Graph Algorithms: Network Inference and Planar Graph Optimization

Hang Zhou

To cite this version:

Hang Zhou. Graph Algorithms: Network Inference and Planar Graph Optimization. Computational Geometry [cs.CG]. Ecole Normale Superieure, 2015. English. NNT : . tel-01174514v1

HAL Id: tel-01174514
https://theses.hal.science/tel-01174514v1

Submitted on 9 Jul 2015 (v1), last revised 20 Apr 2018 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE DE DOCTORAT

présentée et soutenue publiquement
le 6 juillet 2015
en vue de l'obtention du grade de

Docteur de l'École normale supérieure

Spécialité : Informatique

par

Hang ZHOU

Graph Algorithms: Network Inference and Planar Graph Optimization

Membres du jury :

M. Cyril Gavoille (Université de Bordeaux, France)	rapporteur
M. Frédéric Havet (CNRS, INRIA Sophia-Antipolis, France)	examinateur
M. Philip N. Klein (Brown University, USA)	examinateur
M. Marc Lelarge (INRIA, ENS Paris, France)	examinateur
Mme Claire Mathieu (CNRS, ENS Paris, France)	directrice
M. Christophe Paul (CNRS, LIRMM, France)	examinateur
M. Stéphan Thomassé (ENS Lyon, France)	examinateur

Autre rapporteur :

M. Artur Czumaj (University of Warwick, UK)

Unité mixte de recherche 8548 : Département d'Informatique de l'École normale supérieure
École doctorale 386 : Sciences mathématiques de Paris Centre

Résumé

Cette thèse porte sur deux sujets d'algorithmique des graphes.

Le premier sujet est l'inférence de réseaux. Quelle est la complexité pour déterminer un graphe inconnu à partir de requêtes de plus court chemin entre ses sommets ? Nous supposons que le graphe est de degré borné. Dans le problème de *reconstruction*, le but est de reconstruire le graphe ; tandis que dans le problème de *vérification*, le but est de vérifier qu'un graphe donné est correct. Nous développons des algorithmes probabilistes utilisant une décomposition en cellules de Voronoi. Ensuite, nous analysons des algorithmes de type glouton, et montrons qu'ils sont quasi-optimaux. Nous étudions aussi ces problèmes sur des familles particulières de graphes, démontrons des bornes inférieures, et étudions la reconstruction approximative.

Le deuxième sujet est l'étude de deux problèmes d'optimisation sur les graphes planaires. Dans le problème de *classification par corrélations*, l'entrée est un graphe pondéré, où chaque arête a une étiquette $\langle + \rangle$ ou $\langle - \rangle$, indiquant si ses extrémités sont ou non dans la même catégorie. Le but est de trouver une partition des sommets en catégories qui respecte au mieux les étiquettes. Dans le problème d'*augmentation 2-arête-connexe*, l'entrée est un graphe pondéré et un sous-ensemble R des arêtes. Le but est de trouver un sous-ensemble S des arêtes de poids minimum, tel que pour chaque arête de R, ses extrémités sont dans une composante 2-arête-connexe de l'union de R et S. Pour les *graphes planaires*, nous réduisons le premier problème au deuxième et montrons que les deux problèmes, bien que NP-durs, ont un schéma d'approximation en temps polynomial. Nous utilisons la technique récente de décomposition en briques.

Abstract

This thesis focuses on two topics of graph algorithms.

The first topic is network inference. How efficiently can we find an unknown graph using shortest path queries between its vertices? We assume that the graph has bounded degree. In the *reconstruction* problem, the goal is to find the graph; and in the *verification* problem, the goal is to check whether a given graph is correct. We provide randomized algorithms based on a Voronoi cell decomposition. Next, we analyze greedy algorithms, and show that they are near-optimal. We also study the problems on special graph classes, prove lower bounds, and study the approximate reconstruction.

The second topic is optimization in planar graphs. We study two problems. In the *correlation clustering* problem, the input is a weighted graph, where every edge has a label of $\langle + \rangle$ or $\langle - \rangle$, indicating whether its endpoints are in the same category or in different categories. The goal is to find a partition of the vertices into categories that tries to respect the labels. In the *two-edge-connected augmentation* problem, the input is a weighted graph and a subset R of edges. The goal is to produce a minimum-weight subset S of edges, such that for every edge in R, its endpoints are two-edge-connected in the union of R and S. For *planar graphs*, we reduce correlation clustering to two-edge-connected augmentation, and show that both problems, although they are NP-hard, have a polynomial-time approximation scheme. We build on the brick decomposition technique developed recently.

Contents

Acknowledgements **1**

Prologue **3**

I Network Inference 5

1 Introduction **7**

1.1 Background . . . 7
1.2 The Problem . . . 8
1.3 Related Work . . . 9
1.4 Our Results . . . 11
1.5 Notations and Definitions . . . 15
1.6 Organization . . . 16

2 Voronoi Cell Decomposition **17**

2.1 Technique from Compact Routing . . . 17
2.2 Reconstruction via a Distance Oracle . . . 18
2.2.1 Subroutine: Selecting Centers . . . 19
2.2.2 Algorithm and Analysis . . . 21
2.3 Verification via a Distance Oracle . . . 22
2.3.1 Subroutine: Selecting Centers . . . 23
2.3.2 Algorithm and Analysis . . . 23

3 Greedy Algorithms **29**

3.1 Verification via a Distance Oracle . . . 29
3.2 Reconstruction via a Shortest Path Oracle . . . 30

4 Decomposition by Separators **33**
4.1 Preliminaries 33
4.2 Reconstruction of Chordal Graphs 34
4.2.1 Subroutine: Computing a Shortest Path 34
4.2.2 Subroutine: Partitioning by a Set 35
4.2.3 Algorithm and Analysis 38
4.3 Reconstruction of Outerplanar Graphs 41
4.3.1 Subroutine: Partitioning by a Polygon 41
4.3.2 Algorithm and Analysis 44
4.4 Verification of Treewidth Bounded Graphs 46
4.4.1 Warm up: Chordal Graphs 47
4.4.2 Extension: Graphs of Bounded Treewidth 48

5 Side Results **49**
5.1 Lower Bounds 49
5.1.1 General Graphs 49
5.1.2 Graphs of Bounded Degree 49
5.2 Approximate Reconstruction 50

6 Conclusion **55**

II Planar Graph Optimization **59**

7 Introduction **61**
7.1 Correlation Clustering 61
7.1.1 The Problem 61
7.1.2 Related Work 63
7.2 Two-Edge-Connected Augmentation 64
7.2.1 The Problem 64
7.2.2 Related Work 65
7.3 Our Results 65
7.4 Notations and Definitions 66
7.5 Organization 67

8 Reduction from Clustering to Augmentation **69**
8.1 First Stage 69
8.2 Second Stage 70

9 Techniques 73
9.1 Prize-Collecting Partition . . . 73
9.1.1 Steiner Forest . . . 73
9.1.2 Two-Edge-Connected Augmentation . . . 74
9.2 Brick Decomposition . . . 76
9.2.1 Steiner Tree . . . 76
9.2.2 Two-Edge-Connected Augmentation . . . 77
9.3 Framework of Approximation Schemes . . . 78
9.4 Doubling Brick Boundaries . . . 79
9.5 Sphere-Cut Decomposition . . . 80

10 Approximation Scheme 83
10.1 Preprocessing . . . 83
10.2 New Use of Brick Decomposition . . . 84
10.3 Structure Theorem . . . 86
10.3.1 Construction . . . 88
10.3.2 Analysis . . . 90
10.4 Dynamic Programming . . . 96
10.4.1 Specification of DP Table . . . 97
10.4.2 From Children to Parent . . . 99
10.4.3 Implementation . . . 101
10.5 Putting Them Together . . . 102

11 Conclusion 105

List of Publications 107

Bibliography 109

Acknowledgements

Research is like *the journey to the West*, a Chinese legend from the Tang dynasty. I would have probably gotten lost if not for all the help I have been offered. To all the people I list here, and to all those I have forgotten, thank you.

