)$  is not connected with  $W$ . Therefore  $s(u)$  is also not connected with  $W$  after the loop of Line 9, meaning that even if  $v$  is connected with a node  $w \in W$ ,  $u$  is not as  $s(u)$  is not connected with  $s(w) = W$ . This guarantees that they have no common neighbor. ◀

With these lemmas, we can conclude that a pivoting algorithm on  $G'$  gives a clustering that satisfies the hard constraints. This was already observed in [19]; we include a short proof for intuition, as we also use this lemma in Section 5.

► **Lemma 17.** *Let  $(V, E, F, H)$  be a satisfiable instance of Constrained Correlation Clustering and  $G' = (V, E')$  be a graph such that any two friendly nodes are connected and have the same neighborhood in  $G'$ , while hostile nodes are not connected and have no common neighbor in  $G'$ . Then applying a pivoting algorithm on  $G'$  gives a clustering that satisfies the hard constraints. In particular, this holds for  $G' = \text{TRANSFORM}(V, E, F, H)$ .*

**Proof.** The high level idea is that due to the assumptions, the choice of the first pivot does not violate any hard constraint. As pivoting algorithms progress, they work with induced subgraphs of the original graph, which also satisfy the assumptions, and therefore no hard constraint is ever violated.

For the sake of contradiction, assume that two hostile nodes  $u, v$  are placed in the same cluster by a pivoting algorithm. By definition of a pivoting algorithm, this happens when we work with some  $V' \subseteq V$  on the induced subgraph  $G'[V']$ , and we pivot on a node  $w$  that is connected with both  $u, v$ . As  $w$  is connected with both  $u, v$  in  $G'[V']$ , it is also connected with  $u, v$  in  $G'$ . But this contradicts the assumption on hostile nodes.

Similarly, for the sake of contradiction assume that two friendly nodes  $u, v$  are put in separate clusters by a pivoting algorithm. Without loss of generality assume that  $u$  is the first to be placed in a cluster that does not contain  $v$ . Again, this happens when we work with some  $V' \subseteq V$  on the induced subgraph  $G'[V']$ , and we pivot on a node  $w$  that is connected



with  $u$  but not with  $v$ . As  $G'[V']$  is an induced subgraph,  $w$  is connected with  $u$  but not with  $v$  in  $G'$ . But this contradicts the assumption on friendly nodes.

Therefore, by Lemmas 15 and 16 the claim holds for  $G' = \text{TRANSFORM}(V, E, F, H)$ . ◀

To prove Theorem 2, notice that there exists a randomized combinatorial pivoting algorithm which gives a 3 approximation and runs in  $O(|V| + |E|) = O(|V|^2)$  time. Using this algorithm in our Algorithm 1 gives a valid clustering. By Lemmas 11 and 13, its approximation is  $4(1 + \sqrt{5}) + 3 < 14$ . The running time follows from Lemma 9 and its correctness by Lemma 17.

Similarly, Theorem 3 is proved by using the deterministic combinatorial pivoting algorithms of Section 4.

## 4 Deterministic Combinatorial Pivoting for Correlation Clustering

### 4.1 Lower Bound

First, we prove Theorem 4, that is no pivoting algorithm for Correlation Clustering has approximation factor  $3 - \Omega(1)$ .

Let  $G = ([n], E)$  be a graph where  $n$  is an even integer, and the edge-set contains all pairs of nodes except for pairs of the form  $(2k + 1, 2k + 2)$  for integer  $k$ . In other words, the edge-set of  $G$  contains all edges except for a perfect matching.

Notice that if we create a single cluster containing all nodes, then the cost is  $\frac{n}{2}$ . On the other hand, let  $u$  be the first choice that a pivoting algorithm makes. If  $u$  is even, let  $v = u - 1$ , else let  $v = u + 1$ . By definition of  $G$ ,  $v$  is the only node not adjacent to  $u$ . Then the algorithm creates two clusters, one containing all nodes except for  $v$ , and one containing only  $v$ . There are  $n - 2$  edges across the two clusters, and  $\frac{n}{2} - 1$  missing edges in the big cluster, meaning that the cost is  $\frac{3n}{2} - 3$ .

Therefore, the approximation factor of any pivoting algorithm is at least  $\frac{\frac{3n}{2} - 3}{\frac{n}{2}} = 3 - \frac{6}{n}$ . This proves Theorem 4, as for any constant less than 3, there exists a large enough  $n$  such that  $3 - \frac{6}{n}$  is larger than the constant.

### 4.2 Upper Bound

In this section we design a 5.8 approximation pivoting algorithm for (unconstrained) Correlation Clustering. To the best of our knowledge this is the first pivoting algorithm for Correlation Clustering that is both deterministic and combinatorial<sup>4</sup>.

We start with a simple 9 approximation algorithm in order to demonstrate the framework we use, developed in [19]. Then we provide a 6 approximation matching the best known deterministic combinatorial approximation from [20]; this solution demonstrates our new ideas. Finally, using a more complicated version of our ideas we provide the 5.8 approximation.

<sup>4</sup> Independently from our work, a deterministic combinatorial 6 approximation algorithm was presented in [20]. However it is not a pivoting algorithm as it first modifies the input graph and then applies pivoting, meaning it does not satisfy the conditions of Lemma 17.

### 4.2.1 Framework from [19] and 9 approximation

In [19] the authors provide a general framework for designing pivoting algorithms with good approximation guarantees. A high level description of their framework is<sup>5</sup>:

- Determine a lower bound  $L$  for the cost of the optimal clustering.
- Assign each pair of nodes  $\{u, v\}$  a budget  $b_{u,v}$  so that the sum of all budgets is  $L$ .
- Let  $A$  be a number such that for all triplets  $\{u, v, w\}$  satisfying  $\{u, v\} \in E, \{u, w\} \in E$  but  $\{v, w\} \notin E$ , it holds that  $A \geq \frac{3}{b_{u,v} + b_{u,w} + b_{v,w}}$ .

The above steps can then directly be used to give an  $A$  approximation pivoting algorithm.

Before providing their result more formally, let us give some intuition on the last point of the framework. Such a triplet  $\{u, v, w\}$  is known as a bad triplet because any clustering needs to pay for at least one of the three pairs among these nodes. To see this, assume w.l.o.g. that  $u$  is connected with both  $v, w$ . Then not paying for any pair would mean that  $u$  is in the same cluster with  $v$ , and  $u$  is in the same cluster with  $w$ . But then the clustering would pay for having  $v$  and  $w$  in the same cluster.

Furthermore, a pivoting algorithm pays for two types of costs: either because when pivoting on some  $u$  there is a missing  $\{v, w\}$  edge on the new cluster, or there is an edge  $\{v, w\}$  but  $v$  is in the new cluster while  $w$  is not. In both cases  $\{u, v, w\}$  is a bad triplet. The third point of the framework is basically used to show that, on average, when the algorithm pays for a pair of nodes, there is a large budget associated with it.

The result from [19], using the terminology of our paper, is:

► **Lemma 18** (Theorem 3.1 and Lemma 3.1 of [19]). *Given an input  $(V, E)$  of Correlation Clustering with optimal cost  $OPT$  and a set of budgets  $\{b_{u,v} \mid \{u, v\} \in \binom{V}{2}\}$ , let  $A$  be a number such that for all bad triplets  $\{u, v, w\}$  in  $(V, E)$  it holds that  $A \geq \frac{3}{b_{u,v} + b_{u,w} + b_{v,w}}$ . If  $\sum_{\{u,v\} \in \binom{V}{2}} b_{u,v} \leq OPT$ , then there exists a pivoting algorithm explicitly using the set of budgets, whose approximation factor is  $A$  and its running time is  $O(|V|^3)$ .*

Our pivoting algorithm is based on this framework. The way budgets were assigned in [19] was by solving an LP. In our case we compute a maximal set  $T$  of pair-disjoint bad triplets, and assign budget  $\frac{1}{3}$  to each pair of each triplet. The maximality ensures that any bad triplet not in  $T$  shares a pair with at least one triplet in  $T$ . The pair-disjointness ensures that  $|T| \leq OPT$ .

► **Lemma 19.** *It holds that  $|T| \leq OPT$ .*

**Proof.** Given a bad triplet  $t = \{u, v, w\}$ , any clustering needs to pay at least 1 for the three pairs of nodes in  $t$ . As all  $t \in T$  are pair-disjoint, any clustering needs to pay  $|T|$  for the  $3|T|$  pairs of nodes  $\{u', v'\}$  for which there exists a bad triplet  $\{u', v', w'\} \in T$ . ◀

► **Lemma 20.** *A maximal set  $T$  of pair-disjoint bad triplets can be computed in  $O(|V|^3)$  time.*

**Proof.** Set  $T = \emptyset$  and for each pair of nodes store a bit (initially zero) describing whether a bad triplet in  $T$  contains this pair. Iterate over all  $O(|V|^3)$  triplets  $\{u, v, w\}$  in the graph, and check in constant time whether  $\{u, v, w\}$  is a bad triplet. If it is, check in constant

<sup>5</sup> Their framework is in fact more general, allowing weights in the edges (satisfying certain constraints) and allowing the pivoting to be done on a graph different than the natural one from the input. Here we restrict it to our case, where their  $(w_{u,v}^+, w_{u,v}^-)$  are simply  $(1, 0)$  if  $\{u, v\} \in E$  and  $(0, 1)$  otherwise, while  $E^+ = E, E^- = \binom{V}{2} - E$ .

time if any of  $\{u, v\}$ ,  $\{v, w\}$ ,  $\{u, w\}$  is in  $T$ . If none is, then add  $\{u, v, w\}$  in  $T$  and set the bit of  $\{u, v\}$ ,  $\{v, w\}$ ,  $\{u, w\}$  to 1. Trivially, the bad triplets in  $T$  are pair-disjoint and  $T$  is maximal.  $\blacktriangleleft$

By Lemmas 18 and 20, the running time of our pivoting algorithm is  $O(|V|^3)$ . We now prove that it is a 9 approximation.

► **Lemma 21.** *Let  $\{u, v, w\}$  be a bad triplet. Then  $\frac{3}{b_{u,v}+b_{v,w}+b_{u,w}} \leq 9$ .*

**Proof.** As  $T$  is a maximal pair-disjoint set of bad triplets, there exists a bad triplet  $\{u', v', w'\}$  sharing a pair with  $\{u, v, w\}$ . By our budgeting scheme the budget of this pair is  $\frac{1}{3}$ , therefore  $\frac{3}{b_{u,v}+b_{v,w}+b_{u,w}} \leq 9$ .  $\blacktriangleleft$

Using Lemmas 18, 19 and 21 concludes the proof of the second part of Theorem 1.

## 4.2.2 New ideas - 6 approximation

To the best of our knowledge, existing combinatorial algorithms for Correlation Clustering lower bound the optimal value by arguing about bad triplets. Especially for deterministic algorithms, the state-of-the-art solution in [20] uses a maximal set of pair-disjoint bad triplets, as we did with our 9 approximation. However, there are certain cases where such an approach gives weak lower bounds.

For example, both in the case of the claw graph  $(K_{1,3})$ , and in the case of the complete bipartite graph  $K_{2,2}$ , a maximal set of pair-disjoint bad triplets contains only one bad triplet. Therefore, the lower-bound for these graphs would be 1. However, any clustering for these graphs incurs a cost of at least 2.

In our 6 approximation algorithm, we rely on lower-bounding OPT related to the existence of pair-disjoint bad triplets  $K_{1,2}$ , claw graphs  $K_{1,3}$ , and complete bipartite graphs  $K_{2,2}$ .

► **Lemma 22.** *Any clustering of  $K_{1,3}$  incurs a cost of at least 2. Similarly for  $K_{2,2}$ .*

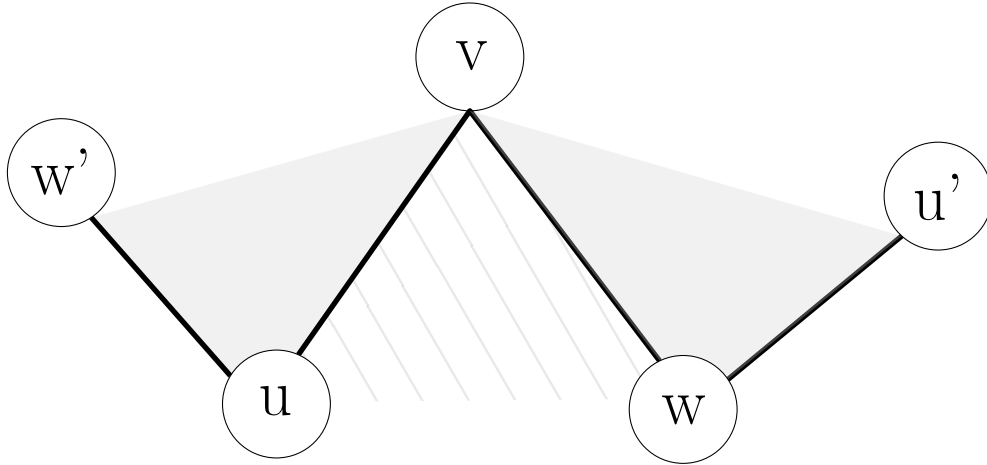
**Proof.**  $K_{1,3}$ : If the cluster  $C$  containing the central node  $u_1$  contains all nodes, we pay for 3 missing edges in  $C$ . If it contains 3 nodes, then we pay for 1 missing edge in  $C$  plus one outgoing edge from  $u_1$ . Else we pay for at least 2 outgoing edges from  $u_1$ .

$K_{1,3}$ : Let  $C$  be any cluster. If it contains all nodes, we pay for 2 missing edges. In all other cases we pay for at least 2 outgoing edges.  $\blacktriangleleft$

Before providing our algorithm, we lay out some terminology. Let  $T$  be a maximal set of pair-disjoint bad triplets. By definition of  $T$ , any bad triplet shares a pair of nodes with some bad triplet in  $T$ . We say that a bad triplet is *poor* if it contains exactly one pair of nodes shared with some triplet in  $T$ . We say that a bad triplet  $\{u, v, w\} \in T$  is  $\{u, v\}$  *associated* (or just *associated*) with a bad triplet  $\{u, v, w'\}$  if  $\{u, v, w'\}$  is poor. There are two reasons we are interested in such cases:

- If  $\{u, v, w\} \in T$  is both  $\{u, v\}$  associated with  $\{u, v, w'\}$  and  $\{v, w\}$  associated with  $\{u', v, w\}$ , and  $u' \neq w'$ , then we can extend  $T$  by removing  $\{u, v, w\}$  and inserting  $\{u, v, w'\}$  and  $\{u', v, w\}$ . See Figure 2.
- The reason our 9 approximation is not better is exactly poor bad triplets.

The intuition behind our new algorithm is the following: poor bad triplets were only charged  $\frac{1}{3}$  in the 9 approximation, because the charging of bad triplets in  $T$  was symmetric. We now apply an asymmetric charging, giving 0.5 charge to 1 pair of nodes, and 0.25 to the other two.



■ **Figure 2** The only bad triplet in  $T$  is  $t = \{u, v, w\}$  (diagonal pattern). It is associated with  $t_1 = \{u, v, w'\}$  and  $t_2 = \{u', v, w\}$  (gray-filled). We can extend  $T$  by removing  $t$  and inserting  $t_1, t_2$ .

The problem that occurs is that a bad triplet in  $T$  may also be associated with a bad triplet on a different pair of nodes, and thus only give charge 0.25 to it. However, in this case we use the idea presented in Figure 2 to extend  $T$  and obtain a stronger lower bound for  $OPT$ ; if extending is not possible, this is because we have a very special case, a subgraph  $K_{1,3}$  or  $K_{2,2}$ . Even though we cannot extend  $T$ , we again obtain stronger lower bounds by arguing directly about  $K_{1,3}$  and  $K_{2,2}$ , instead of through the use of bad triplets.

Finally, stronger lower bounds imply more available budget to distribute, which allows us to charge “problematic” bad triplets more.

The pseudocode of our 6 approximation is in Algorithm 2. It assigns budgets to pairs of nodes and then applies the algorithm from Lemma 18. We use the terms *charge* and *budget* interchangeably.

► **Lemma 23.** *In Algorithm 2,  $T$  is a maximal set of pair-disjoint bad triplets from Line 6 and beyond.*

**Proof.** By definition,  $T$  is initially a maximal set of pair-disjoint bad triplets. Then, when we remove a bad triplet  $t = \{u, v, w\}$  and insert bad triplets  $t_1 = \{u, v, w'\}$ ,  $t_2 = \{u', v, w\}$ , no pair from  $\{u, w'\}$ ,  $\{v, w'\}$ ,  $\{u', v\}$ ,  $\{u', w\}$  has both nodes simultaneously in the same bad triplet of  $T$ , as  $t_1, t_2$  are poor. Moreover, pairs  $\{u, v\}$  and  $\{v, w\}$  are not associated with any bad triplet in  $T$ , as they were only associated with  $t$  and we remove  $t$  from  $T$ . Finally, all 6 pairs are distinct, as  $u' \neq w'$  and  $u', w' \notin \{u, v, w\}$  ( $t_1, t_2$  are disjoint from  $t$  as they are poor). Therefore  $T$  is a set of pair-disjoint bad triplets.

Concerning its maximality, notice that  $\{u, w\}$  is the only pair of nodes that was contained in a bad triplet in  $T$  but is not contained after such a modification. Therefore if  $T$  stops being maximal, it is because of a bad triplet including  $u, w$ . Our algorithm includes such a bad triplet in  $T$ , if one exists, and thus ensures the maximality of  $T$ . ◀

► **Lemma 24.** *The running time of Algorithm 2 is  $O(|V|^5)$ .*

**Proof.** There are trivially  $O(|V|^3)$  bad triplets. Furthermore, there are  $O(|V|^2)$  bad triplets in  $T$  as they are pair-disjoint (Lemma 23 and  $\binom{|V|}{2}$  pairs in total). Computing  $T$  takes  $O(|V|^3)$  time by Lemma 20. Extending  $T$  can happen  $O(|V|^2)$  times as we always increase its size by at least 1. Each time we check over all bad triplets in  $T$  and iterate over all

■ **Algorithm 2** Deterministic combinatorial 6 approximation for Correlation Clustering, based on pivoting.

---

```

1 Compute a maximal set of pair-disjoint bad triplets  $T$ 
2 while  $\exists\{u, v, w\} \in T$  that is associated both with  $\{u, v, w'\}$  and with  $\{u', v, w\}$ , and
    $u' \neq w'$  do
3   Remove  $\{u, v, w\}$  from  $T$  and insert  $\{u, v, w'\}$  and  $\{u', v, w\}$ 
4   if a bad triplet  $\{u, v', w\}$  shares no pair with bad triplets in  $T$  then
5     Insert  $\{u, v', w\}$  in  $T$ 
6 while  $\exists\{u, v, w\} \in T$ , a node  $x$  such that  $G[\{u, v, w, x\}]$  is isomorphic to either  $K_{1,3}$ 
   or  $K_{2,2}$  with no pair of nodes charged, and all bad triplets in  $G[\{u, v, w, x\}]$ 
   containing  $x$  are poor do
7   Give  $\frac{1}{3}$  budget to all 6 pairs of nodes
8 Let  $T'$  be the set of bad triplets containing no pair of nodes with assigned budget
9 foreach  $t = \{u, v, w\} \in T$  that has no pair of nodes with assigned budget do
10  if  $t$  is not associated with any bad triplet in  $T'$  then
11    Give  $\frac{1}{3}$  budget to each one of its 3 pairs of nodes
12  else
13    Pick an arbitrary associated  $t' \in T'$  (w.l.o.g. assume they share nodes  $u, v$ )
14    Give 0.5 budget to  $\{u, v\}$  and 0.25 to  $\{u, w\}$  and  $\{v, w\}$ 
15  Update  $T'$ 
16 Run the pivoting algorithm from Lemma 18 using the assigned budgets

```

---

other nodes. Therefore it takes  $O(|V|^5)$  time. Detecting uncharged subgraphs  $K_{1,3}$  and  $K_{2,2}$  containing a bad triplet in  $T$  again happens at most  $O(|V|^2)$  times and each time we need  $O(|V|^3)$  time. Computing  $T'$  takes  $O(|V|^3)$  time. Finally, for each bad triplet in  $T$  we spend  $O(|V|)$  time to find associated uncharged triplets and update  $T'$ . The algorithm from 18 takes  $O(|V|^3)$  time. ◀

We now prove that the total budget in all pairs of nodes is a lower bound for the cost of the optimal clustering.

► **Lemma 25.** *The total budget from Algorithm 2 in all pairs of nodes is a lower bound for the cost of the optimal clustering.*

**Proof.** During the algorithm, we either give total budget 2 to all pairs of a  $K_{1,3}$  subgraph, or total budget 2 to all pairs of a  $K_{2,2}$  subgraph, or total budget 1 to all pairs of a bad triplet. In all cases, this is a lower bound on the cost of the optimal clustering for these pairs (folklore for bad triplets, Lemma 22 for  $K_{1,3}$  and  $K_{2,2}$ ). As we only assign budgets if all pairs of such a subgraph have not been assigned budget, no pair of nodes is charged more than once. This proves our claim. ◀

It remains to show that our algorithm is a 6 approximation.

► **Lemma 26.** *Algorithm 2 for Correlation Clustering is a 6 approximation.*

**Proof.** By Lemmas 18 and 25, it suffices to show that every bad triplet is charged at least 0.5. Notice that every time we charge all pairs of a bad triplet in  $T$ , the total sum of charges related to the bad triplet is 1. Furthermore, we either charge only the 3 pairs related to

the bad triplet, or the 3 pairs related to the bad triplet and 3 pairs none of which is related to a bad triplet in  $T$  (in the case of  $K_{1,3}$  or  $K_{2,2}$ ). As bad triplets in  $T$  are pair-disjoint (Lemma 23), this means that every time we charge some pairs, we charge all pairs of one bad triplet in  $T$  and no pair of any other bad triplet in  $T$ . Therefore after the loop at Line 9, each bad triplet in  $T$  is charged exactly 1.

By maximality of  $T$  (Lemma 23) each bad triplet  $t' = \{u, v, w'\} \notin T$  shares 2 nodes with at least one bad triplet in  $T$ . If  $t'$  is not poor, then it is charged at least 0.5, as at least two of its pairs are charged and no pair of nodes is ever charged less than 0.25. It thus suffices to show that any poor bad triplet  $t' = \{u, v, w'\}$  associated with some  $t = \{u, v, w\} \in T$  is charged at least 0.5.

If we charge  $t$  because  $G[\{u, v, w, w'\}]$  is isomorphic to  $K_{1,3}$  or  $K_{2,2}$ , then  $t'$  is charged 1 ( $\frac{1}{3}$  on each of its 3 pairs). If we charge  $t$  because there exists some  $w'' \neq w'$  such that  $G[\{u, v, w, w''\}]$  is isomorphic to  $K_{1,3}$  or  $K_{2,2}$ , then either  $\{u, w, w''\}$  or  $\{v, w, w''\}$  is a bad triplet (follows by a straightforward case analysis) and is thus poor by Line 6. Also  $t'$  is associated with  $t$  and  $w' \neq w''$ , meaning that we never reach such cases because we would have instead extended  $T$  in previous steps (Line 2). Therefore we can assume that  $t$  was not charged as part of a  $K_{1,3}$  or  $K_{2,2}$ .

If the bad triplet  $t'$  was not in  $T'$  when  $t$  was charged, then another one of its pairs was already charged; again, as the minimum charge we ever give is 0.25 the claim follows. If  $t'$  was in  $T'$  when  $t$  was charged, assume for the sake of contradiction that  $\{u, v\}$  was charged 0.25. Then  $t$  is associated with another  $t'' = \{u', v, w\}$  and  $\{v, w\}$  was charged 0.5. If  $u' \neq w'$ , then  $t'$  and  $t''$  would replace  $t$  and extend  $T$  before starting charging pairs of nodes (Line 2). Thus,  $u' = w'$ . Notice that this means that  $G[\{u, u', v, w\}]$  is either  $K_{1,3}$  or  $K_{2,2}$ , all 6 pairs of nodes are uncharged, and  $t', t''$  are poor. Even if  $\{u, u', w\}$  is a bad triplet, it is poor because  $\{u, u'\}$  is not contained in any bad triangle in  $T$  (as it is part of  $t'$  which is poor and sharing  $u, v$  with  $t$ ) and similarly  $\{u', w\}$  is not contained in any bad triangle in  $T$  (as it is part of  $t''$  which is poor and sharing  $v, w$  with  $t$ ). Therefore all pairs would be charged  $\frac{1}{3}$  in Line 6, making the charge of  $t'$  equal to 1.  $\blacktriangleleft$

### 4.2.3 5.8 approximation

We take a step back to understand the ideas behind improving the 9 approximation to a 6 approximation. The problem with the 9 approximation was bad triplets  $t'$  sharing only one pair of nodes with some bad triplet  $t \in T$ .

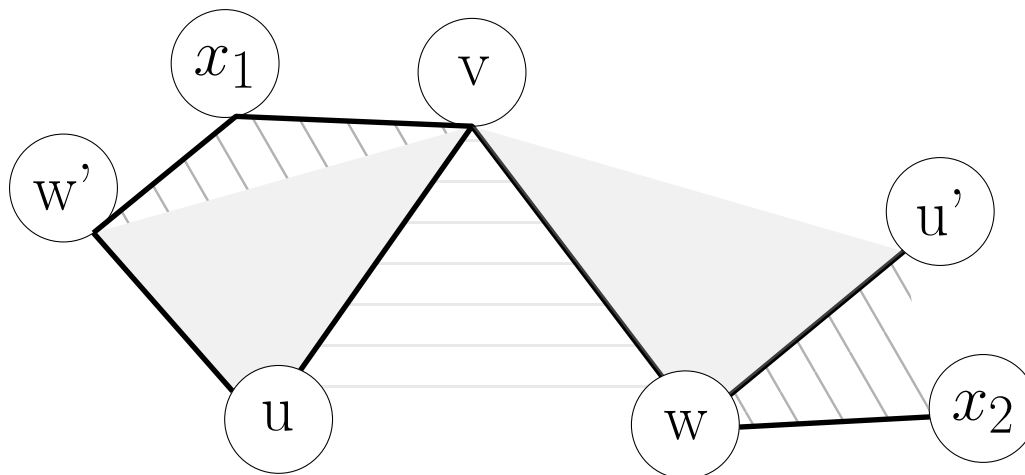
1. Conceptually, the first tool used was modifying the charging so that the problematic pair of nodes shared between  $t$  and  $t'$  is charged more. However, sometimes this was not possible because  $t$  was also associated with some other  $t''$ , sharing a different pair of nodes.
2. The second tool was to use the new problem to our advantage, by finding a related rule extending  $T$ , effectively improving our lower bound. In our case, under some assumptions we could replace the bad triplet  $t$  in  $T$  with the two problematic ones  $t', t''$ . However, sometimes this could not work.
3. The third tool used was to argue that when the second tool fails, it is because of the existence of a subgraph which incurs a large cost, even if it does not have enough pair-disjoint bad triplets in it. In our case this was  $K_{1,3}$  and  $K_{2,2}$ .

We now use this intuition to give a better-than-6 approximation. There are two problems with the previous algorithm, poor bad triplets (where the charge was 0.5 for one pair and 0

for the others), and bad triplets sharing a pair of nodes with two bad triplets  $T$  (where the charge was 0.25 for two pairs and 0 for the other).

In order to eliminate one problem, we use our first tool: modifying the charging to  $0.5 + \delta$  and  $0.25 - \frac{\delta}{2}$  for some  $\delta > 0$ , when the previous algorithm was charging 0.5 and 0.25. Therefore we are now only left to worry about the bad triplets sharing two pairs of nodes with bad triplets in  $T$ , each of which is charged  $0.25 - \frac{\delta}{2}$ .

We now use our second tool: if both these pairs are charged  $0.25 - \frac{\delta}{2}$ , it is because their corresponding bad triplets in  $T$  are associated with some other bad triplet in a different pair of nodes. In Figure 3 we show how to use this to our advantage and extend  $T$ .



■ **Figure 3** Bad triplet  $t_0 = \{u, v, w\}$  (horizontal pattern) shares a pair with two bad triplets in  $T$ , namely  $t = \{u, v, w'\}$ ,  $t' = \{u', v, w\}$  (gray-filled). Each of them is associated with a different bad triplet, namely  $t_1 = \{v, w', x_1\}$ ,  $t_2 = \{u', w, x_2\}$  (diagonal pattern). We extend  $T$  by removing  $t, t'$  and inserting  $t_0, t_1, t_2$ .

Finally, there are certain (in fact a lot) cases where extending  $T$  is not possible. Using our third tool, we directly argue about a large lower bound of the optimal clustering for all these cases.

Having given the intuition, we proceed to the actual description of our algorithm, which slightly modifies Algorithm 2. In fact, our new algorithm only adds the conditional in Line 7 and the loop in Line 13. Notice that they both basically refer to the situation of Figure 3.

We follow the same proof strategy as for Algorithm 2. As our new algorithm only adds the conditional in Line 7 and the loop in Line 13, we mainly focus on the effect of them.

► **Lemma 27.** *In Algorithm 3,  $T$  is a maximal set of pair-disjoint bad triplets from Line 11 and beyond.*

**Proof.** By definition,  $T$  is initially a maximal set of pair-disjoint bad triplets. We then either enter the conditional in Line 3, for which we argued in Lemma 23 that it preserves the properties of  $T$ , or the conditional in Line 7. In the latter case,  $t_1$  is associated with  $t$ ,  $t_2$  is associated with  $t'$ , and  $t_0$  does not share any pair of nodes with bad triplets in  $T \setminus \{t, t'\}$ . Thus, we have that  $t_0, t_1, t_2$  do not share any pair of nodes with bad triplets in  $T \setminus \{t, t'\}$ . As we remove  $t, t'$  from  $T$  and insert  $t_0, t_1, t_2$ , the only bad triplets in  $T$  that may not be pair-disjoint are  $t_0, t_1, t_2$ . However on Line 7 we explicitly make sure that these bad triplets are also pair-disjoint, meaning that  $T$  is still a set of pair-disjoint bad triplets after its modification.



■ **Algorithm 3** Deterministic combinatorial 5.8 approximation for Correlation Clustering, based on pivoting.

---

```

1 Compute a maximal set of pair-disjoint bad triplets  $T$ 
2 while any of the following conditionals is true do
3   if  $\exists\{u, v, w\} \in T$  that is associated both with  $\{u, v, w'\}$  and with  $\{u', v, w\}$ , and
    $u' \neq w'$  then
4     Remove  $\{u, v, w\}$  from  $T$  and insert  $\{u, v, w'\}$  and  $\{u', v, w\}$ 
5     if a bad triplet  $\{u, v', w\}$  shares no pair with bad triplets in  $T$  then
6       Insert  $\{u, v', w\}$  in  $T$ 
7   if  $\exists t = \{u, v, w'\} \in T, t' = \{u', v, w\} \in T$ , bad triplet  $t_0 = \{u, v, w\}$  such that no
   bad triplet in  $T$  contains  $u, w$ , bad triplet  $t_1$  associated with  $t$ , bad triplet  $t_2$ 
   associated with  $t'$ , and  $t_0, t_1, t_2$  are pair-disjoint then
8     Remove  $t, t'$  from  $T$  and insert  $t_0, t_1, t_2$ 
9     while a bad triplet  $t''$  shares no pair with bad triplets in  $T$  do
10      Insert  $t''$  in  $T$ 
11 while  $\exists\{u, v, w\} \in T$ , a node  $x$  such that  $G[\{u, v, w, x\}]$  is isomorphic to either  $K_{1,3}$ 
   or  $K_{2,2}$  with no pair of nodes charged, and all bad triplets in  $G[\{u, v, w, x\}]$ 
   containing  $x$  are poor do
12   Give  $\frac{1}{3}$  budget to all 6 pairs of nodes
13 while  $\exists t = \{u, v, w'\} \in T, t' = \{u', v, w\} \in T$ , bad triplet  $t_0 = \{u, v, w\}$  such that no
   bad triplet in  $T$  contains  $u, w$ , bad triplet  $t_1$  associated with  $t$  such that
    $t \cap t_1 \neq \{u, v\}$ , bad triplet  $t_2$  associated with  $t'$  such that  $t' \cap t_2 \neq \{v, w\}$  and no pair
   of nodes contained in any of  $t_0, t_1, t_2, t, t'$  is charged do
14   if  $t_2$  is the only uncharged bad triplet associated with  $t'$  then
15     Charge  $\frac{15}{29}$  to  $t \cap t_1$  and  $\frac{8}{29}$  to other pairs of nodes in any of  $t, t', t_0, t_1, t_2$ 
16   else
17     Charge  $\frac{15}{29}$  to  $t' \cap t_2$  and  $\frac{8}{29}$  to other pairs of nodes in any of  $t, t', t_0, t_1, t_2$ 
18 Let  $T'$  be the set of bad triplets containing no pair of nodes with assigned budget
19 foreach  $t = \{u, v, w\} \in T$  that has no pair of nodes with assigned budget do
20   if  $t$  is not associated with any bad triplet in  $T'$  then
21     Give  $\frac{1}{3}$  budget to each one of its 3 pairs of nodes
22   else
23     Pick an arbitrary associated  $t' \in T'$  (w.l.o.g. assume they share nodes  $u, v$ )
24     Give  $\frac{15}{29}$  budget to  $\{u, v\}$  and  $\frac{7}{29}$  to  $\{u, w\}$  and  $\{v, w\}$ 
25     Update  $T'$ 
26 Run the pivoting algorithm from Lemma 18 using the assigned budgets

```

---

Concerning maximality, we explicitly have a loop adding bad triplets as long as this is possible. ◀

The running time of our algorithm is now  $O(|V|^6)$ .

► **Lemma 28.** *The running time of Algorithm 3 is  $O(|V|^6)$ .*

**Proof.** The second rule for extending  $T$  can happen at most  $O(|V|^2)$  times, as in the proof



of Lemma 24. Each time we iterate over all bad triplets  $t_0$  in  $O(|V|^3)$  time, find the two bad triplets  $t, t'$  in  $T$  with which it shares a pair of nodes in  $O(1)$  time (as each pair of nodes belongs in at most one bad triplet in  $T$ ), and verify that  $\{v, w\}$  is not contained in any bad triplet in  $T$  in  $O(1)$  time. We can find the associated bad triplets  $t_1$  and  $t_2$  in  $O(|V|)$  time as follows:

We find all bad triplets associated with  $t$  and pair-disjoint with  $t_0$  by iterating over all nodes and over the 2 pairs of  $t$  not shared with  $t_0$ . Let  $A$  be the set of these bad triplets. We keep track of how many of these bad triplets contain any pair of nodes. Then we repeat the same process with  $t'$ ; each time we find a new bad triplet  $q$ , it suffices to check if there exists any pair-disjoint bad triplet in  $A$ . Notice that bad triplets cannot share exactly two pairs of nodes, and if they share 3 then they are the same bad triplet. Therefore, when we have a bad triplet  $q$ , we remove it (if it exists) from  $A$ , decrement the counter of the 3 corresponding pairs by 1, and check the sum of the 3 counters of the 3 pairs of nodes in  $q$ . Then we undo the modifications and continue. It easily follows that the sum of the counters is less than  $|A|$  if and only if there exists a bad triplet in  $A$  that is pair-disjoint from  $q$ .

This process clearly runs in  $O(|V|)$  time and finds proper bad triplets  $t_1, t_2$  if and only if a pair of bad triplets with the desired properties exists. In total, the running time for this step is  $O(|V|^6)$ .

The other extra step is the loop of Line 13 when we detect the same type of subgraphs without the pair-disjointness conditions for  $t_0, t_1, t_2$ . We also check the number of associated bad triplets to a given triplet in  $O(|V|)$  time and charge the related pairs in  $O(1)$  time. Thus running time of this loop is  $O(|V|^4)$  as we do not need to repeat  $O(|V|^2)$  times.

The rest of the steps run in  $O(|V|^5)$  time as in the proof of Lemma 24. ◀

As with the cases of  $K_{1,3}$  and  $K_{2,2}$  (Lemma 22), we need to bound the cost of any clustering for the cases appearing in Line 13 of Algorithm 3.

► **Lemma 29.** *Let  $t, t', t_0, t_1, t_2$  be the bad triplets in Line 13 of Algorithm 3. These five bad triplets contain at most 6 nodes, at most 10 distinct pairs of nodes, and any clustering needs to pay at least 3 for the pairs of nodes in them.*

**Proof.** Let  $x_1$  be the only node in  $t_1 \setminus t$  and  $x_2$  the only node in  $t_2 \setminus t'$ . For the sake of contradiction, assume that  $x_1 \neq x_2$  and both  $x_1, x_2$  are not contained in  $\{u, u', v, w, w'\}$ ; then  $t_0, t_1, t_2$  would be pair-disjoint, which is impossible due to the conditional in Line 7. This means that the five bad triplets contain at most 6 nodes in total, and that a pair of nodes in  $t_0, t_1$  or  $t_2$  is shared by at least two of these three bad triplets. Therefore  $t_0, t_1, t_2$  contain at most 8 distinct pairs of nodes. Bad triplet  $t$  shares  $\{u, v\}$  with  $t_0$  and a different pair with  $t_1$ ; similarly  $t'$  shares  $\{v, w\}$  with  $t_0$  and a different pair with  $t_2$ . We conclude that there are at most 10 pairs in total.

Concerning the lower bound on the cost, notice that if a clustering pays for  $\{u, w\}$ , then it also pays for some pair in  $t$  and some pair in  $t'$ ; as  $t, t'$  are pair-disjoint and none includes both  $u$  and  $w$ , any such clustering pays at least 3.

If it does not pay for  $\{u, w\}$ , then it pays for some other pair of  $t_0$ ; w.l.o.g. assume it is  $\{u, v\}$ . It holds that  $t_1$  contains  $w$ , therefore it cannot contain both  $u$  and  $v$ . Furthermore  $t_1$  is associated with  $t$ , meaning that it is pair-disjoint from  $t'$ . As  $t_1, t'$  are pair-disjoint and none contains  $\{u, v\}$ , any such clustering pays at least 3.

We conclude that any clustering pays at least 3 for pairs of nodes in  $t, t', t_0, t_1, t_2$ . ◀

Using Lemma 29 we can now show that the sum of charges lower bounds the cost of the optimal clustering.

► **Lemma 30.** *In Algorithm 3, the total budget in all pairs of nodes is a lower bound for the cost of the optimal clustering.*

**Proof.** During the algorithm, we either give total budget 2 to all pairs of a  $K_{1,3}$  subgraph, or total budget 2 to all pairs of a  $K_{2,2}$  subgraph, or total budget 3 to at most 10 pairs of the five triplets of Line 13 (Lemma 29), or total budget 1 to all pairs of a bad triplet. In all cases, this is a lower bound on the cost of the optimal clustering for these pairs (folklore for bad triplets, Lemma 22 for  $K_{1,3}$  and  $K_{2,2}$ , Lemma 29 for the rest). As we only assign budgets if all pairs of such a subgraph have not been assigned budget, no pair of nodes is charged more than once. This proves our claim. ◀

It remains to show that our algorithm is a 5.8 approximation. As the proof resembles the one from Lemma 26, we make a separate lemma for the main new technical ingredient.

► **Lemma 31.** *Let  $t, t', t_0, t_1, t_2$  be the bad triangles when entering the conditional in Line 13 of Algorithm 3. Let  $t'_1$  be a bad triplet associated with either  $t$  or  $t'$  that is uncharged at this point of the execution. Then  $t'_1$  is charged at least  $\frac{15}{29}$  at the end of the algorithm's execution.*

**Proof.** Assume that  $t'_1$  is associated with  $t$ , as the other case is symmetric. The high-level idea is that for  $t'_1$  to be charged less than  $\frac{15}{29}$ , there exists at least one more  $t'_2$  associated with  $t'$ . Then we can always find a bad triplet  $t''_1 \in \{t_1, t'_1\}$  and a bad triplet in  $t''_2 \in \{t_2, t'_2\}$  such that  $t_0, t''_1, t''_2$  can replace  $t, t'$  and extend  $T$  (Line 7). There is only one corner-case where no such three pair-disjoint bad triplets can be found; but in this case  $t'_1$  anyway has at least 2 pairs of nodes charged  $\frac{8}{29}$ .

Let  $x_1$  be the only node in  $t_1 \setminus t$  and  $x_2$  the only node in  $t_2 \setminus t$ . Furthermore, we assume  $t'_1 \notin \{t, t', t_0, t_1, t_2\}$ , as else it is charged at least  $3 \cdot \frac{8}{29}$ . Also  $t, t_1, t'_1$  all share the same pair, as else either  $T$  would be extended in Line 3, or they would form a  $K_{1,3}$  or a  $K_{2,2}$ ; but then at least one of their pairs would be charged in Line 11, contradicting the fact that all of these bad triplets are uncharged (by the assumptions of Lemma 31 and Line 13). Similarly, we can assume that there exists some  $t'_2$  associated with  $t'$  (else  $t_1 \cap t'_1$  is charged  $\frac{15}{29}$ ) and  $t', t_2, t'_2$  all share the same pair.

We first note that if  $t_1 = \{v, w', x_1\}$ , then  $x_1 \notin \{u, u', w\}$ , because then  $t_1$  would not be poor (as it is associated with  $t$ ). Similarly,  $x_2 \notin \{u, w, w'\}$  if  $t_2 = \{u', v, x_2\}$ .

If  $t'_1 = \{v, w', x'_1\}$ , then we take cases on  $t_2$ . If  $t_2 = \{u', v, x_2\}$  and  $x_2 \neq x'_1$ , then  $t_0, t'_1, t_2$  are pair-disjoint, meaning that  $T$  would be further extended in Line 7. If  $t_2 = \{u', v, x'_1\}$ , then  $t_0, t_1, t_2$  are pair-disjoint. If  $t_2 = \{u', w, x_2\}$  with  $x_2 \neq u$  then  $t_0, t'_1, t_2$  are pair-disjoint. Finally, if  $t_2 = \{u', w, u\}$ , then  $t_0, t'_1, t'_2$  are pair-disjoint.

If  $t'_1 = \{u, w', x'_1\}$ , then at least one of  $t_1, t'_1$  is equal to  $t''_1 = \{u, w', x\}$  with  $x \neq w$  (if both  $t_1, t'_1$  satisfy this, pick one where  $x \neq u'$ ). We again take cases on  $t_2$ . If  $t_2 = \{u', v, x_2\}$ , then  $t_0, t'_1, t_2$  are pair-disjoint. If  $t_2 = \{u', w, x_2\}$ , then at least one of  $t_2, t'_2$  is equal to  $t''_2 = \{u', w, y\}$  with  $y \neq u$  (if both  $t_2, t'_2$  satisfy this, pick one where  $y \neq w'$ ). As  $t''_1, t_0$  are pair-disjoint, and  $t''_2, t_0$  are pair-disjoint, the only case where  $t''_1, t''_2, t_0$  are not pair-disjoint is if  $t''_1, t''_2$  are not pair-disjoint. This only happens if  $t''_1 = \{u, w', u'\}$  and  $t''_2 = \{u', w, w'\}$ . In this case  $\{t_1, t'_1, t_2, t'_2\} = \{\{u, w', w\}, \{u, w', u'\}, \{u', w, u\}, \{u', w, w'\}\}$ , which means that  $t'_1$  is charged for at least 2 pairs of nodes, thus being charged at least  $2 \cdot \frac{8}{29}$  in total. ◀

The proof of correctness is very similar to the one for the 6 approximation (Lemma 26), but we need to incorporate Lemma 31 as well.

► **Lemma 32.** *Algorithm 3 for Correlation Clustering is a 5.8 approximation.*

**Proof.** By Lemmas 18 and 30, it suffices to show that every bad triplet is charged at least  $\frac{15}{29}$ . Notice that as bad triplets in  $T$  are pair-disjoint (Lemma 27) every time we charge some pairs of nodes, we either charge all pairs of a bad triplet in  $T$  (a total charge of at least  $3 \cdot \frac{8}{29}$ ), or none. Thus, after the loop at Line 9, every bad triplet in  $T$  is charged at least  $3 \cdot \frac{8}{29}$ .