Mes premiers remerciements vont à ma directrice de thèse Claire Mathieu. Elle est une directrice parfaite pour moi. Ses compétences et sa personnalité m'ont beaucoup influencé pendant les trois ans. Quant à la recherche, elle m'a fait explorer des sujets merveilleux, et en plus, elle m'a appris des éléments importants, comme l'optimisme et l'insistance, pour conquérir les problèmes difficiles. Elle m'a donné une ambiance idéale: chaque fois après avoir discuté de recherche avec elle, j'en suis sortie très heureuse. Elle m'a mis en contact avec de nombreux chercheurs, et m'a donné beaucoup d'opportunités pour participer aux conférences. Lorsque j'ai eu des difficultés en dehors de la recherche, elle s'est comportée comme une amie pour m'aider. Grâce à elle, les trois ans de thèse sont devenus les moments les plus beaux de ma vie.

I would like to express my gratitude to Cyril Gavoille and Artur Czumaj who took the time to review this thesis carefully. I feel extremely fortunate they accepted to read, comment, and endorse my thesis. I am very grateful to the other members of my thesis committee: Frédéric Havet, Philip N. Klein, Marc Lelarge, Christophe Paul, and Stéphan Thomassé. I would also like to thank Fabrice Benhamouda who proofread part of this thesis.

It was my great pleasure to collaborate with seasoned researchers. I am very grateful to Marc Lelarge, who ignited my passion for algorithms, and taught me to write my first research article; to Philip N. Klein, who led me to the amazing world of planar graphs, and guided me as a mentor during my visit at Brown University; to Sampath Kannan, who shared many thoughtful insights with me; and to Mikkel Thorup, who invited me for a visit at University of Copenhagen. I would also like to thank Fabrice Benhamouda, Sergio Cabello, Yixin Cao, Vincent Cohen-Addad, Éric Colin de Verdière, Jacob Holm, Howard J. Karloff, Valerie King, Mathias Bæk Tejs Knudsen, Tancrède Lepoint, Tian Liu, Arnaud de Mesmay, Nabil Mustafa,

Thomas Sauerwald, He Sun, and Neal E. Young.

L'École normale supérieure est un endroit charmant pour passer ces années. Je suis très heureuse d'être dans le groupe Talgo avec Vincent Cohen-Addad, Éric Colin de Verdière, Varun Kanade, Reut Levi, Zhentao Li, Frederik Mallmann-Trenn, Arnaud de Mesmay et Victor Verdugo. Je voudrais aussi remercier Patrick Cousot, Jean Ponce, David Pointcheval, Lise-Marie Bivard, Isabelle Delais, Joëlle Isnard, Valérie Mongiat, Jacques Beigbeder, et la bibliothèque pour la qualité de leur travail.

Je voudrais remercier mes amis, qui ont rendu ma vie colorée. Mes remerciements spéciaux à Florent Urrutia, Marion Delehaye et Marie-Odile Faulconnier. Ils ont partagé mes joies et mes tristesses, ils m'ont fait découvrir les paysages merveilleux de France, et ils m'ont appris la phrase "C'est la vie". Dans les jardins, Florent et moi avons formalisé la vie avec la philosophie informatique ; sur la plage, Marion et moi avons couru après le cerf-volant au coucher du soleil ; et dans les Alpes, Marie-Odile et moi avons écouté la nature qui invite à la méditation. Je remercie aussi Antoine Amarilli, Samuel Bizien, Yoann Bourse, Floriane Dardard, Marc Jeanmougin, Jie Lin, Robin Morisset, Ludovic Patey, Pablo Rauzy, Bin Xu, Xuhong Zhang et Cheng Zhong.

Je remercie Paris, une ville unique du monde. C'est un paradis avec ses arts, sa culture, sa gastronomie, son ambiance, etc. Je suis très heureuse d'avoir passé ces années à Paris.

Je remercie de tout mon cœur mes parents Qiuhua Xu et Ye Zhou qui m'ont toujours aimée et soutenue. Je remercie aussi ma tante Yifei Yu qui a éclairé mon chemin depuis mon enfance.

En dernier lieu, j'aimerais adresser un grand merci particulier à mon copain Fabrice. Je n'irais pas bien loin s'il ne marchait pas à mes côtés. Je voudrais aussi remercier sa famille pour son soutien.

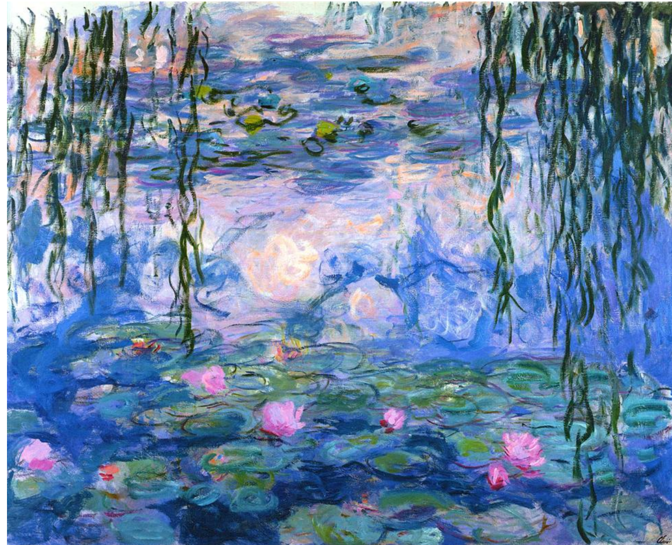
Paris, June 2015

Prologue

The *algorithm* is a unique form of art. Unlike fine arts such as painting and dancing, it has practical applications, yet can be just as expressive as *Monet's Water Lilies* or *Tchaikovsky's Swan Lake*. It can be as ambitiously utilized by Turing to shorten World War II in Europe, or as casually applied by a tourist to see as many fountains as possible during the *Grandes Eaux Musicales* spectacle at the *Versailles Garden*.

On the other hand, the *graph* attracts people's interests since Euler's time, such as in the *Seven Bridges of Königsberg* problem. It conveys the beauty of pure mathematics, and at the same time improves our lives, for example, through its applications to Internet networks.

This thesis is on two topics of algorithms for graphs: *network inference* and *planar graph optimization*.



Part I

Network Inference

Chapter 1

Introduction

1.1 Background

How efficiently can we find an unknown graph using distance or shortest path queries between its vertices? This is a natural theoretical question from the standpoint of recovery of hidden information. This question is related to the *reconstruction* of Internet networks. Discovering the topology of the Internet is a crucial step for building accurate network models and designing efficient algorithms for Internet applications. Yet, this topology can be extremely difficult to find, due to the dynamic structure of the network and to the lack of centralized control. Network reconstruction has been studied extensively [1, 14, 34, 76]. Sometimes we have some idea of what the network should be like, based perhaps on its state at some time in the past, and we want to check whether our image of the network is correct. This is network *verification* and has received attention recently [14, 25]. This is an important task for routing, error detection, or ensuring service-level agreement (SLA) compliance, etc. For example, Internet service providers (ISPs) offer services that require quality of service (QoS) guarantees, such as voice over IP services, and thus need to check regularly whether the networks are correct.

The topology of Internet networks can be investigated at the router and autonomous system (AS) level, where the set of routers (ASs) and their physical connections (peering relations) are the vertices and edges of a graph, respectively. Traditionally, we use tools such as *traceroute* and *mtrace* to infer the network topology. These tools generate path information between a pair of vertices. It is a common and reasonably accurate assumption that the generated path is the shortest one, i.e., minimizes the hop distance between that pair. In our first theoretical model, we assume that we have access to any pair of vertices and get in return a shortest path between them in the graph. Sometimes routers block *traceroute* and

mtrace requests (e.g., due to privacy and security concerns). In this case, inference of topology can only rely on delay information. In our second theoretical model, we assume that we have access to any pair of vertices and get in return the hop distance between them in the graph.

1.2 The Problem

Let $G = (V, E)$ be a hidden graph that is *connected*, *undirected*, and *unweighted*, where $|V| = n$. We consider two query oracles. A *shortest path oracle* receives a pair $(u, v) \in V^2$ and returns a shortest path between u and v.[1] A *distance oracle* receives a pair $(u, v) \in V^2$ and returns the number of edges on a shortest path between u and v.

In the *graph reconstruction* problem, we are given the vertex set V and have access to either a distance oracle or a shortest path oracle. The goal is to find every edge in E.

In the *graph verification* problem, again we are given V and have access to either oracle. In addition, we are given a connected, undirected, and unweighted graph $\hat{G} = (V, \hat{E})$. The goal is to check whether $\hat{G}$ is correct, that is, whether $\hat{G} = G$.

The efficiency of an algorithm is measured by its *query complexity*[2], i.e., the number of queries to an oracle. We focus on query complexity. All our algorithms are of polynomial time and space. We note that $O(n^2)$ queries are enough for reconstruction (also for verification) using a distance oracle or a shortest path oracle: we only need to query every pair of vertices, and return the pairs (u, v) such that $\delta(u, v) = 1$. We call this reconstruction algorithm EXHAUSTIVE-QUERY.

Let Δ denote the maximum degree of any vertex in the graph G. Unless otherwise stated, we assume that Δ is bounded, which is reasonable for real networks that we want to reconstruct or verify. Indeed, when Δ is $\Omega(n)$, both reconstruction and verification require $\Omega(n^2)$ distance or shortest path queries.

Let us focus on bounded degree graphs. It is not hard to see that $\Omega(n)$ distance or shortest path queries are required. The central question in this line of work is therefore: **Is the query complexity linear, quadratic, or somewhere in between?** We show that the query complexity for reconstruction using a distance oracle is subquadratic: $\tilde{O}(n^{3/2})$, and that the query complexity for reconstruction using a shortest path oracle or for verification using either oracle is near-linear: $n^{1+o(1)}$.

[1] If there are several shortest paths between u and v, the oracle returns an arbitrary one.

[2] Expected query complexity in the case of randomized algorithms.

1.3 Related Work

Graph inference using queries that reveal partial information has been studied extensively in different contexts, independently stemming from a number of applications.

All Shortest Path Model and All Distance Model. Beerliova et al. [14] studied the network discovery problem.

> [14] One can view this technique as an approach for discovering the topology of an unknown network by using a certain type of queries. [...] We formalize *network discovery* as a combinatorial optimization problem whose goal is to minimize the number of queries required to discover all edges and non-edges (absent edges) of the network.

In their shortest path oracle model, an oracle receives a node v and returns the shortest-path subgraph rooted at v, i.e., *all* shortest paths from v to *all* other nodes. Their motivation is the following:

> [14] With *traceroute* tools, one can determine the path that packets take in the Internet if they are sent from a node to some destination. If each *traceroute* experiment returns a random shortest path to the destination, this path would be part of the shortest-path subgraph. One could use repeated *traceroute* experiments to all destinations to discover all edges of the shortest-path subgraph.

For the verification problem, they showed that there is no approximation algorithm of factor $o(\log n)$ unless P = NP; and for the reconstruction problem, they gave a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$. They also provided experimental results on real data. They left as future directions:

> [14] It would also be interesting to study further query models. For example, a query could be given by nodes u and v and return all shortest paths between u and v (or just one shortest path).

As noted in [14], *traceroute* experiments in real networks often reveal only a single shortest path (or at most a few different paths), but not all shortest paths. Hence our motivation of the shortest path oracle model: the oracle returns an *arbitrary* shortest path between a given pair of nodes.

The authors of [14] also considered the distance oracle model: a distance oracle receives a node v and returns the distances from v to *all* other nodes in the graph. (This is in contrast to our distance oracle which returns the distance between a given pair of nodes.) They noted that, in many networks, it is realistically possible to obtain the distance information, while it is difficult or impossible to obtain the path information.

In the distance oracle model, they gave a randomized on-line algorithm for reconstruction with competitive ratio $O(\sqrt{n \log n})$. They proved that minimizing the number of queries for verification is NP-hard and they gave an approximation algorithm of factor $O(\log n)$, i.e., the number of queries is $O(\log n)$ times the optimum number of queries. This algorithm is based on a reduction to Set-Cover. To achieve this reduction, they showed an easy property for non-edge verification: a query at v discovers a non-edge ab when $|\delta(v, a) - \delta(v, b)| \geq 2$; and they showed more delicate properties for edge verification.

Our verification algorithm in Section 3.1 bears similarity: we also give a reduction to Set-Cover. In our model, edge verification is straightforward since the graph has bounded maximum degree. The focus is thus on non-edge verification. We will develop interesting properties for non-edge verification.

Evolutionary Biology. There has been extensive work on the reconstruction of evolutionary (phylogenetic) trees using a relation oracle or a distance oracle [24, 49, 54, 55, 59, 74, 78]. This problem was first introduced by Waterman et al. [78].

> [49] In taxonomy and molecular evolution, the problem of reconstructing a tree from distance data is very central.