By maximality of  $T$  (Lemma 27) each bad triplet  $t_0 \notin T$  shares 2 nodes with at least one bad triplet in  $T$ . If  $t_0$  shares all its pairs with bad triplets in  $T$ , then it is charged at least thrice the minimum charge on a pair of a bad triplet in  $T$ , which is  $3 \cdot \frac{7}{29}$ . Else, if some pair in  $t_0$  that is not shared with any bad triplet in  $T$  is charged, then it is charged at least  $\frac{8}{29}$ , and therefore  $t_0$  is charged at least  $\frac{8}{29} + \frac{7}{29} = \frac{15}{29}$ . Therefore, from now on we can assume that  $t_0$  is only charged on pairs shared with bad triplets in  $T$  and that at least one pair of  $t_0$  is not charged.

If  $t_0$  is not poor, then the only way that its total charge is less than  $\frac{15}{29}$  is if it is charged  $\frac{7}{29}$  on two pairs. This only happens if it shares a pair with a bad triplet  $t \in T$ , another pair with a bad triplet  $t' \in T$ , and each of them is associated with a bad triplet, sharing a different pair than the one they share with  $t_0$  (say  $t$  with  $t_1$  and  $t'$  with  $t_2$ ). By Line 7  $t_0, t_1, t_2$  are not pair-disjoint, as that would extend  $T$ . But then, by Line 13  $t_0$  would be charged at least  $3 \times \frac{8}{29}$ . We can therefore assume that  $t_0 = \{u, v, w_0\}$  is poor, and associated with a  $t = \{u, v, w\} \in T$ .

It cannot be that we charge  $t$  because  $G[\{u, v, w, w_0\}]$  is isomorphic to  $K_{1,3}$  or  $K_{2,2}$  (Line 11), as that would charge some pair of  $t_0$  not shared with a bad triplet in  $T$ . If we charge  $t$  because there exists some  $w_1 \neq w_0$  such that  $G[\{u, v, w, w_1\}]$  is isomorphic to  $K_{1,3}$  or  $K_{2,2}$ , then either  $\{u, w, w_1\}$  or  $\{v, w, w_1\}$  is a bad triplet (follows by a straightforward case analysis) and is thus poor by Line 6. Also  $t_0$  is associated with  $t$  and  $w_0 \neq w_1$ , meaning that we never reach such cases because we would have instead extended  $T$  in previous steps (Line 3). If  $t$  is charged due to Line 13, then by Lemma 31  $t_0$  it is charged at least  $\frac{15}{29}$ . Therefore we can assume that  $t$  was charged in the loop of Line 19.

By our assumptions,  $t_0$  was in  $T'$  when  $t$  was charged. Assume for the sake of contradiction that  $\{u, v\}$  was charged  $\frac{7}{29}$ . Then  $t$  is associated with another  $t_1 = \{u', v, w\}$  and  $\{v, w\}$  was charged  $\frac{15}{29}$ . If  $u' \neq w_0$ , then  $t_0$  and  $t_1$  would replace  $t$  and extend  $T$  before starting charging pairs of nodes (Line 2). Thus,  $u' = w_0$ . Notice that this means that  $G[\{u, u', v, w\}]$  is either  $K_{1,3}$  or  $K_{2,2}$ , all 6 pairs of nodes are uncharged, and  $t_0, t_1$  are poor. Even if  $\{u, u', w\}$  is a bad triplet, it is poor because  $\{u, u'\}$  is not contained in any bad triangle in  $T$  (as it is part of  $t_0$  which is poor and sharing  $u, v$  with  $t$ ) and similarly  $\{u', w\}$  is not contained in any bad triangle in  $T$  (as it is part of  $t_1$  which is poor and sharing  $v, w$  with  $t$ ). Therefore all pairs would be charged  $\frac{1}{3}$  in Line 6, making the charge of  $t'$  equal to 1. ◀

## 5 Node-Weighted Correlation Clustering

Let  $I = (V, E, w)$  be an instance of Node-Weighted Correlation Clustering. We describe an instance  $I' = (V', E', F', H')$  of Constrained Correlation Clustering that we claim is equivalent to  $I$ , in the sense that there is a bijection between valid clusterings of  $I'$  and clusterings of  $I$ , and each such pair has the same cost in its respective problem. We note that the size of  $I'$  may be much larger than the size of  $I$ , which is not a problem because we do not explicitly build  $I'$  in our algorithm.

For each node  $u \in V$ , we create nodes  $u_1, \dots, u_{w(u)} \in V'$  and all pairs among them are both in  $E'$  and in  $F'$ . For each edge  $\{u, v\} \in E$ , we introduce edges  $\{u_i, v_j\}$  in  $E'$ , for all  $i \in [w(u)], j \in [w(v)]$ . Finally  $H' = \emptyset$ . Notice that each supernode of  $I'$  trivially corresponds to a node in  $I$ .

► **Lemma 33.** *For each clustering  $C$  of  $I$ , there exists a valid clustering  $C'$  of  $I'$  with the same cost, and vice-versa.*

**Proof.** Let  $C$  be a clustering of  $I$ . For each cluster  $X = \{u^1, \dots, u^{|X|}\} \in C$ , we create a valid clustering  $C'$  of  $I'$  containing all nodes  $u_j^i$  such that  $i \in [|X|], j \in [w(i)]$ . Conversely, given any valid clustering  $C'$  of  $I'$ , by Lemma 10 we have that it does not split the nodes of any supernode in different clusters. Thus it follows that we can reverse the process and acquire  $C$ , given  $C', I, I'$ .

Concerning the costs, notice that given a node  $u \in V$ , we have that  $u_1, \dots, u_{w(u)} \in V'$  are all in the same supernode and connected by edges. Therefore, for  $C'$ , no cost is paid for any pair  $\{u_i, u_j\}, \{i, j\} \in \binom{[w(u)]}{2}$ .

Given two nodes  $u, v \in V$ , if they are connected and are in the same cluster of  $C$ , or are not connected and are in different clusters, then no cost is paid; similarly all pairs  $\{u_i, v_j\}, i \in [w(u)], j \in [w(v)]$  either have connected endpoints that are in the same cluster, or not connected endpoints that are in different clusters. Thus they do not incur any cost.

On the other hand if  $u, v$  are connected and not put in the same cluster, or are not connected and put in the same cluster, then cost  $w(u) \cdot w(v)$  is paid; similarly all pairs  $\{u_i, v_j\}, i \in [w(u)], j \in [w(v)]$  need to pay 1, incurring a cost  $w(u) \cdot w(v)$ . ◀

► **Corollary 34.** *Both  $I$  and  $I'$  have the same optimal cost.*

It follows that it is enough to approximate  $I'$ . However implementing Algorithm 1 naively would not be efficient, because explicitly building  $I'$  would be expensive ( $I'$  may be much larger than  $I$  as its size depends on the weights of  $I$ ). Instead, we make two observations:

- There is no need to run Procedure TRANSFORM because our instance already has the desired properties.
- Given access to  $I$ , we can efficiently simulate the execution of the random pivoting algorithm from [2] on  $I'$ .

These observations imply an efficient way to approximate  $I$  through approximating  $I'$ . The algorithm simply applies the randomized pivoting algorithm from [2] on  $I$ , but samples with probability proportional to the weights, instead of uniform.

► **Lemma 35.** *Let  $u_1, \dots, u_n$  be  $n$  objects such that each object  $u$  has an associated positive integer weight  $w(u)$ . We can design a data structure that supports sampling an object with probability proportional to its weight and immediately deleting it. Furthermore it supports arbitrary deletions of objects. The running time until all objects are removed is  $O(|V|)$  with probability  $1 - \frac{1}{n^c}$ , for any constant  $c > 0$ .*

We first use this lemma as a black-box to show how our algorithm works. Then we show how to prove it.

► **Lemma 36.** *There exists a randomized combinatorial algorithm with 3 (expected) approximation for Node-Weighted Correlation Clustering. Its running time is  $O(|V| + |E|)$  with probability  $1 - \frac{1}{|V|^c}$ , for any constant  $c > 0$ .*

**Proof.** Let  $I = (V, E, w)$  be an instance of Node-Weighted Correlation Clustering. Let  $I' = (V', E', F', H')$  be the corresponding instance of Constrained Correlation Clustering.

By construction of  $I'$ , if two nodes are in the same supernode (corresponding to a node of  $I$ ), then they are connected and have the same neighborhood. Trivially, as  $H' = \emptyset$ , no two hostile nodes are connected or have a common neighbor. Therefore a pivoting algorithm satisfies the hard constraints, by Lemma 17.

We now make some remarks about running the randomized pivoting algorithm of [2] on  $I'$ . We then show how to simulate this using only  $I$ . If we run the randomized pivoting algorithm on  $(V', E')$ , we would simply pick a random node  $u$  in  $V'$  and create a cluster out of  $u$  and its (remaining) neighborhood, then remove these nodes from the graph and repeat. By Lemma 10 and the fact that no hard constraint would be violated during this process, at any time step the remaining graph would consist of all the nodes of some subset  $\{U^1, \dots, U^k\}$  of the supernodes, and each cluster would only consist of intact supernodes.

Notice that it would thus be enough for the algorithm to only store the remaining supernodes and the connections among them, as they anyway imply the connections between the actual nodes contained in them. Similarly, it would be enough to store the clustering in the form of supernodes included in the clusters. However there is a natural bijection between nodes in  $V$  and supernodes in  $I'$ , as well as a natural bijection between edges in  $E$  and connections of supernodes in  $I'$ . Therefore, assuming we can properly perform the sampling, we can run the pivoting algorithm directly on  $I$ , instead of  $I'$ . By the proof of Lemma 33 the clustering we get in  $I$  is exactly the equivalent of the clustering that would be output by the pivoting algorithm on  $(V', E')$ . Together with Corollary 34, we get a 3 (expected) approximation for the  $I$  instance.

Sampling a node uniformly at random among the remaining ones remaining in  $V'$  is the same as first sampling a supernode  $U$  from  $\{U^1, \dots, U^k\}$  with probability proportional to its weight, and then sampling a node in  $U$  uniformly at random. As all nodes within  $U$  are connected and have the same neighborhood, which one we would pick would not make any difference for the pivoting algorithm as they would all lead to the same cluster (containing this same neighborhood). This neighborhood consists of the nodes in  $U$  and the nodes in supernodes connected with  $U$ .

Therefore it suffices to simply sample a supernode  $U$  from  $\{U^1, \dots, U^k\}$  with probability proportional to its weight. To simulate this by only working with  $I$ , it suffices to employ a sampling technique that allows deletions of nodes and supports sampling with probability proportional to the nodes' weight. We perform the sampling using Lemma 35, and the running bound follows trivially.  $\blacktriangleleft$

We are only left with proving Lemma 35. By [21], Alias Method allows us to preprocess  $n$  objects  $u_1, \dots, u_n$  with associated weights  $w(u_1), \dots, w(u_n)$  in  $O(n)$  time so that we can sample in  $O(1)$  time with probability proportional to the weights.

Our new data structure first partitions the objects in buckets, such that the  $i$ -th bucket  $B_i$  contains objects with weights in  $[2^{i-1}, 2^i)$ . Let  $S$  be the set of buckets. As we are working in the word RAM model of computation with word-size  $\Theta(\log n)$ ,  $|S| = O(\log n)$  buckets. We set both the initial weight  $w_{init}(B)$  and the actual weight  $w(B)$  of a bucket  $B$  equal to the sum of the weights of objects inside  $B$ . We then preprocess in  $O(\log n)$  time the set of  $O(\log n)$  buckets based on their initial weights, using the Alias Method. We also preprocess each bucket  $B$  individually, in  $O(|B|)$  time, using the Alias Method. The whole preprocessing clearly runs in  $O(n)$  time.

In order to delete an object  $u$  on bucket  $B_u$ , we mark it as deleted, remove it from  $B_u$ , and update the actual weight  $w(B_u)$  (but not the initial weight  $w_{init}(B_u)$ ). If  $w(B_u) < \frac{w_{init}(B_u)}{2}$ , we update  $w_{init}(B_u)$  and re-apply the preprocessing of the Alias Method twice, both for bucket  $B$  and for the set of buckets (based on the initial weights).

To sample an object, we sample a bucket  $B$  with probability proportional to its initial weight in  $O(1)$  time using the Alias Method. We accept it with probability  $\frac{w(B)}{w_{init}(B)}$ , and reject it otherwise. Then we sample an object from this bucket with probability proportional to its weight in  $O(1)$  time, and reject it if it is already deleted. If at any point we rejected, we restart

the process. The probability of sampling a deleted object is trivially 0, and the probability of sampling a non-deleted object  $u$  in bucket  $B_u$  is  $\frac{w_{init}(B_u)}{\sum_{B \in S} w_{init}(B)} \cdot \frac{w(B_u)}{w_{init}(B_u)} \cdot \frac{w(u)}{w(B_u)} = \frac{w(u)}{\sum_{B \in S} w_{init}(B)}$ , which is proportional to the weight of the object. Furthermore, as the actual weight of each bucket is always within a factor 2 of the initial weight, we have a probability 0.5 not to reject.

As we can successfully sample at most  $n$  times, by a standard Chernoff bound for any constant  $c > 0$  there exists a constant  $c'$  such that the probability of spending more than  $c'n$  time is less than  $\frac{1}{n^c}$ .

We are left to argue about the total time of rebuilding. Whenever we rebuild the structure of a single bucket, it is because its actual weight dropped in half since the last rebuilding. As the objects in a bucket have weights that are within a factor 2, the bucket's weight dropping in half means that at least  $\frac{1}{3}$  of the objects were removed. Thus the total number of rebuildings of a bucket  $B$  is  $O(\log |B|)$  and the total time needed for these rebuildings is at most  $|B| + \frac{2}{3}|B| + \frac{2^2}{3^2}|B| + \dots$ , which is  $O(|B|)$ . Therefore the total rebuilding of individual buckets is proportional to the total number of objects in all buckets, which is  $n$ . We also rebuild the structure of the set of buckets each time a bucket is rebuilt, meaning we may rebuild it  $O(\log^2 n)$  times. Each rebuilding costs  $O(\log n)$  as there are that many buckets. This concludes the proof of Lemma 35.

## 6 Acknowledgments

We would like to thank Lorenzo Beretta for his valuable suggestions on weighted sampling.



---

**References**

---

- 1 Rakesh Agrawal, Alan Halverson, Krishnaram Kenthapadi, Nina Mishra, and Panayiotis Tsaparas. Generating labels from clicks. In Ricardo Baeza-Yates, Paolo Boldi, Berthier A. Ribeiro-Neto, and Berkant Barla Cambazoglu, editors, *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009, Barcelona, Spain, February 9-11, 2009*, pages 172–181. ACM, 2009. doi:10.1145/1498759.1498824.
- 2 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. Announced in STOC 2005. doi:10.1145/1411509.1411513.
- 3 Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 952–963. IEEE Computer Society, 2009. doi:10.1109/ICDE.2009.43.
- 4 Sepehr Assadi and Chen Wang. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. *CoRR*, abs/2109.14528, 2021. URL: <https://arxiv.org/abs/2109.14528>, arXiv:2109.14528.
- 5 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, 2004. doi:10.1023/B:MACH.0000033116.57574.95.
- 6 Francesco Bonchi, Aristides Gionis, and Antti Ukkonen. Overlapping correlation clustering. *Knowl. Inf. Syst.*, 35(1):1–32, 2013. doi:10.1007/s10115-012-0522-9.
- 7 Mark Bun, Marek Eliás, and Janardhan Kulkarni. Differentially private correlation clustering. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1136–1146. PMLR, 2021. URL: <http://proceedings.mlr.press/v139/bun21a.html>.
- 8 Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. A graph-theoretic approach to webpage segmentation. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 377–386. ACM, 2008. doi:10.1145/1367497.1367549.
- 9 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005. Announced in FOCS 2003. doi:10.1016/j.jcss.2004.10.012.
- 10 Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 219–228. ACM, 2015. doi:10.1145/2746539.2746604.
- 11 Yudong Chen, Sujay Sanghavi, and Huan Xu. Clustering sparse graphs. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2213–2221, 2012. URL: <https://proceedings.neurips.cc/paper/2012/hash/1e6e0a04d20f50967c64dac2d639a577-Abstract.html>.
- 12 Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2069–2078. PMLR, 2021. URL: <http://proceedings.mlr.press/v139/cohen-addad21b.html>.

- 13 Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2-3):172–187, 2006. doi:10.1016/j.tcs.2006.05.008.
- 14 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014. doi:10.1016/j.jcss.2014.04.015.
- 15 Dmitri V. Kalashnikov, Zhaoqi Chen, Sharad Mehrotra, and Rabia Nuray-Turan. Web people search via connection analysis. *IEEE Trans. Knowl. Data Eng.*, 20(11):1550–1565, 2008. doi:10.1109/TKDE.2008.78.
- 16 Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang Dong Yoo. Higher-order correlation clustering for image segmentation. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 1530–1538, 2011. URL: <https://proceedings.neurips.cc/paper/2011/hash/98d6f58ab0dafbb86b083a001561bb34-Abstract.html>.
- 17 Domenico Mandaglio, Andrea Tagarelli, and Francesco Gullo. Correlation clustering with global weight bounds. In Nuria Oliver, Fernando Pérez-Cruz, Stefan Kramer, Jesse Read, and José Antonio Lozano, editors, *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part II*, volume 12976 of *Lecture Notes in Computer Science*, pages 499–515. Springer, 2021. doi:10.1007/978-3-030-86520-7\_31.
- 18 Gregory J. Puleo and Olgica Milenkovic. Correlation clustering with constrained cluster sizes and extended weights bounds. *SIAM J. Optim.*, 25(3):1857–1872, 2015. doi:10.1137/140994198.
- 19 Anke van Zuylen and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Math. Oper. Res.*, 34(3):594–620, 2009. Announced in SODA 2007. doi:10.1287/moor.1090.0385.
- 20 Nate Veldt. Faster deterministic approximation algorithms for correlation clustering and cluster deletion. *CoRR*, abs/2111.10699, 2021. URL: <https://arxiv.org/abs/2111.10699>, arXiv:2111.10699.
- 21 Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. Software Eng.*, 17(9):972–975, 1991. doi:10.1109/32.92917.
- 22 Julian Yarkony, Alexander T. Ihler, and Charless C. Fowlkes. Fast planar correlation clustering for image segmentation. In Andrew W. Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI*, volume 7577 of *Lecture Notes in Computer Science*, pages 568–581. Springer, 2012. doi:10.1007/978-3-642-33783-3\_41.



## Appendix C

# SOSA: Multiple-Source Shortest Paths in Planar Digraphs

# A Simple Algorithm for Multiple-Source Shortest Paths in Planar Digraphs

Debarati Das<sup>\*†</sup>    Evangelos Kipouridis<sup>\*†‡</sup>    Maximilian Probst Gutenberg<sup>§</sup>  
Christian Wulff-Nilsen<sup>\*¶</sup>

## Abstract

Given an  $n$ -vertex planar embedded digraph  $G$  with non-negative edge weights and a face  $f$  of  $G$ , Klein presented a data structure with  $O(n \log n)$  space and preprocessing time which can answer any query  $(u, v)$  for the shortest path distance in  $G$  from  $u$  to  $v$  or from  $v$  to  $u$  in  $O(\log n)$  time, provided  $u$  is on  $f$ . This data structure is a key tool in a number of state-of-the-art algorithms and data structures for planar graphs.

Klein's data structure relies on dynamic trees and the persistence technique as well as a highly non-trivial interaction between primal shortest path trees and their duals. The construction of our data structure follows a completely different and in our opinion very simple divide-and-conquer approach that solely relies on Single-Source Shortest Path computations and contractions in the primal graph. Our space and preprocessing time bound is  $O(n \log |f|)$  and query time is  $O(\log |f|)$  which is an improvement over Klein's data structure when  $f$  has small size.


## 1 Introduction


In the Planar Multiple-Source Shortest Paths (MSSP) problem, an embedded digraph  $G$  is given along with a distinguished face  $f$ , and the goal is to compute a data structure answering distance queries between any vertex pair  $(u, v)$  where either  $u$  or  $v$  belong to  $f$ . Data structures for this problem are measured by the *preprocessing time* required to construct the data structure, the *space* required to store the data structure, and the *query time* required to answer a query.

**Applications.** MSSP data structures are a crucial building block for the All-Pairs Shortest Paths (APSP) problem in planar graphs where the data structure needs to be able to return the (approximate or exact) distance between *any* two vertices in the graph. Such data structures are often called *distance oracles* if the query time is subpolynomial in  $n$ . Broadly, there are three different APSP data structure problems that currently rely on MSSP algorithms:

---

<sup>\*</sup>Department of Computer Science, University of Copenhagen, Basic Algorithms Research Copenhagen (BARC).  
Emails: [debaratix710@gmail.com](mailto:debaratix710@gmail.com), [{kipouridis,koolooz}@di.ku.dk](mailto:{kipouridis,koolooz}@di.ku.dk).

<sup>†</sup>Debarati Das and Evangelos Kipouridis are supported by VILLUM Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), Denmark. 

<sup>‡</sup>Evangelos Kipouridis has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 801199. 

<sup>§</sup>ETH Zurich, Email: [maximilian.probst@inf.ethz.ch](mailto:maximilian.probst@inf.ethz.ch)

<sup>¶</sup>Christian Wulff-Nilsen is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

- Exact Distance Oracles: In a recent series of breakthroughs, [5, 14, 4, 20] showed that it is possible to obtain an APSP data structure that requires only space  $n^{1+o(1)}$  and query time  $n^{o(1)}$  where both [4] and the state-of-the-art result in [20] employs MSSP as a building block in their construction.
- Approximate Distance Oracles: Thorup [25] presented a data structure that returns  $(1 + \epsilon)$ -approximate distance estimates using preprocessing time and space  $\tilde{O}(n\epsilon^{-1})^1$  and query time  $O(\log \log n + \epsilon^{-1})$ . Since, his construction time was sped-up by polylogarithmic factors via improvements to the state-of-the-art MSSP data structure [18].
- Exact (Dynamic) APSP: A classic algorithm by Fakcharoenphol and Rao [11] gives a data structure that uses  $\tilde{O}(n)$  space and preprocessing time, and takes query time  $\tilde{O}(\sqrt{n})$  to answer queries exactly. A variant of this algorithm further gives a data structure that processes edge weight changes to the graph  $G$  in time  $\tilde{O}(n^{2/3})$  while still allowing for query time  $\tilde{O}(n^{2/3})$ . Again, while [11] did not directly employ an MSSP data structure, Klein [18] showed that incorporating MSSP leads to a speed-up in the logarithmic factors.