In the distance oracle model, we query two species and get in return their distance in the tree. When the tree has maximum degree Δ, Hein [49] gave a reconstruction algorithm using $O(\Delta n \log_\Delta n)$ queries. King, Zhang, and Zhou [59] showed that this bound is tight by providing a matching lower bound. On the other hand, when the maximum degree is unbounded, there is a lower bound of $\Omega(n^2)$ [49]. Notice that in this problem, the hidden graph is a tree, whereas in our graph reconstruction problem, we allow the hidden graph to have an arbitrarily connected topology.

Statistical Models. Dall'Asta et al. [34] considered a shortest path oracle which, upon receiving a *random* pair of vertices, returns a shortest path between them. This is motivated by the fact that there may be some existing samplings of *traceroute* requests. Our model is different because we have the control on the pair of vertices sent to the oracle.

Network Realization. In this problem, we are given the distances between certain pairs of vertices and asked to determine the sparsest graph (in the unweighted case) or the graph of least total weight that realizes these distances. Chung et al. [30] introduced this problem, motivated by the applications to Internet tomography. They showed that this problem is NP-hard and admits a 2-approximation algorithm.

Network Inference Using Link Queries. A *link query* receives two nodes u and v, and reports whether there is an edge uv in the network. The goal is to discover as many links of the network as possible. Tarissan, Latapy, and Prieur [76] introduced this problem, motivated by social networks, like Facebook or Flickr, where the link query is a primary tool to discover the network topology. They provided strategies based on statistical properties of real-world networks, together with experimental results.

Bioinformatics Applications. Bouvel, Grebinski, and Kucherov [23] considered a graph reconstruction problem motivated by applications to *genome sequencing*. In their model, an oracle receives a subset of vertices S and returns the number of edges in the subgraph induced by S. The goal is to reconstruct the hidden graph. This model has been much studied, e.g., [7, 29, 48, 68, 73]. Our model is different since there is no counting.

Road Network Reconstruction. With path data such as GPS becoming available on a large scale, it is important to find shared structure in path data. Chen et al. [28] gave an algorithm to reconstruct the road network from a collection of path traces in *Euclidean space*. Our model is different since there is no geometry.

1.4 Our Results

The results here have been published in [A, D]. See Table 1.1 for a summary. The results based on the Voronoi cell decomposition and the greedy approaches are the main contributions.

Algorithms Based on Voronoi Cell Decomposition. To design reconstruction and verification algorithms, we apply some algorithmic ideas previously developed for compact routing [77] and for Voronoi cells [50].

Theorem 1.4.1. *For graph reconstruction using a distance oracle, there is a randomized algorithm with query complexity* $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log\log n)$*, which is* $\tilde{O}(n^{3/2})$ *when the maximum degree* $\Delta = O(\text{polylog}\, n)$*.*

Table 1.1: Results (for bounded degree graphs). Main results are in bold.

<table>
<tr><th>Objective</th><th>Query complexity</th></tr>
<tr><td>verification (either oracle)</td><td rowspan="2">$\boldsymbol{n^{1+o(1)}}$ Theorems 1.4.2 to 1.4.4
bounded treewidth: $\tilde{O}(n)$ Theorems 1.4.3, 1.4.4 and 1.4.8</td></tr>
<tr><td>reconstruction (shortest path oracle)</td></tr>
<tr><td>reconstruction (distance oracle)</td><td>$\boldsymbol{\tilde{O}(n^{3/2})}$ Theorem 1.4.1
$\Omega(n \log n / \log \log n)$ Theorem 1.4.9
chordal: $\tilde{O}(n)$ Theorem 1.4.6
outerplanar: $\tilde{O}(n)$ Theorem 1.4.7</td></tr>
</table>

The algorithm of Theorem 1.4.1 selects a set of $\tilde{O}(\sqrt{n})$ nodes (called *centers*) partitioning V into *Voronoi cells* of roughly the same size, and expands them slightly so as to cover every edge of G (Fig. 2.1). It is then sufficient to reconstruct each cell, which is done using exhaustive search inside that cell.

Theorem 1.4.2. *For graph verification using a distance oracle, there is a randomized algorithm with query complexity* $n^{1+O\left(\sqrt{(\log\log n+\log\Delta)/\log n}\right)}$, *which is* $n^{1+o(1)}$ *when the maximum degree* $\Delta = n^{o(1)}$.

The algorithm of Theorem 1.4.2 is a more sophisticated *recursive* version of the algorithm in Theorem 1.4.1. Again, it selects a set of centers partitioning V into *Voronoi cells*. To verify each cell, instead of using exhaustive search, the algorithm applies recursion. This is a challenge because, when we query a pair (u, v) in a cell, the oracle returns the distance between u and v in the entire graph, not in the cell. Our approach is to allow selection of centers *outside* the cell, while still limiting the subcells to being contained *inside* the cell (Fig. 2.2).

Greedy Algorithms. We provide simple greedy algorithms for verification and for reconstruction.

The main task for verification is to confirm the *non-edges* of the graph. We develop a necessary and sufficient condition for a set of queries to confirm all the non-edges. This condition enables us to reduce the non-edge verification problem to Set-Cover. As a counterpart of the greedy algorithm for Set-Cover, greedy non-edge verification repeatedly makes queries that confirm the largest number of non-edges that are not yet confirmed. We have:

Theorem 1.4.3. *If there is an algorithm for graph verification using $f(n, \Delta)$ distance queries, then a greedy algorithm for verification uses $O(\Delta n + \log n \cdot f(n, \Delta))$ distance or shortest path queries.*

Next, we extend the idea of greedy verification, and obtain a greedy algorithm for reconstruction as in the following theorem.

Theorem 1.4.4. *If there is an algorithm for graph verification using $f(n, \Delta)$ distance queries, then a greedy algorithm for reconstruction uses $O(\Delta n + \log n \cdot f(n, \Delta))$ shortest path queries.*

To prove Theorem 1.4.4, we show that each query to a shortest path oracle makes as much progress for reconstruction as the corresponding query to a distance oracle would have made for verifying a given graph. The main realization here is that reconstruction using a shortest path oracle can be viewed as the verification of a dynamically changing graph using a distance oracle.

Combining Theorems 1.4.2 to 1.4.4, we have:

Corollary 1.4.5. *For graph verification using either oracle and for graph reconstruction using a shortest path oracle, greedy algorithms have query complexity $n^{1+o(1)}$.*

Remark. *We note that the greedy algorithm for verification is much simpler than the algorithm using Voronoi cell decomposition (Theorem 1.4.2), although both algorithms have the same query complexity $n^{1+o(1)}$.*

Algorithms Based on Decomposition by Separators. For special classes of graphs, there exist *balanced separators* of small size. This enables us to design reconstruction and verification algorithms with $\tilde{O}(n)$ query complexity.

Theorem 1.4.6. *For reconstruction of chordal graphs using a distance oracle, there is a randomized algorithm with query complexity $O\left(\Delta^3 2^\Delta \cdot n(2^\Delta + \log^2 n) \log n\right)$, which is $\tilde{O}(n)$ when the maximum degree Δ is $O(\log \log n)$.*

A graph is *chordal* if every cycle of length greater than three has a chord: namely, an edge connecting two non-consecutive vertices on the cycle. An introduction to chordal graphs can be found in *e.g.,* [18].