We point out that various improvements were made during the last years over these seminal results when considering different trade-offs [12, 21] or by improving logarithmic or doubly-logarithmic factors [23, 13, 22], however, the fundamental sub-problem of MSSP is present in almost all of these articles. We emphasize that beyond the run-time improvements achieved by MSSP data structure, an additional benefit is a more modular and re-usable design that makes it simpler to understand and implement APSP algorithms.

Another key application of MSSP is the computation of a dense distance graph. An often employed strategy for a planar graph problem is to decompose the embedded planar input graph  $G$  into smaller graphs using vertex separators in order to either obtain a recursive decomposition or a flat so called  $r$ -division of  $G$ . For each subgraph  $R$  obtained, let  $\partial R$  denote its set of boundary vertices (vertices incident to edges not in  $R$ ). The dense distance graph of  $R$  is the complete graph on  $\partial R$  where each edge  $(u, v)$  is assigned a weight equal to the shortest path distance from  $u$  to  $v$  in  $R$ . Since the recursive or flat decomposition can be done in such a way that  $\partial R$  is on a constant number of faces of  $R$ , MSSP can then be applied to each such face to efficiently obtain the dense distance graph of  $R$ . There are numerous applications of dense distance graphs, not only for shortest path problems but also for problems related to cuts and flows [1, 2, 17].

**Previous Work.** The MSSP problem was first considered implicitly by Fakcharoenphol and Rao [11] who gave a data structure that requires  $\tilde{O}(n)$  preprocessing time and space and query time  $\tilde{O}(\sqrt{n})$ . Since, the problem has been systematically studied by Klein [18] who obtained a data structure with preprocessing time and space  $O(n \log n)$  and query time  $O(\log n)$ . Klein’s seminal result was later proven to be tight in  $n$  for preprocessing time and space [8]. Klein and Eisenstat [8] also demonstrated that one can remove all logarithmic factors in the special case of undirected, unit-weighted graphs<sup>2</sup>. Finally, Cabello, Chambers and Erickson [3] gave an algorithm exploiting the same structural claims as in [18] but give a new perspective by recasting the problem as a parametric shortest paths problem. This allowed them to generalize the result in [18] to surface-embedded graphs of genus  $g$ , with preprocessing time/space  $\tilde{O}(gn \log n)$  and query time  $O(\log n)$ .

<sup>1</sup>We use  $\tilde{O}(\cdot)$ -notation to suppress logarithmic factors in  $n$ . To state the bounds in a clean fashion, we assume that the ratio of smallest to largest edge weight in the graph is polynomially bounded in  $n$ .

<sup>2</sup>[8] also shows how to use this data structure to obtain a linear-time algorithm for the Max Flow problem in planar, unit-weighted digraphs

**The Seminal Result by Klein [18].** On a high-level, the result by Klein [18] is obtained by the observation that moving along a face  $f$  with vertices  $v_1, v_2, \dots, v_k$ , from vertex  $v_i$  to  $v_{i+1}$ , the difference between the shortest path trees  $T_{v_i}$  and  $T_{v_{i+1}}$  consists on average of  $O(n/k)$  edges. [18] therefore suggests to dynamically maintain a tree  $T$ , initially equal to the shortest path tree  $T_{v_1}$  of  $v_1$ , and then to make the necessary changes to  $T$  to obtain the shortest path tree  $T_{v_2}$  of  $v_2$ , and so on for  $v_3, \dots, v_k$ . Overall, this requires only  $O(n)$  changes to the tree  $T$  over the entire course of the algorithm while passing through all shortest path trees.

To implement changes to  $T$  efficiently, Klein uses a dynamic tree data structure to represent  $T$ , and uses duality of planar graphs in the concrete form of an interdigitizing tree/ tree co-tree decomposition, with the dual tree also maintained dynamically as a top tree. Finally, he uses an advanced persistence technique [7] to allow access to the shortest path trees  $T_{v_i}$  for any  $i$  efficiently.

Even though formalizing each of these components requires great care, the algorithm by Klein is commonly taught in courses and books on algorithms for planar graphs (see for example [19, 6, 9]), but with dynamic trees and persistence abstracted to black box components.

**Our Contribution.** We give a new approach for the MSSP problem that we believe to be significantly simpler and that matches (and even slightly improves) the time and space bounds of [18]. Our algorithm only uses the primal graph, and consist of an elegant interweaving of Single-Source Shortest Paths (SSSP) computations and contractions in the primal graph.

Our contribution achieved via two variations of our MSSP algorithm comprises of:

- A Simple, Efficient Data Structure: We give a MSSP data structure with preprocessing time/space  $O(n \log |f|)$  and query time  $O(\log |f|)$  which slightly improves the state-of-the-art result by Klein [18] for  $|f|$  subpolynomial and otherwise recovers his bounds.

Our result is achieved by implementing SSSP computations via the linear-time algorithm for planar digraphs by Henzinger et al. [16]. Abstracting the algorithm [16] in black-box fashion, our data structure is significantly simpler than [18] and we believe that it can be taught at an advanced undergraduate level.

Further, by replacing the black-box from [16] with a standard implementation of Dijkstra’s algorithm, our algorithm can easily be taught without any black-box abstractions, at the expense of only an  $O(\log n)$ -factor to the preprocessing time.

- A More Practical Algorithm: We also believe that our algorithm using Dijkstra’s algorithm for SSSP computations is easier to implement and performs significantly better in practice than the algorithm by Klein [18]. We expect this to be the case since dynamic trees and persistence techniques are complicated to implement and incur large constant factors, even in their heavily optimized versions (see [24]). In contrast, it is well-established that Dijkstra’s algorithm performs extremely well on real-world graphs, and contractions can be implemented straight-forwardly.

In fact, one of the currently most successful experimental approaches to compute shortest-paths in road networks is already based on a framework of clever contraction hierarchies and fast SSSP computations (see for example [15]), and it is perceivable that our algorithm can be implemented rather easily by adapting the components of this framework.

We point out that our result can be shown to be tight in  $|f|$  and  $n$  by straight-forwardly extending the lower bound in [8]. We can report paths in the number of edges plus  $O(\log |f|)$  time.

## 2 Preliminaries

Given a graph  $H$ , we use  $V(H)$  to refer to the vertices of  $H$ , and  $E(H)$  to refer to its edges. We denote by  $w_H(e)$  or  $w_H(u, v)$  the weight of edge  $e = (u, v)$  in  $H$ , by  $d_H(u, v)$  the shortest distance from  $u$  to  $v$  in  $H$  and by  $P_H[u, v]$  a shortest path from  $u$  to  $v$ . By SSSP tree from a vertex  $u \in V(H)$ , we refer to the shortest path tree from  $u$  in  $H$  obtained by taking the union of all shortest paths starting in  $u$  (where we assume shortest paths satisfy the subpath property). We use  $T(r)$  to denote a tree rooted at a vertex  $r$ . For a vertex  $v \neq r$  of  $T$ , we let  $\pi_T(v)$  denote the parent of  $v$  in  $T$ .

**Induced Graph/Contractions.** For a vertex set  $X \subseteq V(H)$ , we let  $H[X]$  denote the subgraph of  $H$  induced by  $X$ . We sometimes abuse notation slightly and identify an edge set  $E'$  with the graph having edges  $E'$  and the vertex set consisting of endpoints of edges from  $E'$ . For any edge set  $E' \subseteq E(H)$ , we let  $H/E'$  denote the graph obtained from  $H$  by contracting edges in  $E'$  where we remove self-loops and retain only the cheapest edge between each vertex pair (breaking ties arbitrarily). If  $E$  only contains a single edge  $(u, v)$ , we slightly abuse notation and write  $H/(u, v)$  instead of  $H/\{(u, v)\}$ . When we contract components of vertices  $x_1, x_2, \dots, x_k$  into a super-vertex, we will identify the component with some vertex  $x_i$ . We use the convention that when we refer to some vertex  $x_j$  from the original graph in the context of the contracted graph, then  $x_j$  refers to the identified vertex  $x_i$ .

**Simplifying assumptions.** We let  $G = (V, E)$  refer to the input graph and assume that  $G$  is a planar embedded graph where the embedding is given by a standard rotation system, meaning that neighbors of a vertex are ordered clockwise around it. We assume that  $G$  has unique shortest paths, that  $f$  is the infinite face  $f_\infty$ , and that this face is a simple cycle with edges of infinite weight. We let  $V_\infty$  denote the vertex set of  $f_\infty$  and let  $r_0, r_1, \dots, r_{|V_\infty|-1}$  be the vertices of  $V_\infty$  in clockwise order, starting from some arbitrary vertex  $r_0$ .

We assume that no shortest path has an ingoing edge to a vertex of  $V_\infty$ ; in particular, infinite-weight edges are not allowed to be used in shortest paths. In addition, we assume that every vertex  $r_i \in V_\infty$  can reach every vertex  $u$  of  $V \setminus V_\infty$  in  $G[(V - V_\infty) \cup \{r_i\}]$ ; in words, there is a path from  $r_i$  to  $u$  that only intersects  $V_\infty$  in  $r_i$ .

With simple transformations, it is easy to show that all assumptions can be ensured with only  $O(n)$  additional preprocessing time; see Appendix A for details.

## 3 High-Level Overview

To make it easier to understand the formal construction and analysis of our new data structure, we start by giving a high-level overview.

**Preprocessing.** We first focus on the preprocessing step which constructs the data structure using a divide-and-conquer algorithm. A formal description can be found as pseudo-code in Algorithm 1 but here we only give a sketch.

The general subproblem is to compute shortest path trees from roots forming an interval  $I$  of vertices along  $f_\infty$  (the initial interval contains every vertex of  $f_\infty$ ). Shortest path trees are computed from the two endpoints of  $I$  as well as from the middle vertex of  $I$ . These three roots split  $I$  into two equal-sized sub-intervals on which the algorithm recurses. Since a shortest path

tree can be computed in linear time in a planar graph, the naive implementation would thus give an  $O(n|f_\infty|)$  running time.

To obtain an efficient construction, we rely on the following well-known result about shortest path trees from the roots on  $f_\infty$ : given two roots and a vertex  $u$ , the union of the shortest paths from the roots to  $u$  split  $f_\infty$  in two regions. If the shortest path trees from these two roots share an edge  $e = (u, v)$  lying in one of the regions, it holds that all roots in the other region contain  $e$  in their shortest path trees as well. An immediate corollary is that if the shortest path trees from the two roots share a tree  $T$  rooted at  $u$  and lying in one of the regions, then all roots in the other region contain  $T$  in their shortest path trees.

To exploit this result in the divide-and-conquer algorithm, let  $I'$  be one of the two sub-intervals of  $I$  above and let  $T_1$  and  $T_2$  be the two shortest path trees computed from the endpoints of  $I'$ . The intersection of edges of  $T_1$  and  $T_2$  forms a forest. Consider any tree  $T$  in this forest and the two regions defined by the endpoints of  $I'$  and the root of  $T$ , as in the previous paragraph. If  $T$  lies on the region not containing the roots in  $I'$ , then for the recursive call to  $I'$ ,  $T$  can be contracted since all shortest path trees computed in that recursive call must contain  $T$ ; see Figure 2 with  $T(s)$  playing the role of  $T$ .

Hence, instead of recursing on  $I'$  with the entire graph  $G$ , the algorithm instead recurses on the graph obtained from  $G$  by contracting all trees  $T$  satisfying the above condition. To ensure that contractions preserve shortest paths, let  $T(s)$  be one of the contracted trees. Then edge weights are modified as follows (see Figure 1):

- For every edge ingoing to  $T(s)$ , its weight is increased to  $\infty$  unless its endpoint is the root  $s$ ; this ensures that shortest paths can only enter  $T(s)$  through  $s$ .
- For every edge  $(u, v)$  outgoing from  $T(s)$ , its weight is increased by the shortest path distance from  $s$  to  $u$  in  $T(s)$ ; this ensures that the contraction of  $T(s)$  does not decrease shortest path distances.

It turns out that this simple preprocessing only requires  $O(n \log |f_\infty|)$  time. To sketch why this is true, consider any edge  $e$  of  $G$  and let  $I_e$  be the interval of roots of  $f_\infty$  whose shortest path trees contain  $e$ . Let us refer to the intervals obtained during the recursion as subproblem intervals. We can now make the following observations:

- If a subproblem interval  $I$  is disjoint from  $I_e$  then  $e$  is not part of any shortest path tree in the recursive call to  $I$ .
- If a subproblem interval  $I$  is contained in  $I_e$  then since  $e$  is contracted,  $e$  is also not part of any shortest path tree in the recursive call to  $I$ .
- On each recursion level, only  $O(1)$  subproblem intervals partially intersect  $I_e$ .

It follows that  $e$  is only part of  $O(\log |f_\infty|)$  shortest path trees in all recursive calls so the total size of all shortest path trees is  $O(n \log |f_\infty|)$ . The time spent on computing a shortest path tree in a graph  $H$  is linear in the size of  $H$ . By sparsity of simple planar graphs, the size of  $H$  is proportional to the number of tree edges. It follows that all shortest path trees can be computed in a total of  $O(n \log |f_\infty|)$  time.

We argue that the additional work spent in the recursive calls can also be executed within this time bound. For instance, determining whether to contract a tree  $T$  in the intersection of two shortest path trees  $T_1$  and  $T_2$  can be done by looking at the cyclic ordering of the two parent edges

of  $T_1$  resp.  $T_2$  ingoing to the root  $r$  of  $T$  and the edges of  $T$  emanating from  $r$ ; see Figure 2. This takes time linear in the size of the current graph over all trees  $T$ .

**Handling a query.** Efficiently answering a query is now simple, given the above preprocessing. Pseudo-code for the query algorithm can be found in Algorithm 2 but here we only describe it in words and sketch the analysis.

A query for the shortest path distance from a root  $r$  of  $f_\infty$  to a vertex  $u$  is done by following the path down the recursion tree for the subproblem intervals containing  $r$ . For each such subproblem interval  $I$ , if  $r$  is an endpoint of  $I$ , the shortest path distance is returned since it is stored in the shortest path tree computed from  $r$ . Otherwise, let  $T(s)$  be the tree that contains  $u$  and that is contracted for the next recursive step (this includes the special case where  $T(s)$  is the trivial tree consisting of the single vertex  $u$ ). The shortest path from  $r$  to  $u$  passes through the root  $s$  of  $T(s)$ . We compute the  $r$ -to- $u$  distance by recursively computing the  $r$ -to- $s$  distance and adding to it the  $s$ -to- $u$  distance; the latter is precomputed within the time bound for the preprocessing step above. As each recursive step takes  $O(1)$  time, we get a total query time of  $O(\log |f_\infty|)$ .

## 4 The MSSP Data Structure

We now give a formal description and analysis of our MSSP data structure.

The preprocessing procedure  $\text{MSSP}(I = [i_1, i_2], H_I)$  in Algorithm 1 starts by partitioning the interval  $I$  into two roughly equally sized subintervals  $[i_1, i]$  and  $[i, i_2]$ , in Lines 1-3. It then computes the shortest path trees of the boundary vertices  $r_{i_1}, r_{i_2}, r_i$  in the graph  $H_I$  (after removing the other boundary vertices). If  $i_2 - i_1 > 1$ , the data structure is recursively built for each of the two subintervals  $J = [j_1, j_2]$  in the loop starting in Line 5. To get the desired preprocessing time, the data structure ensures that the total size of all graphs at a given recursion level is  $O(n)$ . For each subinterval  $J$ , this is ensured by letting the graph  $H_J$  for the recursive call be a suitable contraction of  $H_I$ . More precisely,  $H_J$  is obtained from  $H_I$  by contracting suitable edges  $e$  that are guaranteed to be in the SSSP trees of every root  $r_j$  with  $j \in J$ . The construction of  $\mathcal{T}$  and the procedure  $\text{CONTRACT}$  handle the details of these contractions (illustrated in Figure 1). They also store in Line 11 the information necessary for later queries.

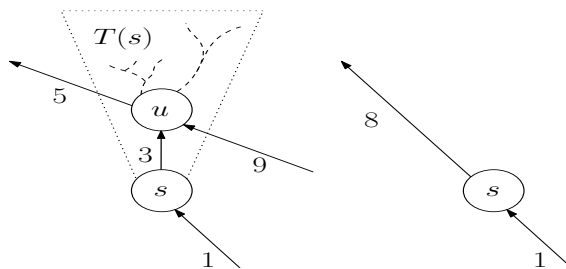


Figure 1: To the left, we have tree  $T(s)$ . To the right we contract  $T(s)$  and identify the supervertex with  $s$ . We update the weight of outgoing edges and remove the edges ingoing in  $T(s) - s$ .

A small but important implementation detail, we point out, is that while the pseudocode specifies that Algorithm 1 initializes each graph  $H_J$  as a copy of  $H_I$  in Line 6, the data structure

---

**Algorithm 1:** The procedure MSSP is given an interval  $I = [i_1, i_2]$  and a graph  $H_I$  obtained by contracting edges in  $G$ .  $H_I$  contains roots  $r_j$  for  $j \in I$ . The initial call is to  $([0, |V_\infty| - 1], G)$ .

---

**Procedure** MSSP( $I = [i_1, i_2], H_I$ )

```

1   $i \leftarrow \lfloor \frac{i_1+i_2}{2} \rfloor$ 
2  foreach  $k \in \{i_1, i_2, i\}$  do
3    Compute and store SSSP tree  $T_{I,k}$  from  $r_k$  in  $H_I$ 
4  if  $i_2 - i_1 \leq 1$  then return
5  foreach  $J = [j_1, j_2] \in \{[i_1, i], [i, i_2]\}$  do
6     $H_J \leftarrow H_I$ 
7     $E_{shared} \leftarrow E(T_{I,j_1}) \cap E(T_{I,j_2})$ 
8    Let  $\mathcal{T}$  be the collection of maximal vertex-disjoint trees  $T(s)$  rooted at  $s$  in  $E_{shared}$ 
      such that  $\pi_{T_{I,j_1}}(s) \neq \pi_{T_{I,j_2}}(s)$ , and for each child  $v$  of  $s$  in  $T(s)$  the edges
       $(s, v), (s, \pi_{T_{I,j_1}}(s)), (s, \pi_{T_{I,j_2}}(s))$  are clockwise around  $s$ 
9    foreach  $T(s) \in \mathcal{T}$  do  $H_J \leftarrow \text{CONTRACT}(H_J, T(s), i)$ 
10   MSSP( $J, H_J$ )

Procedure CONTRACT( $H_J, T(s), i$ )
11  foreach vertex  $u \in T(s)$  do  $(s_i(u), \delta_i(u)) \leftarrow (s, d_{T(s)}(s, u))$  // Global variables
12  foreach  $(u, v) \in E(H_J)$  with exactly one endpoint in  $T(s)$  do
13    if  $v \notin T(s)$  then  $w_{H_J}(u, v) \leftarrow w_{H_J}(u, v) + \delta_i(u)$  //  $(u, v)$  outgoing from  $T(s)$ 
14    else if  $v \neq s$  then  $w_{H_J}(u, v) \leftarrow \infty$  //  $(u, v)$  ingoing to  $T(s) - \{s\}$ 
15  Contract  $T(s)$  to a vertex in  $H_J$  and identify it with  $s$ 
16  return  $H_J$ 

```

---

technically only copies the subset  $\{r_{j_1}, r_{j_1+1}, \dots, r_{j_2}\}$  of the vertices on  $f_\infty$ . By our earlier simplifying assumption, the omitted roots are not part of any shortest path tree in any recursive calls involving sub-intervals of  $J$  so omitting them will not affect the behaviour of MSSP. Including the entire face  $f_\infty$  in all recursive calls, however, eases the presentation of proofs.

**Query.** The query procedure is straight-forward and given by Algorithm 2.

---

**Algorithm 2:** The procedure to query  $d_G(b_j, u)$ . Initial call is QUERY( $u, j, [0, |V_\infty| - 1]$ ).

---

**Procedure** QUERY( $u, j, I = [i_1, i_2]$ )

```

1  if  $j = i_1$  or  $j = i_2$  then return  $d_{T_{I,j}}(r_j, u)$ 
2   $i \leftarrow \lfloor \frac{i_1+i_2}{2} \rfloor$ 
3  if  $j \leq i$  then return QUERY( $s_i(u), j, [i_1, i]$ ) +  $\delta_i(u)$ 
4  else return QUERY( $s_i(u), j, [i, i_2]$ ) +  $\delta_i(u)$ 

```

---



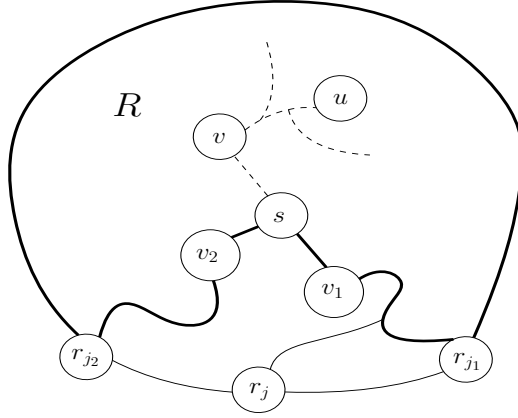


Figure 2: The shortest path trees from  $r_{j_1}$  and  $r_{j_2}$  share the dashed edges denoting the tree  $T(s)$  containing  $u$ . The shortest  $r_{j_1}$ -to- $s$ ,  $r_{j_2}$ -to- $s$  paths and the subpath from  $r_{j_2}$  to  $r_{j_1}$  in clockwise order along  $f_\infty$  define  $C$  (fat line). As  $v, v_1, v_2$  are in clockwise order around  $s$ ,  $T(s) - s$  is in  $R$ . For  $j \in [j_1, j_2]$ , any  $r_j$ -to- $u$  path intersects either the shortest  $r_{j_1}$ -to- $s$  or the shortest  $r_{j_2}$ -to- $s$  path.

## 5 Analysis

We now prove the following theorem which summarizes our main result.

**Theorem 5.1.** *Let  $G$  be an  $n$ -vertex planar embedded graph and  $f_\infty$  be the infinite face on  $G$ . Then we can build a data structure answering the distance  $d_G(b_j, u)$  between any vertex  $b_j \in f_\infty$  and any other vertex  $u$  in  $O(\log |f_\infty|)$  time, using procedure  $\text{QUERY}(u, j, [0, |f_\infty| - 1])$ . Preprocessing requires  $O(n \log |f_\infty|)$  time and space, using procedure  $\text{MSSP}([0, |f_\infty| - 1], G)$ .*

**Correctness.** Let us first prove correctness of the data structure. We start with the observation that no edge incident to  $f_\infty$  is ever contracted.