Consider the following algorithm: Let x be a node that is on the most shortest paths between all pairs of vertices. The algorithm grows a clique separator including this node. Next, it partitions the graph into subgraphs with respect to this separator and recurses on each subgraph. Such partition is *balanced*, which ensures that there are $O(\log n)$ levels of the recursion.

However, computing all shortest paths to find x requires too many queries. Instead, our algorithm finds an approximate version of the node x. First, we give a subroutine to compute a shortest path between a pair of vertices using $O(n \log n)$ distance queries. To obtain an approximate version of x, the algorithm computes a sampling of shortest paths and selects the node which has the most occurrences among all sampled shortest paths. We show that the node obtained in this way leads to a balanced partition with high probability.

Theorem 1.4.7. *For reconstruction of outerplanar graphs using a distance oracle, there is a randomized algorithm with query complexity $O(\Delta^2 \cdot n \log^3 n)$, which is $\tilde{O}(n)$ when the maximum degree $\Delta = O(\text{polylog}\, n)$.*

A graph is *outerplanar* if it can be embedded in the plane with all vertices on the exterior face. Chartrand and Harary [27] first introduced outerplanar graphs and showed that an outerplanar graph contains no subgraph homeomorphic from K_4 or $K_{2,3}$. Outerplanar graphs have received much attention in the literature because of their simplicity and numerous applications.

Similar as for chordal graphs, the algorithm first finds a separator using random sampling and statistical estimation, and then partitions the graph into subgraphs with respect to this separator and recurses on each subgraph. However, the separator here may be a *polygon* of unbounded size. So we need more care in the algorithmic design.

Remark. *The query complexity for reconstructing outerplanar graphs is only slightly worse than the optimal query complexity $O(\Delta n \log_\Delta n)$ for reconstructing trees (a special case of outerplanar graphs) in [49]. On the other hand, the tree model typically restricts queries to pairs of tree leaves, but our model allows queries of any pair of vertices, not just leaves.*

Theorem 1.4.8. *For verification of graphs of treewidth w using a distance oracle, there is a deterministic algorithm with query complexity $O(\Delta(\Delta + w \log n) n \log n)$, which is $\tilde{O}(n)$ when Δ and w are $O(\text{polylog}\, n)$.*

The algorithm uses some *bag* of a *tree decomposition* to separate the graph into balanced subgraphs, and then recursively verifies each subgraph. In the recursive calls, it adds a few weighted edges to each subgraph in order to preserve the distance metric.

Lower Bounds. For graphs of unbounded degree, we give a simple $\Omega(n^2)$ query lower bound for both reconstruction and verification and under both oracle models. This lower bound is achieved using a star graph plus possibly one more edge. We

note that this lower bound holds even when the graph is restricted to outerplanar, to chordal, or to bounded treewidth.

On the other hand, for graphs of bounded degree, it is easy to see that both reconstruction and verification require $\Omega(n)$ distance or shortest path queries. In addition, there is a slightly better lower bound for graph reconstruction using a distance oracle (Theorem 1.4.9). We thank Uri Zwick and Cyril Gavoille for this lower bound.

Theorem 1.4.9. *For graph reconstruction using a distance oracle, assuming the maximum degree $\Delta \geq 3$ is such that $\Delta = o\left(n^{1/2}\right)$, any algorithm has query complexity $\Omega(\Delta n \log n / \log(\log n / \log \Delta))$.*

Approximate Reconstruction. Recall that for graphs of unbounded degree, we need $\Omega(n^2)$ distance queries for graph reconstruction. However, fewer queries are needed if we consider an approximate version of the reconstruction problem. We say that the metric $\tilde{\delta}$ is an *f-approximation* of the metric δ if for every pair of nodes (u, v), $\tilde{\delta}(u, v) \leq \delta(u, v) \leq f \cdot \tilde{\delta}(u, v)$. The goal is to compute an f-approximation of the metric using a distance oracle. We give a simple algorithm with query complexity $O(n^2/f)$ and we show a matching lower bound.

1.5 Notations and Definitions

Let $G = (V, E)$ be a connected, undirected, and unweighted graph, where V is the vertex set and E is the edge set. Let δ be the distance metric of G. For a subset of vertices $S \subseteq V$ and a vertex $v \in V$, define $\delta(S, v)$ to be $\min_{s \in S} \delta(s, v)$. For $v \in V$, define the neighborhood of v as $N(v) = \{u \in V : \delta(u, v) \leq 1\}$, and define the neighborhood of v within distance 2 as $N_2(v) = \{u \in V : \delta(u, v) \leq 2\}$. For $S \subseteq V$, define the neighborhood of S as $N(S) = \bigcup_{s \in S} N(s)$. We define $\hat{\delta}$, $\hat{N}$, and $\hat{N}_2$ similarly with respect to the graph $\hat{G}$.

For a pair of distinct vertices $(u, v) \in V^2$, we say that uv is an *edge* of G if $uv \in E$, and is a *non-edge* of G if $uv \notin E$.

For a subset of vertices $S \subseteq V$, let $G[S]$ be the subgraph induced by S. For a subset of edges $H \subseteq E$, we identify H with the subgraph induced by the edges of H. Let δ_H denote the distance metric of the subgraph H.

For a vertex $s \in V$ and a subset $T \subseteq V$, define $\textsc{Query}(s, T)$ (or equivalently $\textsc{Query}(T, s)$) as $\textsc{Query}(s, t)$ for every $t \in T$. For subsets $S, T \subseteq V$, define $\textsc{Query}(S, T)$ as $\textsc{Query}(s, t)$ for every $(s, t) \in S \times T$.

In the verification problem, an algorithm, after performing a set of queries, outputs *no* if some query gives the wrong distance (or shortest path), and outputs *yes* if all queries give the right distances (or shortest paths).

1.6 Organization

The main contributions of Part I of this thesis are in Chapters 2 and 3. In Chapter 2, we give reconstruction and verification algorithms based on Voronoi cell decomposition. Section 2.1 reviews the *center-selecting* technique from compact routing [77], which is the main subroutine in our algorithms. Sections 2.2 and 2.3 prove Theorems 1.4.1 and 1.4.2, respectively. In Chapter 3, we give greedy algorithms for verification using a distance oracle and for reconstruction using a shortest path oracle. Sections 3.1 and 3.2 prove Theorems 1.4.3 and 1.4.4, respectively.

Section 4.1 gives some preliminaries on *separators*, and then Sections 4.2 to 4.4 prove Theorems 1.4.6 to 1.4.8, respectively. In Section 5.1, we consider lower bounds. In Section 5.2, we provide results on the approximate reconstruction.

Finally, in Chapter 6, we recapitulate our results and expose some future directions of research to solve the open problems raised there.

Chapter 2

Voronoi Cell Decomposition

In this chapter, we design a reconstruction algorithm (Section 2.2) and a verification algorithm (Section 2.3) using a distance oracle. Both algorithms are based on the *center-selecting* technique, which comes from *compact routing* [77]. We first review this technique in Section 2.1.