**Claim 5.2.** *For any graph  $H_I$ , no edge incident to  $f_\infty$  is contracted by  $\text{MSSP}(I, H_I)$ .*

*Proof.* This is immediate from our assumption that no shortest path has an ingoing edge to a vertex of  $f_\infty$  since only edges shared by shortest path trees are contracted.  $\square$

Next, we prove a lemma and its corollary that roughly show that the contractions made in Procedure  $\text{MSSP}$  do not destroy shortest paths from roots on sub-intervals  $J$  of  $f_\infty$ . The reader is referred to Figure 2 for intuition. We first need the following simple claim.

**Claim 5.3.** *For any invocation of  $\text{MSSP}(I, H_I)$ , any  $i \in I$ , and any  $u \in V(H_I) - V_\infty$ , some  $r_i$ -to- $u$  path exists in  $H_I[(V(H_I) - V_\infty) \cup \{r_i\}]$ .*

*Proof.* By assumption, an  $r_i$ -to- $u$  path exists in  $G[(V - V_\infty) \cup \{r_i\}]$ . The claim now follows since  $H_I$  is obtained from contractions in  $G \setminus V_\infty$  by Claim 5.2.  $\square$

**Lemma 5.4.** *Consider any invocation of  $\text{MSSP}(I, H_I)$  where  $H_I$  has unique shortest paths from  $r_i$  for each  $i \in I$ . Then for each vertex  $u \in T(s) \in \mathcal{T}$  in Line 8 and for each  $j \in J$ ,  $P_{H_I}[s, u] \subseteq P_{H_I}[r_j, u]$ .*

*Proof.* The proof is trivial for  $u = s$ , thus we assume  $u \neq s$ . Now consider paths  $P_{T_{I,j_1}}[r_{j_1}, s]$  and  $P_{T_{I,j_2}}[r_{j_2}, s]$ . Both paths exist by Claim 5.3. By uniqueness of shortest paths in  $H_i$ , they share only vertex  $s$ .

Next, consider the concatenation  $P[j_1, j_2]$  of  $P_{T_{I,j_1}}[r_{j_1}, s]$  and the reverse of  $P_{T_{I,j_2}}[r_{j_2}, s]$ . From the above,  $P[j_1, j_2]$  is simple. Further, consider the concatenation  $C$  of  $P[j_1, j_2]$  and the path segment  $F_\infty[r_{j_2}, r_{j_1}]$  from  $r_{j_2}$  to  $r_{j_1}$  in clockwise order around  $f_\infty$ . We note that  $C$  is a cycle. As  $f_\infty$  is simple, so is  $F_\infty[r_{j_2}, r_{j_1}]$ , and by our simplifying assumption for shortest paths and Claim 5.2,  $P[j_1, j_2]$  only intersects  $f_\infty$  in vertices  $r_{j_1}$  and  $r_{j_2}$ . Thus,  $C$  is a simple directed cycle.

By the Jordan curve theorem,  $C$  partitions the plane into two regions. One region, denoted  $R$ , is the region to the right when walking along  $C$ .  $T(s) - s$  does not intersect the simple curve  $P[j_1, j_2]$  since  $H_I$  is a planar embedded graph and since shortest paths are simple. It also does not intersect  $F_\infty[r_{j_2}, r_{j_1}]$  as there are no ingoing edges to  $f_\infty$  in  $G_\infty$  (and  $f_\infty$  is preserved in contractions by Claim 5.2). Since for each child  $v$  of  $s$  in  $T(s)$ , the edges  $(s, v)$ ,  $(s, \pi_{T_{I,j_1}}(s))$ ,  $(s, \pi_{T_{I,j_2}}(s))$  are clockwise around  $s$ , children of  $s$  are contained in  $R$ , and hence so is  $T(s) - s$ .

By the choice of  $F_\infty[r_{j_2}, r_{j_1}]$ ,  $r_j$  does not belong to  $R$  so  $P_{H_I}[r_j, u]$  intersects  $C$ . Since  $u \notin F_\infty[r_{j_2}, r_{j_1}]$  and since there is no edge  $(a, b)$  of  $H_I$  with  $a \notin V_\infty$  and  $b \in V_\infty$ ,  $P_{H_I}[r_j, u]$  cannot intersect  $F_\infty[r_{j_2}, r_{j_1}] - \{r_{j_1}, r_{j_2}\}$  so it must intersect  $C$  in  $P[j_1, j_2]$  and hence intersect either  $P_{T_{I,j_1}}[r_{j_1}, s]$  or  $P_{T_{I,j_2}}[r_{j_2}, s]$ ; assume the former (the other case is symmetric). Then  $P_{H_I}[r_j, u]$  intersects  $P_{T_{I,j_1}}[r_{j_1}, s]$  in some vertex  $x$ . Since shortest paths are unique, the subpath of  $P_{H_I}[r_j, u]$  from  $x$  to  $u$  must then equal  $P_{T_{I,j_1}}[x, u]$  and since  $s$  is on this subpath, the lemma follows.  $\square$

**Corollary 5.5.** *Let  $H_J$  be one of the graphs obtained in a call to  $\text{MSSP}(I, H_I)$  by contracting edges  $E'$  in  $H_I$ . Then, for each  $j \in J$  and each  $u \in V(H_J) - V_\infty$ ,  $P_{H_J}[r_j, u] = P_{H_I}[r_j, u]/E'$  is unique and  $w_{H_J}(P_{H_J}[r_j, u]) = w_{H_I}(P_{H_I}[r_j, u])$ .*

*Proof.* By Lemma 5.4, when contracting  $P_{H_I}[r_j, u]$  by  $E'$ , no edge on the path has its weight increased to  $\infty$ . Further, for any edge  $(x, y) \in P_{H_J}[r_j, u]$ , either  $(x, y)$  existed already in  $H_I$  in which case its weight is unchanged, or  $(x, y)$  originated from an edge  $(w, y)$  for  $w \in T(x)$ . But in the latter case,  $P_{H_I}[r_j, u]$  contains  $P_{H_I}[x, w]$  followed by  $(w, y)$  (Lemma 5.4), whose weight is  $d_{H_I}(x, w) + w_{H_I}(w, y) = d_{H_J}(x, y)$ . Thus  $w_{H_J}(P_{H_J}[r_j, u]) \leq w_{H_I}(P_{H_I}[r_j, u])$ .

It is straight-forward to see that distances in  $H_J$  also have not decreased since edges affected by the contractions obtain weights corresponding to paths in  $H_I$  between their endpoints or weight  $\infty$ . Uniqueness follows since two different shortest paths in  $H_J$  from  $r_j$  to a vertex  $u$  would imply two different shortest paths between the same pair in  $H_I$  and, by an inductive argument, also in  $G$ , contradicting our assumption of uniqueness.  $\square$

In fact, Corollary 5.5 is all we need to prove correctness of our algorithm.

**Lemma 5.6.** *The call  $\text{QUERY}(u, j, I = [i_1, i_2])$  outputs  $d_{H_I}(r_j, u)$ , for  $u \in V(H_I) - V_\infty$ . In particular,  $\text{QUERY}(u, j, I = [0, |V_\infty| - 1])$  outputs  $d_G(r_j, u)$  for  $u \in V$ .*

*Proof.* We prove this by induction on  $i_2 - i_1$ . If  $j \in \{i_1, i_2\}$  then  $\text{QUERY}(u, j, [i_1, i_2])$  directly returns  $d_{T_{I,j}}(r_j, u) = d_{H_I[(H_I - V_\infty) \cup \{r_j\}]}(r_j, u)$ . This, in turn, is equal to  $d_{H_I}(r_j, u)$  as the only  $r_j$ -to- $u$  paths not considered contain an  $\infty$ -weight edge with both endpoints in  $V_\infty$ . Therefore the claim holds if  $0 \leq i_2 - i_1 \leq 1$  as  $j \in \{i_1, i_2\}$  is implied.

For the inductive step, we thus assume  $i_2 - i_1 > 1$  and  $i_1 < j < i_2$ . Let  $i = \lfloor \frac{i_1 + i_2}{2} \rfloor$  and assume  $j \leq i$  (the case  $j > i$  is symmetric). Let  $s$  be the vertex in  $H_I$  such that  $u \in T(s)$ , where possibly  $u = s$  (Line 11). By the inductive hypothesis,  $\text{QUERY}(u, j, [i_1, i])$  returns  $\text{QUERY}(s_i(u), j, [i_1, i]) + \delta_i(u) =$

$d_{H_{[i_1, i]}}(r_j, s) + d_{T(s)}(s, u)$ . By Corollary 5.5,  $d_{H_{[i_1, i]}}(r_j, s) = d_{H_I}(r_j, s)$ . By definition of  $T(s)$ ,  $d_{T(s)}(s, u)$  is a suffix of  $d_{H_I}(r_{i_1}, u)$ , meaning that  $d_{T(s)}(s, u) = d_{H_I}(s, u)$ . Finally, by Lemma 5.4 the shortest path in  $H_I$  from  $r_j$  to  $u$  contains  $s$ , therefore  $d_{H_{[i_1, i]}}(r_j, s) + d_{T(s)}(s, u) = d_{H_I}(r_j, s) + d_{H_I}(s, u) = d_{H_I}(r_j, u)$ .  $\square$

**Bounding Time and Space.** The following Lemma captures our key insight about Algorithm 1 that ensures that the algorithm can be implemented efficiently.

**Definition 5.7.** Let  $\mathcal{I}_h$  be the set of all intervals  $I$ , such that  $\text{MSSP}(I, H_I)$  is executed at recursion level  $h$  after invoking  $\text{MSSP}([0, |V_\infty| - 1], G)$ .

**Lemma 5.8.** For each edge  $e = (u, v) \in E(G)$  and recursion level  $h$ , there are only  $O(1)$  intervals  $I \in \mathcal{I}_h$  for which there exists an  $i \in I$  such that the SSSP tree  $T_{I, i}$  from  $r_i$  in  $H_I$  contains  $e$ .

*Proof.* We use induction on  $h$ . As  $\mathcal{I}_0 = \{[0, |V_\infty| - 1]\}$  the Lemma is trivial for level  $h = 0$ . For  $h \geq 0$ , we show the inductive step  $h \mapsto h + 1$ . Each interval  $[i_1, i_2] = I \in \mathcal{I}_h$  satisfies exactly one of the following conditions:

- $\forall i \in I, e \notin E(T_{I, i})$ : Consider a recursive call  $\text{MSSP}(J, H_J)$  for  $J \subseteq I$  issued in  $\text{MSSP}(I, H_I)$  where  $H_J$  was obtained by contracting some edge set  $E'$  in  $H_I$ . By Corollary 5.5, for any  $j \in J$ ,  $P_{H_J}[r_j, u] = P_{H_I}[r_j, u]/E'$ , but due to our condition and  $j \in I$ , this path cannot contain  $e$ .
- $\forall i \in I, e \in E(T_{I, i})$ : Let  $\mathcal{I}'_h$  be the subset of intervals  $I \in \mathcal{I}_h$  satisfying this condition and let  $\mathcal{J}'_{h+1}$  be the intervals  $J \in \mathcal{I}_{h+1}$  contained in such intervals  $I$ . We show that for at most one  $I \in \mathcal{I}'_h$  does there exist a sub-interval  $J \in \mathcal{J}'_{h+1}$  where  $e$  has not been contracted in  $H_J$ .

Consider the set  $\mathcal{P}$  of shortest paths in  $G$  from each endpoint of an interval in  $\mathcal{J}'_{h+1}$  to vertex  $u$ . Their union is a tree  $T$  with leaves in  $V_\infty$  and with all edges directed towards the root  $u$ ; see the dashed paths in Figure 3. By definition of  $\mathcal{I}'_h$  and  $\mathcal{J}'_{h+1}$  and by repeated applications of Corollary 5.5 to intervals containing  $I$  at recursion levels less than  $h$ , each shortest path in  $\mathcal{P}$  is a subpath of a shortest path to  $v$  in  $G$  with  $(u, v)$  as the final edge. Since shortest paths are simple,  $v$  cannot belong to  $T$ .

Now, consider an interval  $J = [j_1, j_2] \in \mathcal{J}'_{h+1}$ . Let  $x$  be the nearest common ancestor of  $r_{j_1}$  and  $r_{j_2}$  in  $T$ . Let  $C_J$  be the simple directed cycle obtained by the concatenation of the path  $T[r_{j_1}, x]$ , the reverse of the path  $T[r_{j_2}, x]$ , and the path from  $r_{j_2}$  to  $r_{j_1}$  in counter-clockwise order along  $f_\infty$ . Let  $R_J$  be the open region of the plane to the left of  $C_J$ . Note that shortest path  $P_G[x, v]$  is  $T[x, u]$  concatenated with  $(u, v)$ . If  $P_G[x, v]$  emanates to the left of  $C_J$  at  $x$  then since a shortest path cannot cross itself,  $v$  must belong to  $R_J$ . In this case,  $v$  cannot also belong to  $R_{J'}$  for any other  $J' \in \mathcal{J}'_{h+1}$  since these regions are pairwise disjoint; see Figure 3. Hence, there is at most one choice of  $J$  where  $P_G[x, v]$  emanates to the left of  $C_J$  at  $x$ .

Using the above notation, it now suffices to show that for an interval  $J = [j_1, j_2] \in \mathcal{J}'_{h+1}$  where  $P_G[x, v]$  emanates to the right of  $C_J$  at  $x$ ,  $e$  is contracted in  $H_J$ .

Let  $I \in \mathcal{I}'_h$  be the parent interval containing  $J$ . By Corollary 5.5,  $P_{H_I}[r_{j_1}, v]$  is obtained from  $P_G[r_{j_1}, v]$  by edge contractions and  $P_{H_I}[r_{j_2}, v]$  is obtained similarly from  $P_G[r_{j_2}, v]$ . Let  $(u_1, x)$  resp.  $(u_2, x)$  be the ingoing edge to  $x$  in  $P_G[r_{j_1}, v]$  resp.  $P_G[r_{j_2}, v]$ . Edge  $(u_1, x)$  is not in the shortest path tree from  $r_{j_2}$  in  $G$  (otherwise, this tree would have both  $(u_1, x)$  and  $(u_2, x)$  ingoing to  $x$ ) so by Corollary 5.5,  $(u_1, x)$  is not contracted in  $H_I$ . Similarly,  $(u_2, x)$

is not contracted in  $H_I$ . It follows that  $x$  is a vertex in  $H_I$ . Let  $(x, y)$  be the first edge on  $P_{H_I}[x, v]$ ; this is well-defined since  $P_{H_I}[x, v]$  contains at least one edge, namely  $e$ . By the choice of  $J$  and by the embedding-preserving properties of contractions,  $(x, y)$ ,  $(x, \pi_{T_I, j_1}(x))$ , and  $(x, \pi_{T_I, j_2}(x))$  are clockwise around  $x$ . Inspecting the description of  $\text{MSSP}(I, H_I)$ , we see that  $P_{H_I}[x, v]$  is contained in a tree  $T(s) \in \mathcal{T}$  with  $s = x$ , so  $e$  is contracted in  $H_J$ .

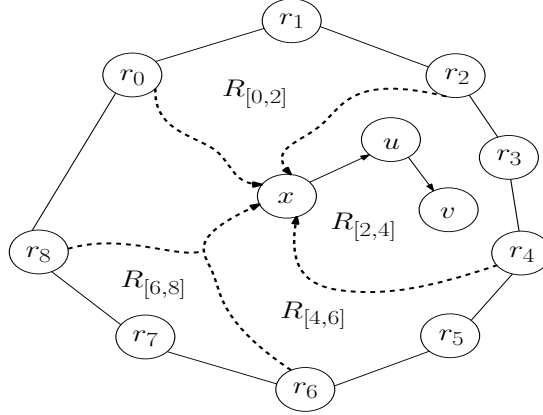


Figure 3: Dashed lines are parts of shortest paths  $P_G[r_i, v]$  from vertices  $r_i$  in  $f_\infty$  to  $v$  and form a tree. In the second recursion level we have intervals  $I_0 = [0, 2], I_1 = [2, 4], I_2 = [4, 6], I_3 = [6, 8]$ . Vertex  $x$  is the nearest common ancestor of  $r_2$  and  $r_4$  and  $R_{[2,4]}$  is the open region to the left of the simple cycle defined by  $P_G[r_2, x]$ , the reverse of  $P_G[r_4, x]$  and the part of  $f_\infty$  from  $r_4$  to  $r_2$  in counter-clockwise order. Regions  $R_{[0,2]}, R_{[4,6]}, R_{[6,8]}$  are defined similarly for the other intervals. Focusing on interval  $I_1$ , let  $y$  be the vertex after  $x$  towards  $u$  in the tree (in this case  $y = u$ ). Edges  $(x, y), (x, \pi_{P_G[r_2, u]}(x)), (x, \pi_{P_G[r_4, u]}(x))$  are in counter-clockwise order, implying that  $v$  belongs to  $R_{[2,4]}$ . Since regions are pairwise disjoint,  $v$  belongs to no other region.

- $\exists i, i' \in I, e \in E(T_{I,i})$  and  $e \notin E(T_{I,i'})$ : At most two intervals  $I \in \mathcal{I}_h$  satisfy this condition which can then spawn at most four intervals in  $\mathcal{I}_{h+1}$ ; this follows from repeated applications of Corollary 5.5 combined with the easy observation that roots of  $f_\infty$  whose shortest path trees in  $G$  contain  $e$  are consecutive in the cyclic ordering along  $f_\infty$  [18].

□

**Corollary 5.9.** *The procedure  $\text{MSSP}([0, |V_\infty| - 1], G)$  uses  $O(n \log |f|)$  time and space. Procedure  $\text{QUERY}(u, j, [0, |V_\infty| - 1])$  has  $O(\log |f|)$  running time.*

*Proof.* For each recursion level  $h$ , Lemma 5.8 implies that each edge  $e \in E(G)$  appears in at most  $O(1)$  computed SSSP trees. Further,  $\sum_{I \in \mathcal{I}_h} |I| = O(n)$ . Thus, the total number of vertices in  $H_I$  summed over all  $I \in \mathcal{I}_h$  is  $O(n)$ . By sparsity of planar graphs, the total number of edges in these graphs is  $O(n)$  as well.

For each such graph  $H_I$ , each SSSP computation in  $H_I$  can be implemented in time linear in the size of  $H_I$  by [16]. We also spend this amount time on constructing the graphs  $H_J$  from  $H_I$  since the set  $E_{\text{shared}} \subseteq E(H_I)$  and associated trees  $T(s)$  can be found in  $O(1)$  time per edge. Each

edge is outgoing from at most one and ingoing to at most one contracted tree and thus possibly changing its weight requires only  $O(1)$  time.

It follows that the entire time spent on recursion level  $h$  is  $O(n)$ . Since there are at most  $O(\log |f_\infty|)$  levels, the first part of the corollary follows. The second part follows since each recursive step in QUERY takes constant time.  $\square$

## References

- [1] G. BORRADAILE, P. N. KLEIN, S. MOZES, Y. NUSSBAUM, AND C. WULFF-NILSEN, *Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time*, SIAM J. Comput., 46 (2017), pp. 1280–1303.
- [2] G. BORRADAILE, P. SANKOWSKI, AND C. WULFF-NILSEN, *Min st-cut oracle for planar graphs with near-linear preprocessing time*, ACM Trans. Algorithms, 11 (2015).
- [3] S. CABELLO, E. W. CHAMBERS, AND J. ERICKSON, *Multiple-source shortest paths in embedded graphs*, SIAM Journal on Computing, 42 (2013), pp. 1542–1571.
- [4] P. CHARALAMPOPOULOS, P. GAWRYCHOWSKI, S. MOZES, AND O. WEIMANN, *Almost optimal distance oracles for planar graphs*, in Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, 2019, pp. 138–151.
- [5] V. COHEN-ADDAD, S. DAHLGAARD, AND C. WULFF-NILSEN, *Fast and compact exact distance oracle for planar graphs*, in 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2017, pp. 962–973.
- [6] E. DEMAINE, S. MOZES, C. SOMMER, AND S. TAZARI, *Lecture 11 in 6.889: Algorithms for planar graphs and beyond (fall 2011)*, 2011.
- [7] J. R. DRISCOLL, N. SARNAK, D. D. SLEATOR, AND R. E. TARJAN, *Making data structures persistent*, Journal of computer and system sciences, 38 (1989), pp. 86–124.
- [8] D. EISENSTAT AND P. N. KLEIN, *Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graphs*, in Proceedings of the forty-fifth annual ACM symposium on Theory of computing, 2013, pp. 735–744.
- [9] J. ERICKSON, *Lecture 14 in cs 598 jge: One-dimensional computational topology*, 2020.
- [10] J. ERICKSON, K. FOX, AND L. LKHAMSUREN, *Holiest minimum-cost paths and flows in surface graphs*, in Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25–29, 2018, I. Diakonikolas, D. Kempe, and M. Henzinger, eds., ACM, 2018, pp. 1319–1332.
- [11] J. FAKCHAROENPHOL AND S. RAO, *Planar graphs, negative weight edges, shortest paths, and near linear time*, Journal of Computer and System Sciences, 72 (2006), pp. 868–889.
- [12] V. FREDSLUND-HANSEN, S. MOZES, AND C. WULFF-NILSEN, *Truly subquadratic exact distance oracles with constant query time for planar graphs*, arXiv preprint arXiv:2009.14716, (2020).