2.1 Technique from Compact Routing

A *routing scheme* is a mechanism that delivers packets of information from any node to any other node in the network. The *compact routing* problem aims at finding a tradeoff between the space and the efficiency of routing, see, e.g., [8, 9, 32, 38, 40, 43, 70, 77].

In [77], Thorup and Zwick gave a compact routing scheme that uses $\tilde{O}(n^{1/2})$ bits of memory at each node such that, the ratio between the length of a path on which a packet is routed and the length of a shortest path is at most 3. To achieve that, they selected a set of *centers*. For a packet to reach a destination v that is far away, it first goes to the center that is closest to v, and then follows the shortest path from this center to v.

More precisely, let $A \subseteq V$ be a subset of nodes called *centers*. For every $w \in V$, define the *cluster* of w with respect to the set A as

$$C_A(w) := \{v \in V : \delta(w, v) < \delta(A, v)\}.$$

Note that if $w \in A$, then $C_A(w) = \emptyset$, since $\delta(w, v) \geq \delta(A, v)$, for every $v \in V$. The subscript A is omitted when clear from the context. They observed that the required memory at each node w is $O(|A| + C(w))$. So the goal is to find a set of centers A such that every $C(w)$ is small, i.e., roughly $n/|A|$.

Algorithm 2.1 Finding Centers [77]

```
1: function CENTERS(G, s)
2:     A ← ∅, W ← V
3:     while W ≠ ∅ do
4:         A′ ← SAMPLE(W, s)
5:         A ← A ∪ A′
6:         for w ∈ W do
7:             C(w) ← {v ∈ V : δ(w, v) < δ(A, v)}
8:         W ← {w ∈ W : |C(w)| > 4n/s}
9:     return A
```

Algorithm 2.1 reviews the center-selecting algorithm from [77]. The algorithm takes as input the graph G and an integer parameter $s \in [1, n]$ and outputs a set of centers A. It uses a subroutine $\textsc{Sample}(W, s)$, which receives a set $W \subseteq V$ and an integer s and returns a random subset of W obtained by selecting each element, independently, with probability $s/|W|$. If $|W| \leq s$, then $\textsc{Sample}(W, s)$ returns the set W itself.

Thorup and Zwick [77] proved the following lemma, which is the key to show the $\tilde{O}(n^{1/2})$ memory bound at each node in their routing scheme.

Lemma 2.1.1 (Rephrasing of Theorem 3.1 in [77])**.** *With probability at least* $1/2$*, the algorithm* $\textsc{Centers}$ *(Algorithm 2.1) outputs a set* $A \subseteq V$ *such that* $|A| \leq 4s \log n$*, and that* $|C(w)| \leq 4n/s$*, for every* $w \in V$*.*

We will use modified versions of the algorithm $\textsc{Centers}$ (Algorithm 2.1) and its analysis (Lemma 2.1.1) in the design of the reconstruction and verification algorithms in Sections 2.2 and 2.3.

2.2 Reconstruction via a Distance Oracle

In this section, we prove the following theorem.

Theorem 2.2.1. *For graph reconstruction using a distance oracle, there is a randomized algorithm (Algorithm 2.3), such that with probability at least* $1/4$*, it terminates after* $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log\log n)$ *queries and returns the edge set* E*.*

Theorem 2.2.1 implies Theorem 1.4.1, because if the reconstruction algorithm in Theorem 2.2.1 fails to terminate after $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log\log n)$ queries, we can stop it and execute it again. The expected number of executions of the algorithm is a constant. Therefore we obtain a reconstruction algorithm with expected query complexity $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log\log n)$, as stated in Theorem 1.4.1.

Algorithm 2.2 Finding Centers Using Estimation

```
1: function ESTIMATED-CENTERS(V, s)
2:     A ← ∅, W ← V
3:     T ← K · s · log n · log log n          ▷ K = O(1) defined in Lemma 2.2.2
4:     while W ≠ ∅ do
5:         A' ← SAMPLE(W, s)
6:         QUERY(A', V)
7:         A ← A ∪ A'
8:         for w ∈ W do
9:             X ← random multi-subset of V with T elements
10:            QUERY(X, w)
11:            C̃(w) ← |{v ∈ X : δ(w, v) < δ(A, v)}| · n/T
12:        W ← {w ∈ W : C̃(w) ≥ 5n/s}
13:    return A
```

2.2.1 Subroutine: Selecting Centers

The reconstruction algorithm uses a subroutine ESTIMATED-CENTERS (see Algorithm 2.2) to find *centers* $A \subseteq V$ such that the vertices of V are roughly equipartitioned into the Voronoi cells centered at vertices in A. This algorithm is a modified version of the algorithm CENTERS (Algorithm 2.1). It takes as input the vertex set V and a parameter $s \in [1, n]$ and outputs a set of centers A. Unlike the algorithm CENTERS, it uses sampling to estimate each $|C(w)|$ so as to reduce the query complexity. Recall that the algorithm CENTERS eliminates $w \in W$ when $|C(w)| \leq 4n/s$. However, in our query model, computing $|C(w)|$ would require $\Omega(n)$ queries for each w. Instead, the algorithm ESTIMATED-CENTERS computes an estimate $\tilde{C}(w)$ of $|C(w)|$ using fewer queries, and then eliminates $w \in W$ when $\tilde{C}(w)$ is small. The following lemma guarantees the performance of the algorithm ESTIMATED-CENTERS. It is a counterpart of Lemma 2.1.1. Its proof combines arguments from [77] and Chernoff bounds.

Lemma 2.2.2. *Let K (Algorithm 2.2 of Algorithm 2.2) be some well-chosen constant. With probability at least $1/4$, the algorithm* ESTIMATED-CENTERS *(Algorithm 2.2) terminates after $O(s \cdot n \cdot \log^2 n \cdot \log\log n)$ queries, and outputs a set $A \subseteq V$ such that $|A| \leq 12s \log n$, and that $|C(w)| \leq 6n/s$, for every $w \in V$.*

Proof. Let W_i be the set W at the beginning of the i-th iteration. Let $A' =$ SAMPLE(W, s) be the centers selected in this iteration, and let A be the union of A' with previously selected centers. We say that the i-th iteration is *successful* if

$$\sum_{w \in W_i} |C_A(w)| \leq 2n|W_i|/s. \tag{2.1}$$

Thorup and Zwick [77] showed that every iteration is successful with probability at least 1/2.

Consider a node $w \in W_i$. Let X be a random multi-subset of V with T elements. Define $Y = |\{v \in X : \delta(w, v) < \delta(A, v)\}|$, which is $|X \cap C_A(w)|$. Let $E[Y]$ be the expected value of Y. By standard Chernoff bounds, there is an absolute constant K such that:

$$\begin{cases} \mathbb{P}\left[Y \geq 5T/s\right] \geq 1 - 1/(96n \log n), & \text{if } E[Y] > 6T/s \\ \mathbb{P}\left[Y < 5T/s\right] > 1 - 1/(96n \log n), & \text{if } E[Y] < 4T/s. \end{cases}$$

Note that $|C_A(w)| = E[Y] \cdot n/T$. Define $\tilde{C}(w) = Y \cdot n/T$. Thus with probability at least $1 - 1/(48n \log n)$, we have:

$$\begin{cases} \tilde{C}(w) \geq 5n/s, & \text{if } |C_A(w)| > 6n/s \\ \tilde{C}(w) < 5n/s, & \text{if } |C_A(w)| < 4n/s. \end{cases} \tag{2.2}$$

Since $|W_i| \leq n$, with probability at least $1 - 1/(48 \log n)$, Property (2.2) holds for all $w \in W_i$ in the i-th iteration. We call iterations in which this happens *correct* iterations. As a consequence, if the i-th iteration is correct, then for every $w \in W_{i+1}$, we have $|C_A(w)| \geq 4n/s$.

If the i-th iteration is both successful and correct, which happens with probability at least $1/2 - 1/(48 \log n) > 1/3$, then we have

$$4n|W_{i+1}|/s \leq \sum_{w \in W_i} |C_A(w)| \leq 2n|W_i|/s,$$

and thus $|W_{i+1}| \leq |W_i|/2$. Since $|W|$ is non-increasing during the algorithm, the expected number of iterations is at most $3 \log n$.

By Markov's inequality, with probability at least 3/4, the number of iterations is at most $12 \log n$. The probability that the first (at most) $12 \log n$ iterations are correct is at least $1 - 12 \log n/(48 \log n) = 3/4$. Therefore, with probability at least 1/2, there are at most $12 \log n$ iterations and all iterations are correct. In that case, every $w \in V$ has been eliminated when $|C_A(w)| \leq 6n/s$. Observe that $|C_A(w)|$ cannot increase when elements are added to A. Therefore $|C_A(w)| \leq 6n/s$ for every $w \in V$ when the **while** loop terminates.

The expected size of A is at most $3s \log n$, since the expected number of iterations is at most $3 \log n$, and in every iteration, a set A' of expected size s is added to A. By Markov's inequality, $|A| \leq 12s \log n$ with probability at least 3/4.

All together, with probability at least 1/4, there are at most $12 \log n$ iterations, and $|A| \leq 12s \log n$, and $|C_A(w)| \leq 6n/s$ for every $w \in V$. In that case, the number of queries is at most

$$|A| \cdot n + (12 \log n) \cdot nT = O(s \cdot n \cdot \log^2 n \cdot \log \log n).$$

This completes the proof. □

Algorithm 2.3 Reconstruction

```
1: procedure RECONSTRUCT(V, s)
2:     A ← ESTIMATED-CENTERS(V, s)          ▷ every pair in A × V is queried
3:     for a ∈ A do
4:         QUERY(N_2(a), V)
5:         for b ∈ N_2(a) do
6:             C(b) ← {v ∈ V : δ(b, v) < δ(A, v)}
7:         D_a ← ⋃{C(b) : b ∈ N_2(a)} ∪ N_2(a)
8:         E_a ← EXHAUSTIVE-QUERY(D_a)
9:     return ⋃_a E_a
```

2.2.2 Algorithm and Analysis

The reconstruction algorithm (Algorithm 2.3) takes as input the vertex set V and an integer parameter $s \in [1, n]$, and computes the edge set of G. It first finds a set of centers A using ESTIMATED-CENTERS, and then partitions the graph into slightly overlapped subgraphs with respect to the centers in A, see Figure 2.1. More precisely, we define, for each $a \in A$, its *extended Voronoi cell* $D_a \subseteq V$ as

$$D_a := \bigcup \{C(b) : b \in N_2(a)\} \cup N_2(a). \tag{2.3}$$

The algorithm then proceeds by exhaustive search within each subgraph $G[D_a]$, and returns all the edges found in these subgraphs. Inspired by the Voronoi diagram partitioning in [50], we show in Lemma 2.2.3 that these subgraphs together cover every edge of the graph. Thus the output of the algorithm is indeed the edge set E.

Lemma 2.2.3. $\bigcup_{a \in A} G[D_a]$ *covers every edge of* G.

Proof. Let uv be any edge of G. We prove that there is some $a \in A$, such that both u and v are in D_a. Without loss of generality, we assume $\delta(A, u) \le \delta(A, v)$. We choose $a \in A$ such that $\delta(a, u) = \delta(A, u)$. If $\delta(a, u) \le 1$, then both u and v are in $N_2(a) \subseteq D_a$. If $\delta(a, u) \ge 2$, let b be the vertex at distance 2 from a on a shortest a-to-u path in G. By the triangle inequality, we have $\delta(b, v) \le \delta(b, u) + \delta(u, v) = \delta(b, u) + 1$. Since $\delta(b, u) = \delta(a, u) - 2$ and $\delta(a, u) = \delta(A, u) \le \delta(A, v)$, we have $\delta(b, u) < \delta(A, u)$ and $\delta(b, v) < \delta(A, v)$. So both u and v are in $C(b)$, and thus in D_a, since $b \in N_2(a)$. □

Query Complexity Analysis. In the first step of the algorithm RECONSTRUCT, by Lemma 2.2.2, with probability at least 1/4, ESTIMATED-CENTERS uses $O(s \cdot n \cdot \log^2 n \cdot \log \log n)$ queries, and outputs a set $A \subseteq V$ such that $|A| \le 12 s \log n$, and

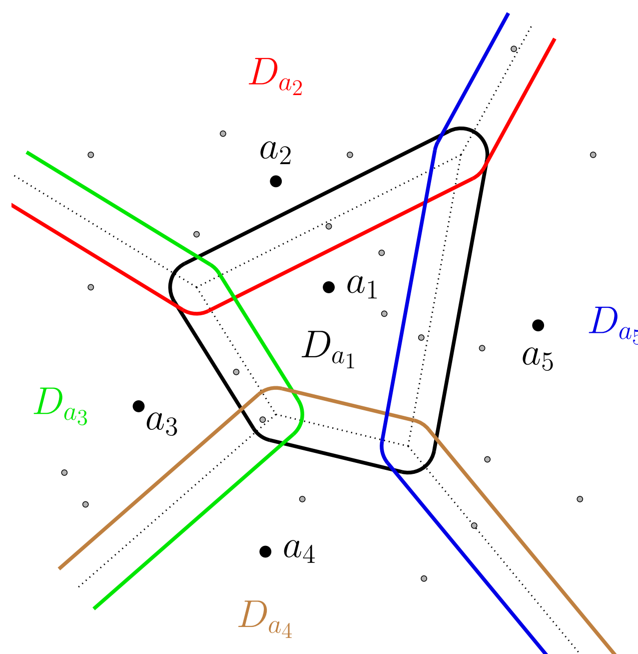


Figure 2.1: Partition by centers. Here vertices $a_1, \ldots, a_5$ are centers in A. The dotted lines indicate the partition of V into Voronoi cells by those centers. Extending the Voronoi cells slightly, we obtain the cells $D_{a_1}, \ldots, D_{a_5}$.