- [13] P. GAWRYCHOWSKI AND A. KARZMARZ, *Improved bounds for shortest paths in dense distance graphs*, in 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- [14] P. GAWRYCHOWSKI, S. MOZES, O. WEIMANN, AND C. WULFF-NILSEN, *Better tradeoffs for exact distance oracles in planar graphs*, in Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, no. SODA '18, 2018, pp. 515–529.
- [15] R. GEISBERGER, P. SANDERS, D. SCHULTES, AND D. DELLING, *Contraction hierarchies: Faster and simpler hierarchical routing in road networks*, in International Workshop on Experimental and Efficient Algorithms, Springer, 2008, pp. 319–333.
- [16] M. R. HENZINGER, P. N. KLEIN, S. RAO, AND S. SUBRAMANIAN, *Faster shortest-path algorithms for planar graphs*, J. Comput. Syst. Sci., 55 (1997), pp. 3–23.
- [17] G. F. ITALIANO, Y. NUSSBAUM, P. SANKOWSKI, AND C. WULFF-NILSEN, *Improved algorithms for min cut and max flow in undirected planar graphs*, in Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, STOC '11, New York, NY, USA, 2011, Association for Computing Machinery, p. 313–322.
- [18] P. N. KLEIN, *Multiple-source shortest paths in planar graphs*, in Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005, SIAM, 2005, pp. 146–155.
- [19] P. N. KLEIN AND S. MOZES, *Optimization algorithms for planar graphs*, preparation, manuscript at <http://planarity.org>, (2014).
- [20] Y. LONG AND S. PETTIE, *Planar distance oracles with better time-space tradeoffs*, in Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA'21, 2021, pp. 2517–2537.
- [21] Y. LONG AND S. PETTIE, *Planar distance oracles with better time-space tradeoffs*, in Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2021, pp. 2517–2537.
- [22] S. MOZES, Y. NUSSBAUM, AND O. WEIMANN, *Faster shortest paths in dense distance graphs, with applications*, Theoretical Computer Science, 711 (2018), pp. 11–35.
- [23] S. MOZES AND C. WULFF-NILSEN, *Shortest paths in planar graphs with real lengths in  $o(n \log^2 n / \log \log n)$  time*, in European Symposium on Algorithms, Springer, 2010, pp. 206–217.
- [24] R. E. TARJAN AND R. F. WERNECK, *Dynamic trees in practice*, Journal of Experimental Algorithmics (JEA), 14 (2010), pp. 4–5.
- [25] M. THORUP, *Compact oracles for reachability and approximate distances in planar digraphs*, Journal of the ACM (JACM), 51 (2004), pp. 993–1024.

## A Satisfying the input assumptions

In this section, we show how to ensure the input assumptions from the preliminaries.

If  $f$  is not the external face  $f_\infty$ , a linear time algorithm can reembed  $G$  to ensure this.

Next, we need  $f_\infty$  to be a simple cycle. We let  $b_0, b_1, \dots, b_{|f_\infty|-1}$  be the vertices on  $f_\infty$  ordered by their first appearance in the walk along  $f_\infty$  in clockwise order starting in an arbitrary vertex  $b_0$ , such that the rest of  $G$  is on the right when moving on the walk. For each  $0 \leq i < |f_\infty|$ , we add a vertex  $r_i$ , and a zero-weight edge  $(r_i, b'_i)$  embedded in the region enclosed by  $f_\infty$  that does not contain the rest of  $G$ . Finally, we add edges  $(r_i, r_{i+1 \bmod |f_\infty|})$  in a simple cycle, with infinite weights, and redefine  $f_\infty$  to be this cycle. It is not hard to see that this transformation can be done in  $O(n)$  time and that any query  $(b_i, u)$  in the original graph can be answered by querying  $(r_i, u)$  in the transformed graph.

Let  $V_\infty = \{r_0, r_1, \dots, r_{|f_\infty|-1}\}$ . To ensure the requirement that every  $r_i \in V_\infty$  can reach every vertex of  $V \setminus V_\infty$  in  $G[(V - V_\infty) \cup \{r_i\}]$ , we add suitable edges of large finite weight to  $G \setminus V_\infty$  to make this subgraph strongly connected without violating the embedding of  $G$ .

Disallowing shortest paths from having edges ingoing to  $V_\infty$  will not disallow any shortest path from  $b_i$  in the original graph: each such path can be mapped to a path of the same weight in the transformed graph that only intersects  $f_\infty$  in  $r_i$ .

Finally, unique shortest paths can be ensured using the deterministic perturbation technique in [10]).

## Appendix D

# CPM: Longest Common Subsequence on Weighted Sequences



# Longest Common Subsequence on Weighted Sequences

Evangelos Kipouridis 

Basic Algorithms Research Copenhagen (BARC), University of Copenhagen, Denmark  
kipouridis@di.ku.dk

Kostas Tsihclas

Computer Engineering and Informatics Department, University of Patras, Greece  
ktsichlas@ceid.upatras.gr



---

## Abstract

We consider the general problem of the Longest Common Subsequence (*LCS*) on weighted sequences. Weighted sequences are an extension of classical strings, where in each position every letter of the alphabet may occur with some probability. Previous results presented a *PTAS* and noticed that no *FPTAS* is possible unless  $P = NP$ . In this paper we essentially close the gap between upper and lower bounds by improving both. First of all, we provide an *EPTAS* for bounded alphabets (which is the most natural case), and prove that there does not exist any *EPTAS* for unbounded alphabets unless  $FPT = W[1]$ . Furthermore, under the Exponential Time Hypothesis, we provide a lower bound which shows that no significantly better *PTAS* can exist for unbounded alphabets. As a side note, we prove that it is sufficient to work with only one threshold in the general variant of the problem.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis; Theory of computation → W hierarchy; Theory of computation → Problems, reductions and completeness

**Keywords and phrases** WLCS, LCS, weighted sequences, approximation algorithms, lower bound

**Funding** *Evangelos Kipouridis*: Thorup’s Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation, and European Union’s Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 801199.  

**Acknowledgements** We would like to thank the anonymous reviewers for their careful reading of our paper and their many insightful comments and suggestions.

## 1 Introduction

### 1.1 General concepts

We consider the problem of determining the *LCS* (Longest Common Subsequence) on weighted sequences. Weighted sequences, also known as  $p$ -weighted sequences or Position Weighted Matrices (PWM) [3, 35] are probabilistic sequences which extend the notion of strings, in the sense that in each position there is some probability for each letter of an alphabet  $\Sigma$  to occur there.

Weighted sequences were introduced as a tool for motif discovery and local alignment and are extensively used in molecular biology [23]. They have been studied both in the context of short sequences (binding sites, sequences resulting from multiple alignment, etc.) and on large sequences, such as complete chromosome sequences that have been obtained using a whole-genome shotgun strategy [31, 36]. Weighted sequences are able to keep all the information produced by such strategies, while classical strings impose restrictions that oversimplify the original data.

Basic concepts concerning the combinatorics of weighted sequences (like pattern matching,

repeats discovery and cover computation) were studied using weighted suffix trees [26], Crochemore’s partitioning [9, 11, 18], the Karp-Miller-Rabin algorithm [18], and other approaches [42, 29]. Other interesting results include approximate and gapped pattern matching [6, 40, 33], online pattern matching [16], weighted indexing [2, 10], swapped matching [39], the all-covers and all-seeds problem [38, 41], extracting motifs [28], and the weighted shortest common supersequence problem [4, 17]. There are also some more practical results on mapping short weighted sequences to a reference genome [7] (also studied in the parallel setting [27]), as well as on the reporting version of the problem which we also consider in this paper [11].

The Longest Common Subsequence (*LCS*) problem is a well-known measure of similarity between two strings. Given two strings, the output should be the length of the longest subsequence common to both strings. Dynamic programming solutions [25, 37] for this problem are classical textbook algorithms in Computer Science. *LCS* has been applied in computational biology for measuring the commonality of DNA molecules or proteins which may yield similar functionality. A very interesting survey on algorithms for the *LCS* can be found in [13]. The current *LCS* algorithms are considered optimal, since matching lower bounds (under the Strong Exponential Time Hypothesis) were proven [1, 14].

Extensions of this problem on more general structures have also been investigated (trees and matrices [5], run-length encoded strings [8], and more). One interesting variant of the *LCS* is the Heaviest Common Subsequence (*HCS*) where the matching between different letters is assigned a different weight, and the goal is to maximize the weight of the common subsequence, rather than its length.

## 1.2 Weighted LCS

The problem studied in this paper is the weighted *LCS* (WLCS) problem. It was introduced by Amir et al. [3] as an extension of the classical *LCS* problem on weighted sequences. Given two weighted sequences, the goal is to find a longest string which has a high probability of appearing in both sequences. Amir et al. initially solved an easier version of this problem in polynomial time, but unfortunately its applications are limited. As far as the general problem is concerned, they hinted its NP-Hardness by giving an NP-Hardness result on a closely related problem, which they call the log-probability version of WLCS. In short, the problem is the same, but all products in its definition are replaced with sums. Their proof is based on a Turing reduction and only works for unbounded alphabets. Finally, Amir et al. provide an  $\frac{1}{|\Sigma|}$ -approximation algorithm for the WLCS problem.

Cygan et al. [19] strengthened the evidence that WLCS is NP-Hard by providing an NP-Completeness result on the decision log-probability version of WLCS (informally introduced in the previous paragraph), already for alphabets of size 2, using a Karp reduction; for alphabets of size 1 the solution is trivial since there is no uncertainty. They also gave an  $\frac{1}{2}$ -approximation algorithm and a *PTAS*, while also noticing that an *FPTAS* cannot exist, assuming WLCS is indeed NP-Hard, as hinted by their evidence, and that  $P \neq NP$ . Finally, they proved that every instance of the problem can be reduced to a more restricted class of instances. However, for this to be achieved their algorithm needs to perform exact computations of roots and logarithms that may make the algorithm to err.

Finally, it is worth noting that Charalampopoulos et al. [17], proved that unless  $P=NP$ , WLCS cannot be solved in  $\mathcal{O}(n^{f(a)})$  time, for any function  $f(a)$ , where  $a$  is the cut-off probability. We note that this result concerns exact computations rather than approximations.

### 1.3 Our results

In this paper we essentially close the gap between upper and lower bounds for WLCS by improving both; we prove that the problem is indeed NP-Hard even for alphabets of size 2. Furthermore, we provide an *EPTAS* for bounded alphabets. These two results, along with the *FPTAS* observation by Cygan et al. completely characterize the complexity of WLCS for bounded alphabets. For unbounded alphabets, a *PTAS* was already known by Cygan et al. [19]. We show matching lower bounds, both by ruling out the possibility of an *EPTAS*, and by showing that, under the Exponential Time Hypothesis, no significantly better *PTAS* can exist. We also prove that every instance of WLCS can be reduced to a restricted class of instances without using roots and logarithms, thus being able to actually achieve exact computations without rounding errors that can make the algorithm err.

As noted in the previous paragraph, apart from essentially closing the gap between hardness results and faster algorithms we also circumvent the need to work with roots and logarithms as the previous results did. In short, by taking advantage of the property that  $(ab)^c = a^c b^c$  and setting  $c$  to be an appropriate logarithm, previous results transformed any instance to a more manageable form. However, this transformation introduces an error that may make the algorithm err as noted in Appendix A. Table 1 summarizes the above discussion. Table 2 summarizes our results depending on the alphabet-size.

A short discussion is in order with respect to what new insights on weighted *LCS* enabled us to achieve progress. Our most crucial observation is the fact that the problem behaves differently in the natural case of a bounded alphabet, and in the case of an unbounded alphabet. Without this distinction, closing the gap between upper and lower bounds was unlikely. That's because, on the one hand, no *EPTAS* for the general case could be found, as none existed. On the other hand, proving that no *EPTAS* exists requires reductions that work only on unbounded alphabets. The aforementioned distinction is what enabled us to understand that modifying the existing reductions, which work for alphabets of size 2, would be futile in proving  $W[1]$ -Hardness.

Furthermore, it was crucial to identify that working with products is the core difficulty in proving NP-Hardness of weighted *LCS*. The introduction of the log-probability version of the weighted *LCS* reflects the assumption that the difference between working with sums and working with products is just a technicality. After [3] and [19] successfully proved NP-Hardness for the log-probability version, it was natural to attempt reducing from it for proving NP-Hardness of the weighted *LCS* problem. Despite the apparent similarities between the two problems, their difference did not allow us to craft such a reduction. For the same reason, Cygan et al. used a model that assumed infinite precision computations with reals, while we are able to avoid such a strong assumption.

### 1.4 Organization of the paper

The rest of the paper is organized as follows. In Section 2, we provide necessary definitions and discuss the model of computation. In Section 3, we show that WLCS is NP-Complete while in Section 4, we provide the *EPTAS* algorithm for bounded alphabets, which is also an improved *PTAS* for unbounded alphabets. In Section 5, we show that there can be no *EPTAS* for unbounded alphabets by showing that this problem is  $W[1]$ -hard and in Section 6, we describe the matching conditional lower bound. We conclude in Section 7.

For clarity purposes, some proofs and technical discussions are moved to the Appendix. More specifically, in Appendix A we present an algorithm that transforms any instance of our problem to an equivalent, but easier to handle, instance. We also show that the rounding

## 4 Longest Common Subsequence on Weighted Sequences

errors introduced by working with reals (logarithms and roots) may cause a similar algorithm by Cygan et al. [19] to err if standard rounding is used.

■ **Table 1** Results on WLCS.

|                                                           | Amir et al.                                                                                         | Cygan et al.                                                                                       | Our results                                                            |
|-----------------------------------------------------------|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|
| NP-Hardness of WLCS                                       | Hinted, by NP-Hardness of Log-probability version (Turing reduction - only for unbounded alphabets) | Hinted, by NP-Hardness of Log-probability version (Karp reduction - already from alphabets size 2) | Proved (Karp Reduction - already from alphabets of size 2)             |
| Approximation Algorithms                                  | $\frac{1}{\Sigma}$ -Approximation                                                                   | <i>PTAS</i>                                                                                        | <i>EPTAS</i> for bounded alphabets, Improved <i>PTAS</i> for unbounded |
| Proof that no <i>EPTAS</i> exists for unbounded alphabets | No                                                                                                  | No                                                                                                 | Yes                                                                    |
| Lower bound on any <i>PTAS</i>                            | No                                                                                                  | No                                                                                                 | Matching the upper bound, under <i>ETH</i>                             |
| Reduction to a restricted class of instances              | No                                                                                                  | Yes, by assuming exact computations of logarithms                                                  | Yes, without any extra assumptions                                     |

■ **Table 2** Results depending on the Alphabet Size

| Alphabet Size          | Previous Results         | Our results                                                           |
|------------------------|--------------------------|-----------------------------------------------------------------------|
| 1                      | Trivial                  | Trivial                                                               |
| Constant Size          | No <i>FPTAS</i> possible | Achieved <i>EPTAS</i>                                                 |
| Depending on the input | Achieved <i>PTAS</i>     | No <i>EPTAS</i> possible, Improved <i>PTAS</i> , Matching Lower Bound |

## 2 Preliminaries

### 2.1 Basic Definitions

Let  $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_K\}$  be a finite alphabet. We deal both with bounded ( $K = O(1)$ ) and unbounded alphabets.  $\Sigma^d$  denotes the set of all words of length  $d$  over  $\Sigma$ .  $\Sigma^*$  denotes the set of all words over  $\Sigma$ .

► **Definition 1** (Weighted Sequence). *A weighted sequence  $X$  is a sequence of functions  $p_1^{(X)}, \dots, p_{|\Sigma|}^{(X)}$ , where each function assigns a probability to each letter from  $\Sigma$ . We thus have  $\sum_{j=1}^K p_i^{(X)}(\sigma_j) = 1$  for all  $i$ , and  $p_i^{(X)}(\sigma_j) \geq 0$  for all  $i, j$ .*

By  $WS(\Sigma)$  we denote the set of all weighted sequences over  $\Sigma$ . Let  $X \in WS(\Sigma)$ . Let  $Seq_d^{|X|}$  be the set of all increasing sequences of  $d$  positions in  $X$ . For a string  $s \in \Sigma^d$  and  $\pi \in Seq_d^{|X|}$ , define  $P_X(\pi, s)$  as the probability that the subsequence on positions corresponding to  $\pi$  in  $X$  equals  $s$ . More formally, if  $\pi = (i_1, i_2, \dots, i_d)$  and  $s_k$  denotes the  $k$ -th letter of  $s$ , then

$$P_X(\pi, s) = \prod_{k=1}^d p_{i_k}^{(X)}(s_k)$$

Denote

$$SUBS(X, a) = \{s \in \Sigma^* \mid \exists \pi \in Seq_{|s|}^{|X|} \text{ such that } P_X(\pi, s) \geq a\}$$

That is,  $SUBS(X, a)$  is the set of deterministic strings which match a subsequence of  $X$  with probability at least  $a$ . Every  $s \in SUBS(X, a)$  is called an  $a$ -subsequence of  $X$ .

Let us give a clarifying example. If  $\Sigma = \{\sigma_1, \sigma_2\}$  and  $X$  is a long weighted sequence, where in each position the probability for each letter to appear is 0.5, then  $SUBS(X, 0.3)$  does not contain  $s = \sigma_1\sigma_1$ , as, for any increasing subsequence of 2 positions, the probability of  $s$  appearing is  $0.25 < 0.3$ .

The decision problem we consider is the following:

► **Definition 2** ( $(a_1, a_2)$ -WLCS decision problem). *Given two weighted sequences  $X, Y$ , two cut-off probabilities  $a_1, a_2$  and a number  $k$ , find if the longest string  $s$  contained in  $SUBS(X, a_1) \cap SUBS(Y, a_2)$  has length at least  $k$ .*

Naturally, the respective optimization problem is the following:

► **Definition 3** ( $(a_1, a_2)$ -WLCS optimization problem). *Given two weighted sequences  $X, Y$ , and two cut-off probabilities  $a_1, a_2$ , find the length of the longest string contained in  $SUBS(X, a_1) \cap SUBS(Y, a_2)$ .*

Both in the decision and the optimization version, the WLCS problem is the  $(a_1, a_2)$ -WLCS problem, where  $a_1 = a_2$ . We denote these (equal) probabilities by  $a$  ( $a = a_1 = a_2$ ) for concreteness.

Let us note that the problem is only interesting if  $|\Sigma| \geq 2$ . For  $|\Sigma| = 1$  the problem is trivial since there is no uncertainty at all. The same letter appears in every position in both strings with probability 1, and thus the answer is simply the length of the shorter weighted sequence.

Finally, let us also state that the Log-Probability version of the WLCS, studied in previous papers, is the same as the original WLCS if we replace  $P_X(\pi, s) = \prod_{k=1}^d p_{i_k}^{(X)}(s_k)$  by  $P_X(\pi, s) = \sum_{k=1}^d p_{i_k}^{(X)}(s_k)$ .

## 2.2 Model of Computation

Our model of computation is the standard word *RAM*, introduced by Fredman and Willard [20] to simulate programming languages like C. The word size is  $w = \Omega(\log I)$ , where  $I$  is the input size in bits, so as to allow random access indexing of the whole input. Thus, arithmetic operations between words take constant time. However, due to the nature of our problem, it is necessary to compute products of many numbers. This can produce numbers that are much larger than the word size. We even allow numbers in the input to be larger than  $2^w$  (these numbers just need to use more than one word to be represented). We generally assume that each number in the input is represented by at most  $B$  bits, but we do not pose any constraint on  $B$  other than the trivial one that  $B < I$ . Of course, in cases where we deal with numbers that occupy many words, we no longer have unit-cost arithmetic operations; we guarantee, however, that our results only use linear or near-linear time operations (like comparisons and multiplications) on numbers polynomial in the input size. Thus, although we do not enjoy the unit-cost assumption for arbitrary numbers, we always stay within the polynomial-time regime.

### 2.3 Basic Operations

In this subsection we discuss the multiplication of two  $B$ -bit input numbers in (polynomial)  $Mul_w(B)$  time, where  $w$  is the word-size. For example, for integers there exists a multiplication algorithm by Harvey and van der Hoeven [24] with time complexity  $Mul_w(B) = \mathcal{O}(B \log B)$  (generally the running time can also depend on  $w$ , although in this case it does not). Let us notice that although the result is unpublished yet, we use it due to its easy to read time complexity; it is trivial to use other algorithms instead, like the one from Fürer [21], or the more practical one by Schönhage and Strassen [34]. We establish the complexity of multiplying  $x$   $B$ -bit numbers. Our divide and conquer algorithm splits the numbers into two (equal sized) groups, recursively multiplies each, and multiplies the results in  $Mul_w(\frac{xB}{2})$  time. By a direct application of the Master Theorem by Bentley et al. [12] we prove the following lemma.

- **Lemma 4.** *Multiplying  $x$   $B$ -bit numbers costs*
- $\mathcal{O}(Mul_w(xB) \log(xB))$  time if  $Mul_w(xB) = \Theta(xB \log^k(xB))$  for some constant  $k$ ,
  - $\mathcal{O}((xB)^c)$  else if  $Mul_w(xB) = \mathcal{O}((xB)^c)$  for some constant  $c \geq 1$ ,
- assuming that  $Mul_w(N)$  is a polynomial time algorithm that multiplies two  $N$ -bit numbers.

**Proof.** The algorithm simply splits the numbers in two equal-sized groups, recursively multiplies each, and then multiplies the results. Let  $N = xB$ . We have that the running time for multiplying  $x$   $B$ -bit numbers is  $T(N) = 2T(\frac{N}{2}) + Mul_w(N)$ . Since  $c_{crit} = \log_2 2 = 1$ , and  $Mul_w(N) = \Omega(N)$ , the Master Theorem [12] gives two cases. Either  $Mul_w(N) = \Theta(N \log^k(N))$  for some constant  $k$ , in which case  $T(N) = \mathcal{O}(Mul_w(N) \log N)$ , or else  $Mul_w(N) = \mathcal{O}(N^c)$  for some constant  $c \geq 1$  (such a constant exists since we assume polynomial time multiplications). In this case, since it holds that  $2Mul_w(\frac{N}{2}) \leq 2Mul_w(N)$ , we get that  $T(N) = Mul_w(N)$  if  $c > c_{crit} = 1$ . Notice that we handled all cases, since  $Mul_w(N) = N$  is handled by the first case with  $k = 0$ , and whatever does not fit in the first case, definitely fits in the second, since we assumed that  $Mul_w(N)$  is polynomial in  $N$ . ◀

- **Corollary 5.** *Multiplying  $x$   $B$ -bit numbers costs polynomial time by using any polynomial time algorithm for multiplying two  $B$ -bit numbers as a black box. Especially if we use Harvey and Van Der Hoeven's algorithm, the time cost is  $\mathcal{O}(xB \log^2(xB))$ .*

Let us also notice that the way to divide two  $B$ -bit numbers is simply storing both the numerator and the denominator. Comparing two numbers  $x_1 = \frac{num_1}{den_1}$  and  $x_2 = \frac{num_2}{den_2}$  can be done by comparing  $num_1 \times den_2$  and  $num_2 \times den_1$ . The only other operation we need when working with such fractions is subtracting a  $B$ -bit number  $x = \frac{num}{den}$  from 1. This is simply  $\frac{den-num}{den}$ .

## 3 NP-Completeness

An NP-Completeness proof for the integer log-probability version of the WLCS problem has been given in [19]. This is a closely related problem, with the main difference being that products are replaced with sums. We do not know of any way to reduce from this log-probability version to WLCS other than exponentiating. As stated in the explanation of our model of computation in Section 2, there is no limit on the number of bits needed to represent a single number (it just occupies a lot of words). This means that, if the input consisted of  $I$  bits, and there was a number (probability) represented with  $\frac{I}{100}$  bits, exponentiating would result in a number with  $2^{\frac{I}{100}}$  bits, meaning the reduction would not

be a polynomial-time one. For this reason, we believe that although it is easier to prove NP-Completeness for the integer log-probability version of the problem, there is no easy way to use it for proving NP-Completeness for the general version. We, thus, give a reduction from the NP-Complete problem Subset Product [22] which proves NP-Completeness directly for the general problem.

Notice that for alphabets consisting of one letter, the problem is trivial since there is no uncertainty at all. In the following, we prove that even for alphabets consisting of two letters, the problem is NP-Complete.

► **Definition 6** (Subset Product). *Given a set  $L$  of  $n$  integers and an integer  $P$ , find if there exists a subset of the numbers in  $L$  with product  $P$ .*

► **Lemma 7.** *WLCS is NP-Complete, even for alphabets of size 2.*

**Proof.** Obviously  $WLCS \in NP$  since the increasing subsequences  $\pi_1, \pi_2$  and the string  $s$  for which  $P_X(\pi_1, s) \geq a, P_Y(\pi_2, s) \geq a$  are a certificate which, along with the input, can be used to verify in polynomial time that the problem has a solution.

Let  $(L, P)$  be an instance of Subset Product and let  $n = |L|$ . By  $L_i$  we denote the  $i$ -th number of the set  $L$ , assuming any fixed ordering of the  $n$  numbers of  $L$ . We give a polynomial-time reduction to a  $(X, Y, a, k)$  instance of WLCS, with alphabet size 2 (we call the letters ' $A$ ' and ' $B$ ').

The core idea is the following: The weighted sequences have  $n$  positions (plus 2 more for technical reasons related to the threshold  $a$ ). The number  $k$  is equal to the length of the sequences, meaning that we pick every position, and the only question is whether we picked letter ' $A$ ' or letter ' $B$ '. Letter ' $A$ ' in position  $i$  corresponds to picking the  $i$ -th number in the original Subset Product, while letter ' $B$ ' corresponds to not picking it. Finally, the letters ' $A$ ' picked in  $X$  form an inequality of the form: "some product is  $\geq P$ ", while the same letters in  $Y$  form the inequality: "the same product is  $\leq P$ ". For these two to hold simultaneously, it must be the case that we found some product equal to  $P$ , which is the goal of the original Subset Product.

More formally, the weighted sequences have size  $n + 2$ . Let  $c_i = \frac{1}{1+L_i}$  and  $d_i = \frac{1}{1+\frac{1}{L_i}}$ .

$$\begin{aligned} p_i^{(X)}('A') &= c_i L_i, 1 \leq i \leq n & p_i^{(Y)}('A') &= \frac{d_i}{L_i}, 1 \leq i \leq n \\ p_{n+1}^{(X)}('A') &= 1 & p_{n+1}^{(Y)}('A') &= \prod_{j=1}^n \frac{1}{L_j} = \frac{\prod_{j=1}^n c_j}{\prod_{j=1}^n d_j} \\ p_{n+2}^{(X)}('A') &= \frac{1}{P^2} & p_{n+2}^{(Y)}('A') &= 1 \end{aligned}$$

where  $p_i^{(X)}('B') = 1 - p_i^{(X)}('A')$  for all  $i$ , and similarly for  $Y$ . Notice that, in particular,  $p_i^{(X)}('B') = c_i, 1 \leq i \leq n$  and  $p_i^{(Y)}('B') = d_i, 1 \leq i \leq n$ . Finally, we set  $k = n + 2$  and  $a = \frac{\prod_{j=1}^n c_j}{P}$ .

First of all, notice that since we must find a string of length  $n + 2$ , we must choose a letter from every position. Thus, a choice of letter at some position on  $X$  corresponds to the same choice of letter at that position on  $Y$ . The choice of letter on positions  $n + 1$  and  $n + 2$  is ' $A$ ' in both cases since

$$p_{n+1}^{(X)}('B') = p_{n+2}^{(Y)}('B') = 0$$

Suppose that the numbers at positions  $\{i_1, \dots, i_\ell\}$  give product  $P$ :

$$\prod_{j=1}^{\ell} L_{i_j} = P$$

Then, we form the string  $s$  by picking 'A' at positions  $\{i_1, \dots, i_\ell, n+1, n+2\}$  and 'B' at all other positions. Thus

$$P_X(\{1, 2, \dots, n+2\}, s) = \frac{\prod_{j=1}^{\ell} L_{i_j} \prod_{j=1}^n c_i}{P^2} = \frac{\prod_{j=1}^n c_i}{P} = a$$

$$P_Y(\{1, 2, \dots, n+2\}, s) = \frac{\prod_{j=1}^n d_i \prod_{j=1}^n c_i}{\prod_{j=1}^{\ell} L_{i_j} \prod_{j=1}^n d_i} = \frac{\prod_{j=1}^n c_i}{P} = a$$

Conversely, suppose a solution for the WLCS problem, where the string  $s$  is formed by picking 'A' at positions  $\{i_1, \dots, i_\ell, n+1, n+2\}$  and 'B' at all other positions. It holds that:

$$P_X(\{1, 2, \dots, n+2\}, s) = \frac{\prod_{j=1}^{\ell} L_{i_j} \prod_{j=1}^n c_i}{P^2} \geq a \implies \prod_{j=1}^{\ell} L_{i_j} \geq P$$

$$P_Y(\{1, 2, \dots, n+2\}, s) = \frac{\prod_{j=1}^n d_i \prod_{j=1}^n c_i}{\prod_{j=1}^{\ell} L_{i_j} \prod_{j=1}^n d_i} \geq a \implies \prod_{j=1}^{\ell} L_{i_j} \leq P$$

The above imply that  $\prod_{j=1}^{\ell} L_{i_j} = P$ . Finally, notice that all computations are done in polynomial time, due to Corollary 5.  $\blacktriangleleft$

#### 4 EPTAS for Bounded Alphabets, Improved PTAS for Unbounded Alphabets

We now give an Efficient Polynomial Time Approximation Scheme (*EPTAS*) for the case where our alphabet size is bounded ( $|\Sigma| = O(1)$ ). Let us notice that this is the case when working with DNA sequences ( $|\Sigma| = 4$ ), the most usual application of weighted sequences. The same algorithm is an improved (when compared to [19]) *PTAS* in the case of unbounded alphabets. This means that the WLCS problem is Fixed-Parameter Tractable for constant size alphabets and thus belongs to the corresponding complexity class *FPT* as shown in Corollary 11.

The authors in [19] first noted that there is no *FPTAS* unless  $P = NP$ , and so we can only hope for an *EPTAS*. Our result relies on their following result:

► **Lemma 8** (Lemma 4.6 of [19]). *It is possible to find, in polynomial time, a solution of size  $d$  to the WLCS optimization problem such that the optimal value  $OPT$  is guaranteed to be either  $d$  or  $d+1$  (however we do not know which one holds).*

Their *PTAS* uses the above result and in case the approximation is guaranteed to be good enough ( $d > (1 - \epsilon)(d+1)$ , which implies that  $d > (1 - \epsilon)OPT$ ), it stops. Else, it holds that  $\frac{1}{\epsilon} \geq d+1 \geq OPT$ , and the *PTAS* exhaustively searches all subsequences of  $X$ , all subsequences of  $Y$ , and all possible strings of length  $d+1$ , for a total complexity of

$$\mathcal{O}\left(\text{Mul}_w\left(\frac{B}{\epsilon}\right) \log\left(\frac{B}{\epsilon}\right) |\Sigma|^{\frac{1}{\epsilon}} \binom{n}{\frac{1}{\epsilon}}^2\right)$$



$Mul_w(\frac{B}{\epsilon}) \log(\frac{B}{\epsilon})$  is the time needed to multiply  $d + 1$  numbers with at most  $B$ -bits each, by Lemma 4, and is insignificant compared to the other terms. Our *EPTAS* improves the exhaustive search part to

$$\mathcal{O}\left(Mul_w\left(\frac{B}{\epsilon}\right) \frac{n}{\epsilon} |\Sigma|^{\frac{1}{\epsilon}}\right)$$

which is polynomial in the input size, in case of bounded alphabets. The following lemma is needed.

► **Lemma 9.** *Given a weighted sequence  $X$  of length  $n$ , and a string  $s$  of length  $d$ , it is possible to find the maximum number  $a$  such that there exists an increasing subsequence  $\pi$  of length  $d$  for which  $P_X(\pi, s) = a$ . The running time of the algorithm is  $O(Mul_w(dB)nd)$ , where  $B$  is the maximum number of bits needed to represent each probability in  $X$ .*

**Proof.** We use dynamic programming. Let  $s_j$  be the string formed by the first  $j$  letters of  $s$ ,  $c_j$  be the  $j$ -th letter of  $s$  and  $opt_X(i, j)$  be the maximum number such that there exists an increasing subsequence  $\pi'$  of length  $j$  whose last term  $\pi'_j$  is at most  $i$  and for which  $P_X(\pi', s_j) = opt_X(i, j)$ . Since we choose whether  $c_j$  is picked from the  $i$ -th position of  $X$ , it holds that:

$$opt_X(i, j) = \max\{opt_X(i-1, j), opt_X(i-1, j-1)p_i^{(X)}(c_j)\}$$

For the base cases,  $opt_X(i, 0) = 1$  for all  $i$  (we can always form the empty string with certainty, by not picking anything), and  $opt_X(0, j) = 0$  for  $j > 0$  (not picking anything never gives us a non-empty string). We are interested in the value  $opt_X(|X|, |s|)$ . ◀

Now we are ready to give our *EPTAS*.

► **Theorem 10.** *For any value  $\epsilon \in (0, 1]$  there exists an  $(1 - \epsilon)$ -approximation algorithm for the *WLCS* problem which runs in  $\mathcal{O}\left(\text{poly}(I) + \frac{n}{\epsilon} Mul_w\left(\frac{B}{\epsilon}\right) |\Sigma|^{\frac{1}{\epsilon}}\right)$  time and uses  $\mathcal{O}(\text{poly}(I))$  space, where  $I$  is the input size,  $n = |X| + |Y|$  and  $B$  is the maximum number of bits needed to represent a probability in  $X$  and  $Y$ . Consequently, the *WLCS* problem admits an *EPTAS* for bounded alphabets.*

**Proof.** We begin by using Lemma 8 to find an  $a$ -subsequence of length  $d$ , such that the optimal solution is at most  $d + 1$ . If  $d + 1 \geq \frac{1}{\epsilon}$ , we are done, since in that case we have a  $\frac{d}{d+1} = 1 - \frac{1}{d+1} \geq (1 - \epsilon)$  approximation. Otherwise, we try all possible strings  $s \in |\Sigma|^{d+1}$ , and use Lemma 9 to check if any one of them can appear in both weighted sequences with probability at least  $a$ . ◀

► **Corollary 11.**  *$WLCS \in FPT$  for bounded alphabets, parameterized by the solution length.*

**Proof.** Follows directly from [30], Proposition 2. ◀

## 5 No *EPTAS* for Unbounded Alphabets

We have already seen that there is no *FPTAS* for *WLCS*, even for alphabets of size 2, unless  $P = NP$ . We have also shown an *EPTAS* for bounded alphabets and a *PTAS* for unbounded alphabets. The natural question that arises is: Is it possible to give an *EPTAS* for unbounded alphabets?

We answer this question negatively, by proving that *WLCS* is  $W[1]$ -hard, meaning that it does not admit an *EPTAS* (and is in fact not even in *FPT*) unless  $FPT = W[1]$

([30], Corollary 1). To show this, we give a 2-step *FPT*-reduction from Perfect Code, which was shown to be  $W[1]$ -Complete in [15], to  $k$ -sized Subset Product and then to WLCS. The  $k$ -sized Subset Product problem is the Subset Product problem with the additional constraint that the target subset must be of size  $k$ .

► **Definition 12** (Perfect Code). *A perfect code is a set of vertices  $V' \subseteq V$  with the property that for each vertex  $u \in V$  there is precisely one vertex in  $N_G(u) \cap V'$ , where  $N_G(u)$  is the set of adjacent nodes of  $u$  in  $G$ .*

In the perfect code problem, we are given an undirected graph  $G$  and a positive integer  $k$ , and we need to decide whether  $G$  has a  $k$ -element perfect code. Notice that the definition of a perfect code implies that there is a perfect code iff there is a set  $V' \subseteq V$  for which  $\bigcup_{u \in V'} N_G(u) = V$  and  $N_G(u) \cap N_G(v) = \emptyset$  for all  $u, v \in V', u \neq v$ . First we show that  $k$ -sized Subset Product is  $W[1]$ -hard.

► **Lemma 13.**  *$k$ -sized Subset Product is  $W[1]$ -hard.*

**Proof.** Let  $(G = (V, E), k)$  be an instance of Perfect Code. Suppose that the vertices are  $V = \{1, \dots, n\}$ . First of all, we compute the first  $n$  prime numbers using the Sieve of Eratosthenes. We denote the  $i$ -th prime number as  $p_i$ . The set of positive integers  $L = \{L_1, L_2, \dots, L_n\}$  as well as the positive integer  $P$  are defined as follows:

$$L_v = \prod_{u \in N_G(v)} p_u, \quad P = \prod_{v=1}^n p_v$$

Notice that due to the unique prime factorization theorem, a subset of  $k$  numbers from the set  $L$  have product  $P$  iff  $G$  has a  $k$ -element Perfect Code.

The size of our primes is  $O(n \log n)$  due to the prime number theorem. Thus, they require  $O(\log n)$  bits to be represented. Each integer in  $L$ , as well as in  $P$ , is computed using Corollary 5 in  $O(n \log^3 n)$  time, for an overall  $O(n^2 \log^3 n)$  complexity for our reduction. Since the new parameter  $k$  is the same as the old one (no dependence on  $n$ ), our reduction is in fact an *FPT*-reduction. ◀

Our result for this section is the following.

► **Theorem 14.** *WLCS, parameterized by the length of the solution, is  $W[1]$ -hard.*

**Proof.** To prove the theorem we create diagonal weighted sequences. That is, we require each letter to appear only in one position and vice-versa. In this way, the subsequences picked for  $X$  and  $Y$  are the same. The above rule is broken by the addition of two auxiliary letters that are there to make the probabilities add up to 1 in each position. This creates no problem because we make sure that these letters are never picked. Finally, we force the product to be equal to our target, by forcing it to be at most our target and at least our target at the same time.

More formally, let  $(L = \{L_1, L_2, \dots, L_n\}, k, P)$  be an instance of the  $k$ -sized Subset Product problem and let  $M = m^{k+1}$ , where  $m$  is the maximum number in set  $L$ . Notice that if  $m^k \leq P$  then we only need to check the product of the highest  $k$  numbers of  $L$ , which means the problem is solvable in polynomial time. Thus we can assume that  $M \geq m^k > P$ .

The alphabet of  $X, Y$  is  $\Sigma = \{1, 2, \dots, n, n+1, n+2, n+3\}$  and we set  $a = \frac{1}{PM^k}$ .

$$\begin{aligned} p_i^{(X)}(i) &= \frac{L_i}{M}, \quad 1 \leq i \leq n & p_i^{(Y)}(i) &= \frac{1}{ML_i}, \quad 1 \leq i \leq n \\ p_{n+1}^{(X)}(n+1) &= \frac{1}{P^2} & p_{n+1}^{(Y)}(n+1) &= 1 \\ p_i^{(X)}(n+2) &= 1 - p_i^{(X)}(i), \quad 1 \leq i \leq n+1 & p_i^{(Y)}(n+3) &= 1 - p_i^{(Y)}(i), \quad 1 \leq i \leq n+1 \end{aligned}$$

All non-specified probabilities are equal to 0. Notice that symbols  $n+2$  and  $n+3$  are used to guarantee that probabilities sum up to 1.

We show that the instance  $(X, Y, a, k+1)$  has a solution iff  $(L, k, P)$  has a solution. Suppose there exists a solution to  $(L, k, P)$ . Then, there exists an increasing subsequence  $\pi = (i_1, \dots, i_k)$  such that  $\prod_{j=1}^k L_{i_j} = P$ . Let  $\pi'$  be  $\pi$  extended by the number  $i_{k+1} = n+1$  and  $s$  be the string  $i_1 i_2 \dots i_{k+1}$ . It holds that  $P_X(\pi', s) = P_Y(\pi', s) = a$ .

Conversely, suppose there exists a solution to  $(X, Y, a, k+1)$ . Then there exist increasing subsequences  $\pi = (i_1, \dots, i_{k+1}), \pi' = (j_1, \dots, j_{k+1})$  and a string  $s$  such that  $P_X(\pi, s) \geq a, P_Y(\pi', s) \geq a$ . First of all, notice that, due to  $p_i^{(X)}(n+3) = p_i^{(Y)}(n+2) = 0$  for all  $i$ ,  $s$  does not contain letters  $n+2$  and  $n+3$ , which leaves only one choice for every position. Also each letter appears only once in each sequence, and in the same position. Thus,  $\pi = \pi'$ , and due to our construction the  $i$ -th letter of  $s$  is the  $i$ -th member of  $\pi$ . Finally, not picking position  $n+1$  would result in  $P_Y(\pi, s) < a$  due to the fact that  $P < M$ . Thus, the last letter of  $s$  is  $n+1$ . It holds that:

$$\begin{aligned} P_X(\{i_1, \dots, i_{k+1}\}, s) \geq a &\implies \frac{\prod_{i=1}^k L_{\pi_i}}{P^2 M^k} \geq \frac{1}{PM^k} \implies \prod_{i=1}^k L_{\pi_i} \geq P \\ P_Y(\{i_1, \dots, i_{k+1}\}, s) \geq a &\implies \frac{1}{M^k \prod_{i=1}^k L_{\pi_i}} \geq \frac{1}{PM^k} \implies \prod_{i=1}^k L_{\pi_i} \leq P \end{aligned}$$

The above two inequalities imply a  $k$ -sized subset of  $L$  with product equal to  $P$ .

The reduction is a polynomial-time one, due to Corollary 5. More than that, it is an *FPT*-reduction since the new parameter  $k$  is equal to the old parameter incremented by one, and thus has no dependence on  $n$ .  $\blacktriangleleft$

## 6 Matching Conditional Lower Bound on any PTAS

In the  $d$ -SUM problem, we are given  $N$  numbers and need to decide whether there exists a  $d$ -tuple that sums to zero. Patrascu and Williams [32] proved that any algorithm for solving the  $d$ -SUM problem requires  $n^{\Omega(d)}$  time, unless the Exponential Time Hypothesis (*ETH*) fails. To show this, they first proved a hardness result for a variant of 3-SAT, the sparse 1-in-3 SAT.

► **Definition 15** (Sparse 1-in-3 SAT). *Given a boolean formula with  $n$  variables and  $O(n)$  clauses in 3 CNF form, where each variable appears in a constant number of clauses, determine whether there exists an assignment of the variables such that each clause is satisfied by exactly one variable.*

They first prove the following hardness result under *ETH*.

► **Proposition 16.** *Under ETH, there is an (unknown) constant  $s_3$  such that there exists no algorithm to solve sparse 1-in-3 SAT in  $\mathcal{O}(2^{\delta n})$  time for  $\delta < s_3$ .*

By assuming an  $n^{\mathcal{O}(d)}$  time algorithm for  $d$ -SUM they disproved the above fact, which cannot happen under *ETH*. We use the same technique for proving an  $n^{\Omega(k)}$  lower bound for  $k$ -sized Subset Product.

► **Lemma 17.** *Assuming the ETH, the problem of  $k$ -sized Subset Product cannot be solved in  $\mathcal{O}(n^{\frac{s_3 k}{101}})$  time on instances satisfying  $k < n^{0.99}$  and each number in the input set  $L$  has  $\mathcal{O}(\log n(\log k + \log \log n))$  bits, where  $n$  is the size of  $L$ , and  $P$  is the target which can be arbitrarily big.*

**Proof.** Let  $f$  be a sparse 1-in-3 SAT instance with  $N$  variables and  $M = \mathcal{O}(N)$  clauses, and  $k > \frac{1}{s_3}$ . Conceptually, we split the variables of  $f$  into  $k$  blocks of equal size - apart from the last block that may have smaller size. Each block contains at most  $\lceil \frac{N}{k} \rceil$  variables, and thus there are at most  $2^{\lceil \frac{N}{k} \rceil}$  different assignments of values to the group-of-variables within a block. For each block and for each one of these assignments we generate a number which serves as an identifier of the corresponding block and assignment. Thus, there are  $n = k2^{\lceil \frac{N}{k} \rceil}$  different identifiers.

Let  $p_i$  be the  $i$ -th prime number. In order to compute an identifier, we initialize it to  $p_b$ , where  $b$  is the index of the identifier's corresponding block. Then, we run through all of the  $M = \mathcal{O}(N)$  clauses and do the following: suppose we process the  $i$ -th clause and let  $0 \leq j \leq 3$  be the number of variables of the identifier's corresponding assignment that satisfy the clause. We update the identifier by multiplying it with  $p_{k+i}^j$ .

Since each variable appears only in a constant number of clauses, each identifier is a product of  $\mathcal{O}(\frac{N}{k})$  numbers. The prime number theorem guarantees  $\mathcal{O}(\log N)$  bits to represent each factor, which means the identifiers have  $\mathcal{O}(\frac{N}{k} \log N)$  bits. Using the fact that  $n = k2^{\lceil \frac{N}{k} \rceil}$ , each identifier is represented by  $\mathcal{O}(\log n(\log k + \log \log n))$  bits.

These  $n$  identifiers, along with the target  $P = \prod_{i=1}^{k+M} p_i$  (recall that  $p_i$  is the  $i$ -th prime number), form a  $k$ -sized Subset Product instance. This preprocessing step costs  $\mathcal{O}(2^{\frac{N}{k}})$  time, ignoring polynomial terms, which is more efficient than  $\mathcal{O}(2^{s_3 N})$ .

Due to the unique prime factorization, a solution to the  $k$ -sized Subset Product corresponds to a solution in  $f$  and vice-versa. If the running time of the  $k$ -sized Subset Product was  $\mathcal{O}(n^{\frac{s_3 k}{101}})$  then we could solve the above instance in  $\mathcal{O}((k2^{\frac{N}{k}})^{\frac{s_3 k}{101}})$  time.

Since  $k = \frac{n}{2^{\lceil \frac{N}{k} \rceil}}$  and  $k < n^{0.99}$ , it follows that  $\frac{n}{2^{\lceil \frac{N}{k} \rceil}} < n^{0.99} \implies n^{0.99} < 2^{99 \lceil \frac{N}{k} \rceil}$ . But  $k < n^{0.99}$ , which means  $k < 2^{99 \lceil \frac{N}{k} \rceil}$ .

Thus the previous running time becomes  $\mathcal{O}(2^{\frac{100}{101} s_3 N})$ . Both the preprocessing step and the solution of the  $k$ -sized Subset Product can be achieved in time  $\mathcal{O}(2^{\delta N})$ , where  $\delta < s_3$ . However, this would violate Proposition 16. ◀

Using the above, we are ready to prove our (matching) lower bound, conditional on *ETH*.

► **Theorem 18.** *Under ETH, there is no PTAS for WLCS with running time  $|I|^{\mathcal{O}(\frac{1}{\epsilon})}$ , where  $|I|$  is the input size in bits.*

**Proof.** Suppose that such an algorithm  $A(I, \epsilon)$  existed. Let  $R()$  be the polynomial time reduction from  $k$ -sized Subset Product to WLCS given in the proof of Theorem 14. Then, there is a solution to  $k$ -sized Subset Product iff there is a solution to WLCS of size  $k + 1$ , or, equivalently, iff the optimal solution to WLCS is at least  $k + 1$ .

Using the hypothetical  $A(I, \epsilon)$  with an appropriate value of  $\epsilon$ , we solve  $k$ -sized Subset Product more efficiently than possible, thus reaching a contradiction.

Consider the following algorithm for  $k$ -sized Subset Product, where there are  $|L|$  numbers in the input, each having  $\mathcal{O}(\log |L|(\log k + \log \log |L|))$  bits and  $k < |L|^{0.99}$ . Given an

instance  $(L, k, P)$ , we define the instance for the WLCS to be  $I = R(L, k, P)$ . We run  $A(I, \frac{1}{2^{(k+1)}})$  and if the output is at least  $k + 1$  we return that  $(L, k, P)$  is satisfied, otherwise we return that it cannot be satisfied.

Note that if  $k$ -sized Subset Product is solvable, then  $OPT(I) \geq k + 1$ , and the value output by  $A$  is at least  $(1 - \frac{1}{2^{(k+1)}})(k + 1) = k + \frac{1}{2} > k$ . Thus, the value output by  $A$  is at least  $k + 1$ . On the other hand, if  $k$ -sized Subset Product is not solvable, then  $OPT(I) < k + 1$ , and obviously the value output by  $A$  is at most  $k$ .

Thus we found an algorithm for  $k$ -sized Subset Product whose running time is  $|I|^{o(k)}$ . Since  $I$  is obtained by a polynomial time reduction, its size is bounded by a polynomial in  $|(L, k, P)|$ . Therefore, the above running time becomes  $|(L, k, P)|^{o(k)}$ . Under our assumptions, this becomes  $|L|^{o(k)}$ , which is not feasible under  $ETH$ , due to Lemma 17. ◀

## 7 Conclusion

In this paper we prove NP-Completeness for the WLCS decision problem, and give a  $PTAS$  along with a matching conditional lower bound for the optimization problem. In the most usual setting, where the alphabet size is constant, the above  $PTAS$  is in fact an  $EPTAS$ , and it is known that no  $FPTAS$  can exist unless  $P = NP$ . In the Appendix we give a transformation such that algorithms for the WLCS problem can also be applied for the  $(a_1, a_2)$ -WLCS problem.

In proving that WLCS does not admit any  $EPTAS$ , we proved that it is  $W[1]$  – hard. It may be interesting to determine the exact complexity of WLCS in the  $W$  – hierarchy.

---

## References

- 1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.
- 2 Amihod Amir, Eran Chencinski, Costas S. Iliopoulos, Tsvi Kopelowitz, and Hui Zhang. Property matching and weighted matching. *Theoretical Computer Science*, 395(2-3):298–310, 2008. doi:10.1016/j.tcs.2008.01.006.
- 3 Amihod Amir, Zvi Gotthilf, and B. Riva Shalom. Weighted LCS. *Journal of Discrete Algorithms*, 8(3):273–281, 2010. doi:10.1016/j.jda.2010.02.001.
- 4 Amihod Amir, Zvi Gotthilf, and B. Riva Shalom. Weighted shortest common supersequence. In *String Processing and Information Retrieval, 18th International Symposium, SPIRE 2011, Pisa, Italy, October 17-21, 2011. Proceedings*, pages 44–54, 2011. doi:10.1007/978-3-642-24583-1\\_6.
- 5 Amihod Amir, Tzvika Hartman, Oren Kapah, B. Riva Shalom, and Dekel Tsur. Generalized LCS. *Theoretical Computer Science*, 409(3):438–449, 2008. doi:10.1016/j.tcs.2008.08.037.
- 6 Amihod Amir, Costas S. Iliopoulos, Oren Kapah, and Ely Porat. Approximate matching in weighted sequences. In *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, pages 365–376, 2006. doi:10.1007/11780441\\_33.
- 7 Pavlos Antoniou, Costas S. Iliopoulos, Laurent Mouchard, and Solon P. Pissis. Algorithms for mapping short degenerate and weighted sequences to a reference genome. *International Journal of Computational Biology and Drug Design*, 2(4):385–397, 2009. doi:10.1504/IJCDD.2009.030768.
- 8 Alberto Apostolico, Gad M. Landau, and Steven Skiena. Matching for run-length encoded strings. *Journal of Complexity*, 15(1):4–16, 1999. doi:10.1006/jcom.1998.0493.

- 9 Carl Barton, Costas S. Iliopoulos, and Solon P. Pissis. Optimal computation of all tandem repeats in a weighted sequence. *Algorithms for Molecular Biology*, 9:21, 2014. doi:10.1186/s13015-014-0021-5.
- 10 Carl Barton, Tomasz Kociumaka, Chang Liu, Solon P. Pissis, and Jakub Radoszewski. Indexing weighted sequences: Neat and efficient. *Information and Computation*, 270, 2020. doi:10.1016/j.ic.2019.104462.
- 11 Carl Barton and Solon P. Pissis. Crochemore’s partitioning on weighted strings and applications. *Algorithmica*, 80(2):496–514, 2018. doi:10.1007/s00453-016-0266-0.
- 12 Jon Louis Bentley, Dorothea Haken, and James B. Saxe. A general method for solving divide-and-conquer recurrences. *SIGACT News*, 12(3):36–44, September 1980. doi:10.1145/1008861.1008865.
- 13 Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *Seventh International Symposium on String Processing and Information Retrieval, SPIRE 2000, A Coruña, Spain, September 27-29, 2000*, pages 39–48, 2000. doi:10.1109/SPIRE.2000.878178.
- 14 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1216–1235, 2018. doi:10.1137/1.9781611975031.79.
- 15 Marco Cesati. Perfect code is W[1]-complete. *Information Processing Letters*, 81(3):163–168, 2002. doi:10.1016/S0020-0190(01)00207-1.
- 16 Panagiotis Charalampopoulos, Costas S. Iliopoulos, Solon P. Pissis, and Jakub Radoszewski. On-line weighted pattern matching. *Information and Computation*, 266:49–59, 2019. doi:10.1016/j.ic.2019.01.001.
- 17 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba. Weighted shortest common supersequence problem revisited. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings*, volume 11811 of *Lecture Notes in Computer Science*, pages 221–238. Springer, 2019. doi:10.1007/978-3-030-32686-9\_16.
- 18 Manolis Christodoulakis, Costas S. Iliopoulos, Laurent Mouchard, Katerina Perdikuri, Athanasios K. Tsakalidis, and Kostas Tsichlas. Computation of repetitions and regularities of biologically weighted sequences. *Journal of Computational Biology*, 13(6):1214–1231, 2006. doi:10.1089/cmb.2006.13.1214.
- 19 Marek Cygan, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Polynomial-time approximation algorithms for weighted LCS problem. *Discrete Applied Mathematics*, 204:38–48, 2016. doi:10.1016/j.dam.2015.11.011.
- 20 Michael L. Fredman and Dan E. Willard. BLASTING through the information theoretic barrier with FUSION TREES. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 1–7, 1990. doi:10.1145/100216.100217.
- 21 Martin Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009. doi:10.1137/070711761.
- 22 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 23 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.
- 24 David Harvey and Joris Van Der Hoeven. Integer multiplication in time  $O(n \log n)$ . working paper or preprint, March 2019. URL: <https://hal.archives-ouvertes.fr/hal-02070778>.
- 25 Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975. doi:10.1145/360825.360861.



- 26 Costas S. Iliopoulos, Christos Makris, Yannis Panagis, Katerina Perdikuri, Evangelos Theodoridis, and Athanasios K. Tsakalidis. Efficient algorithms for handling molecular weighted sequences. In *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004), 22-27 August 2004, Toulouse, France*, pages 265–278, 2004. doi:10.1007/1-4020-8141-3\22.
- 27 Costas S. Iliopoulos, Mirka Miller, and Solon P. Pissis. Parallel algorithms for mapping short degenerate and weighted DNA sequences to a reference genome. *International Journal of Foundations of Computer Science*, 23(2):249–259, 2012. doi:10.1142/S0129054112400114.
- 28 Costas S. Iliopoulos, Katerina Perdikuri, Evangelos Theodoridis, Athanasios K. Tsakalidis, and Kostas Tsichlas. Motif extraction from weighted sequences. In *String Processing and Information Retrieval, 11th International Conference, SPIRE 2004, Padova, Italy, October 5-8, 2004, Proceedings*, pages 286–297, 2004. doi:10.1007/978-3-540-30213-1\41.
- 29 Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Pattern matching and consensus problems on weighted sequences and profiles. *Theory of Computing Systems*, 63(3):506–542, 2019. doi:10.1007/s00224-018-9881-2.
- 30 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.
- 31 Gene Myers. The whole genome assembly of drosophila. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, page 753, 2000. URL: <http://dl.acm.org/citation.cfm?id=338219.338635>.
- 32 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075, 2010. doi:10.1137/1.9781611973075.86.
- 33 Jakub Radoszewski and Tatiana Starikovskaya. Streaming  $k$ -mismatch with error correcting and applications. *Information and Computation*, 271:104513, 2020. doi:10.1016/j.ic.2019.104513.
- 34 Arnold Schönhage and Volker Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971. doi:10.1007/BF02242355.
- 35 Julie Thompson, Desmond G. Higgins, and Toby J. Gibson. W: Clustal. improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic acids research*, 22:4673–80, 12 1994. doi:10.1093/nar/22.22.4673.
- 36 J. Craig Venter. Sequencing the human genome. In *RECOMB*, pages 309–309. ACM, 2002.
- 37 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.
- 38 Hui Zhang, Qing Guo, Jing Fan, and Costas S. Iliopoulos. Loose and strict repeats in weighted sequences of proteins. *Protein and Peptide Letters*, 17(9):1136–1142, 2010.
- 39 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. String matching with swaps in a weighted sequence. In *CIS*, volume 3314 of *Lecture Notes in Computer Science*, pages 698–704. Springer, 2004.
- 40 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. An algorithmic framework for motif discovery problems in weighted sequences. In *Algorithms and Complexity, 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings*, pages 335–346, 2010. doi:10.1007/978-3-642-13073-1\30.
- 41 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. Varieties of regularities in weighted sequences. In *AAIM*, volume 6124 of *Lecture Notes in Computer Science*, pages 271–280. Springer, 2010.
- 42 Hui Zhang, Qing Guo, and Costas S. Iliopoulos. Locating tandem repeats in weighted sequences in proteins. *BMC Bioinformatics*, 14(S-8):S2, 2013.

## A One Threshold is Enough

For clarity purposes, some proofs and technical discussions are moved in this appendix. In particular, in this section we show that  $(a_1, a_2)$ -WLCS and WLCS are equivalent, thus one threshold is enough. Furthermore, we show that the rounding errors introduced by working with reals (logarithms and roots) may cause a similar algorithm from a paper by Cygan et al. [19] to err if standard rounding is used.

In the following,  $B$  corresponds to the maximum number of bits to represent a number in the input (a probability or a symbol of the alphabet).  $B$  is not to be confused with the word-size  $w$  since an input number may need many words to be represented.

► **Lemma 19.** *Given an instance  $(X, Y, a_1, a_2, k)$  of  $(a_1, a_2)$ -WLCS ( $a_1 < a_2$ ), it is possible to reduce it to an instance  $(X', Y', a, k + 1)$  of WLCS. The construction of  $X'$  and  $Y'$  requires  $\mathcal{O}(n|\Sigma|Mul_w(B))$  time, while parameter  $a$  is computed in  $\mathcal{O}(Mul_w(nB) \log(nB))$  time, where  $n = |X| + |Y|$  is the total length of the weighted sequences  $X$  and  $Y$ , while  $B$  is the maximum number of bits needed to represent an input number.*

**Proof.** We first provide a sketch of the proof. Our goal is to use the same weighted sequences with one additional position at the end. We introduce a new letter ( $'\%'$ ) which only appears in this position, and we make sure that any correct algorithm picks it, by making its probability very appealing (high). Since we cannot assign a probability higher than one, increasing it is simulated by reducing all other probabilities, in all positions. Knowing that this specific letter is picked at this specific position allows us to choose the two corresponding probabilities in a way that completes the proof. In order for the probabilities to sum to 1 in every position, we introduce two auxiliary letters ( $'\#'$  and  $'\$'$ ) that are never picked ( $'\$'$  never appears on the first weighted sequence,  $'\#'$  never appears on the second).

The alphabet  $\Sigma'$  of  $X', Y'$  is the alphabet  $\Sigma$  of  $X, Y$  extended by three new letters,  $\Sigma' = \Sigma \cup \{'\#', '\$', '\%'\}$ . Let  $m = \frac{a_1}{a_2}$  and  $a = m^k a_1$ . Notice that since  $k \leq n$ , the size of  $a$  in bits is only polynomial compared to the input size, not exponential. The new sequences  $X'$  and  $Y'$  are constructed as follows:

$$\begin{aligned} p_i^{(X')}(\sigma) &= mp_i^{(X)}(\sigma), 1 \leq i \leq |X|, \sigma \in \Sigma & p_i^{(Y')}(\sigma) &= mp_i^{(Y)}(\sigma), 1 \leq i \leq |Y|, \sigma \in \Sigma \\ p_i^{(X')}(' \# ') &= 1 - \sum_{\sigma \in \Sigma \setminus \{'\#'\}} p_i^{(X')}(\sigma), \forall i & p_i^{(Y')}(' \$ ') &= 1 - \sum_{\sigma \in \Sigma \setminus \{'\$\''\}} p_i^{(Y')}(\sigma), \forall i \\ p_{|X|+1}^{(X')}(' \% ') &= 1 & p_{|Y|+1}^{(Y')}(' \% ') &= \frac{a_1}{a_2} \end{aligned}$$

All non-specified probabilities are equal to 0.

If there exists a solution to  $(X, Y, a_1, a_2, k)$ , then there exist two increasing subsequences  $\pi_1 = (i_1, \dots, i_k), \pi_2 = (j_1, \dots, j_k)$  and a string  $s$  such that  $P_X(\pi_1, s) \geq a_1, P_Y(\pi_2, s) \geq a_2$ . Define  $\pi'_1 = (i_1, \dots, i_k, |X| + 1), \pi'_2 = (j_1, \dots, j_k, |Y| + 1)$  and  $s'$  to be equal to  $s$  extended with the letter  $'\%'$ . It holds that:

$$P_{X'}(\pi'_1, s') = m^k P_X(\pi_1, s) \geq m^k a_1 = a, P_{Y'}(\pi'_2, s') = m^k P_Y(\pi_2, s) \frac{a_1}{a_2} \geq m^k a_2 \frac{a_1}{a_2} = a$$

Conversely, suppose there exists a solution to  $(X', Y', a, k + 1)$ . Then, there exist increasing subsequences  $\pi_1 = (i_1, \dots, i_{k+1}), \pi_2 = (j_1, \dots, j_{k+1})$  and a string  $s$  such that  $P_{X'}(\pi_1, s) \geq a, P_{Y'}(\pi_2, s) \geq a$ . First of all, notice that, due to  $p_i^{(X')}(' \$ ') = p_i^{(Y')}(' \# ') = 0$  for all  $i$ ,  $s$  does not contain letters  $'\$'$  and  $'\#'$ . In addition, the letter  $'\%'$  only appears at the last position, and it is the only possible option for this position. Finally, the last position shall be used on both subsequences, because otherwise  $P_{X'}(\pi_1, s), P_{Y'}(\pi_2, s) \leq m^{k+1} < a$ . Thus, the last letter of  $s$  is  $'\%'$ . If we denote by  $s'$  the string  $s$  without its last letter, it holds that  $P_X(\{i_1, \dots, i_k\}, s') \geq a_1, P_Y(\{j_1, \dots, j_k\}, s') \geq a_2$ .



The computation of  $a$  requires  $\mathcal{O}(Mul_w(nB) \log(nB))$  time due to Corollary 5, and the  $n|\Sigma|$ -multiplications of two numbers with at most  $B$  bits each cost  $\mathcal{O}(n|\Sigma|Mul_w(B))$ . All other computations take linear time.  $\blacktriangleleft$

We note that [19] proved the same result, but their reduction required computations with real numbers (raising to the  $\log_{a_2} a_1$  power). To the best of our knowledge, there is no way to modify that reduction so that it tolerates the rounding error in the word  $RAM$  introduced by working with roots and logarithms.

In what follows, we show that the rounding errors may cause the algorithm by Cygan et al. [19], which reduces any instance of  $WLCS$  to a more restricted class of instances, to err. This does not rule out the possibility that more clever rounding algorithms (depending on the input size) may indeed be used so that the algorithm does not err; however we are not aware of any such rounding technique, and even if it exists, the algorithm would probably become too complicated compared to ours.

► **Lemma 20.** *The reduction from  $(a_1, a_2)$ - $WLCS$  to  $WLCS$  with only one threshold given by Cygan et al. in [19] may err, if exact computations with logarithms and roots are not assumed (assuming the rounding technique does not depend on the input, for example it only keeps a constant number of decimal digits).*

**Proof.** We prove the above with an example that demonstrates that the rounding error, introduced by not assuming exact computations with logarithms and roots, may cause the reduction to err.

Let  $a_1 = \frac{1}{8}, a_2 = \frac{1}{4}$  and the two weighted sequences  $X$  and  $Y$  on alphabet  $\Sigma = \{a, b\}$  be:

|     |   |   |   |               |
|-----|---|---|---|---------------|
| $X$ | 1 | 2 | 3 | 4             |
| a   | 1 | 1 | 1 | $\frac{1}{8}$ |
| b   | 0 | 0 | 0 | $\frac{7}{8}$ |

|     |       |               |               |   |
|-----|-------|---------------|---------------|---|
| $Y$ | 1     | 2             | 3             | 4 |
| a   | $x$   | $\frac{1}{2}$ | $\frac{1}{2}$ | 1 |
| b   | $1-x$ | $\frac{1}{2}$ | $\frac{1}{2}$ | 0 |

where  $0 \leq x \leq 1$  is a constant to be specified later. For  $x = 1$ , the weighted  $LCS$  is  $aaaa$  and for  $x < 1$  the weighted  $LCS$  is  $aaa$ . The transformation described in [19] would give  $a = \frac{1}{8}, \gamma = \frac{3}{2}$  and the new sequences would be:

|      |   |   |   |               |
|------|---|---|---|---------------|
| $X'$ | 1 | 2 | 3 | 4             |
| a    | 1 | 1 | 1 | $\frac{1}{8}$ |
| b    | 0 | 0 | 0 | $\frac{7}{8}$ |
| #    | 0 | 0 | 0 | 0             |

|      |                             |                            |                            |   |
|------|-----------------------------|----------------------------|----------------------------|---|
| $Y'$ | 1                           | 2                          | 3                          | 4 |
| a    | $x^\gamma$                  | $\frac{1}{2}^\gamma$       | $\frac{1}{2}^\gamma$       | 1 |
| b    | $(1-x)^\gamma$              | $\frac{1}{2}^\gamma$       | $\frac{1}{2}^\gamma$       | 0 |
| #    | $1-x^\gamma - (1-x)^\gamma$ | $1-2 * \frac{1}{2}^\gamma$ | $1-2 * \frac{1}{2}^\gamma$ | 0 |

Since  $\frac{1}{2}^\gamma$  is an irrational number, it is rounded to some number  $r = \lfloor \frac{1}{2}^\gamma \rfloor$ . Suppose  $r < \frac{1}{2}^\gamma$ . In this case, when  $x = 1$ , while the weighted  $LCS$  is  $aaaa$  the algorithm returns  $aaa$  due to the rounding errors. On the other hand, if  $r > \frac{1}{2}^\gamma$ , we can always find an appropriate  $x < 1$  such that the weighted  $LCS$  should have been  $aaa$  but the algorithm returns  $aaaa$  due to the rounding errors. To show this, let  $x = \left(\frac{k-1}{k}\right)^2$  for some integer  $k$ . Then  $x^\gamma = \left(\frac{k-1}{k}\right)^3$ . It holds that  $\left(\frac{k-1}{k}\right)^3 r^2$  is an increasing function of  $k$  which converges to  $r^2 > \frac{1}{8}$ . Thus, we can find a big enough  $k$  such that  $x^\gamma r^2 \geq \frac{1}{8}$  and err on this particular example, as long as the rounding technique does not depend on the input (for example it only keeps a constant number of decimal digits).  $\blacktriangleleft$

Once again, the above is not a proof that the algorithm given by Cygan et al. can never be correct, despite of the rounding algorithm used. It just shows that it is necessary to explicitly specify such a rounding algorithm in order to construct a correct algorithm.