$|C(w)| \leq 6n/s$, for every $w \in V$. In this case, the following steps of the algorithm have query complexity:

$$\sum_{a \in A} \left(|N_2(a)| \cdot n + |D_a|^2 \right) = O(s \log n \cdot \Delta^2 (n + \Delta^2 n^2 / s^2)),$$

using the bounds on $|A|$ and $|C(w)|$, and the fact that $|N_2(a)| \leq \Delta^2 + 1$.

Let $s = \Delta \cdot \sqrt{n}$. Then with probability at least $1/4$, the algorithm Reconstruct terminates after $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log\log n)$ queries, as stated in Theorem 2.2.1.

2.3 Verification via a Distance Oracle

In this section, we prove Theorem 1.4.2 that we recall:

Theorem 1.4.2. *For graph verification using a distance oracle, there is a randomized algorithm with query complexity* $n^{1+O\left(\sqrt{(\log\log n + \log \Delta)/\log n}\right)}$, *which is* $n^{1+o(1)}$ *when the maximum degree* $\Delta = n^{o(1)}$.

Algorithm 2.4 Finding Centers for a Subset

```
1: function SUBSET-CENTERS(Ĝ, U, s)
2:     A ← ∅, W ← V
3:     while W ≠ ∅ do
4:         A' ← SAMPLE(W, s)
5:         A ← A ∪ A'
6:         for w ∈ W do
7:             Ĉ(w) ← {v ∈ V : δ̂(w, v) < δ̂(A, v)}
8:         W ← {w ∈ W : |Ĉ(w) ∩ U| > 4|U|/s}
9:     return A
```

2.3.1 Subroutine: Selecting Centers

The verification algorithm uses the subroutine SUBSET-CENTERS (Algorithm 2.4), which takes as input a graph $\hat{G} = (V, \hat{E})$, a *subset* of vertices $U \subseteq V$, and an integer $s \in [1, n]$, and outputs a set of *centers* $A \subseteq V$ such that in the graph $\hat{G}$, the vertices of the subset U are roughly equipartitioned into the Voronoi cells centered at vertices in A. This algorithm is a generalization of the algorithm CENTERS (Algorithm 2.1). When the subset U equals V, this algorithm becomes the same as the algorithm CENTERS. For $w \in V$, we recall the definition of w's cluster $C_A(w) := \{v \in V : \delta(w, v) < \delta(A, v)\}$. Similarly, we define w's cluster with respect to the graph $\hat{G}$ as $\hat{C}_A(w) := \{v \in V : \hat{\delta}(w, v) < \hat{\delta}(A, v)\}$. The subscript A is omitted when clear from the context. The following lemma is a straightforward extension of Lemma 2.1.1.

Lemma 2.3.1. *With probability at least* $1/2$*, the algorithm* SUBSET-CENTERS *(Algorithm 2.4) outputs a set* $A \subseteq V$*, such that* $|A| \leq 4s \log n$*, and that* $|\hat{C}(w) \cap U| \leq 4|U|/s$*, for every* $w \in V$*. It uses no queries since* $\hat{G}$ *is given.*

2.3.2 Algorithm and Analysis

The task of verification comprises verifying that every edge of $\hat{G}$ is an edge of G, and verifying that every non-edge of $\hat{G}$ is a non-edge of G. The second part is called *non-edge verification*. In the second part, we assume that the first part is already done, which guarantees that $\hat{E} \subseteq E$. For graphs of bounded degree, the first part requires only $O(\Delta n)$ queries, thus the focus is on non-edge verification.

We design a recursive algorithm for non-edge verification. Let $U \subseteq V$ represent the set of vertices in a recursive call. The goal is to verify that every non-edge of $\hat{G}[U]$ is a non-edge of $G[U]$. This is equivalent to verifying that every edge of $G[U]$ is an edge of $\hat{G}[U]$.

Let A be a set of centers computed by SUBSET-CENTERS. We define, for each $a \in A$, its *extended Voronoi cell restricted on U*:

$$D_a := \left(\bigcup \{C(b) : b \in N_2(a)\} \cup N_2(a)\right) \cap U. \tag{2.4}$$

Similarly, with respect to the graph $\hat{G}$, we define:

$$\hat{D}_a := \left(\bigcup \left\{\hat{C}(b) : b \in \hat{N}_2(a)\right\} \cup \hat{N}_2(a)\right) \cap U. \tag{2.5}$$

The following lemma is a simple extension of Lemma 2.2.3.

Lemma 2.3.2. $\bigcup_{a\in A} G[D_a]$ *covers every edge of* $G[U]$.

Proof. Let uv be any edge of $G[U]$. We prove that there is some $a \in A$, such that both u and v are in D_a. Without loss of generality, we assume $\delta(A, u) \leq \delta(A, v)$. We choose $a \in A$ such that $\delta(a, u) = \delta(A, u)$. If $\delta(a, u) \leq 1$, then both u and v are in $N_2(a) \cap U \subseteq D_a$. If $\delta(a, u) \geq 2$, let b be the vertex at distance 2 from a on a shortest a-to-u path in G. By the triangle inequality, we have $\delta(b, v) \leq \delta(b, u) + \delta(u, v) = \delta(b, u) + 1$. Since $\delta(b, u) = \delta(a, u) - 2$ and $\delta(a, u) = \delta(A, u) \leq \delta(A, v)$, we have $\delta(b, u) < \delta(A, u)$ and $\delta(b, v) < \delta(A, v)$. So both u and v are in $C(b)$, and thus in D_a, since $b \in N_2(a)$. □

From Lemma 2.3.2, in order to verify that every edge of $G[U]$ is an edge of $\hat{G}[U]$, we only need to verify that every edge of $G[D_a]$ is an edge of $\hat{G}[D_a]$, for every $a \in A$. This enables us to apply recursion on each D_a.

The main difficulty is: **How to obtain D_a efficiently?** If we compute D_a from its definition, we need to compute $N_2(a)$, which requires $\Omega(n)$ queries since $N_2(a)$ may contain nodes outside U. Instead, a careful analysis shows that we can check whether $D_a = \hat{D}_a$ without even knowing $N_2(a)$, whereas $\hat{D}_a$ can be inferred from the graph $\hat{G}$ with no queries. This is shown in Lemma 2.3.3, which is the key to the design of the verification algorithm.

Lemma 2.3.3. *Assume that* $\hat{E} \subseteq E$. *If* $\delta(u, v) = \hat{\delta}(u, v)$ *for every pair* (u, v) *from* $\bigcup_{a\in A} \hat{N}_2(a) \times U$, *then* $D_a = \hat{D}_a$ *for all* $a \in A$.

Proof. The proof is delicate but elementary. For every $b \in \bigcup_{a\in A} \hat{N}_2(a)$, we have $\hat{C}(b) \cap U = C(b) \cap U$, because $\hat{\delta}(b, u) = \delta(b, u)$ and $\hat{\delta}(A, u) = \delta(A, u)$ for every $u \in U$. Therefore, $\hat{D}_a$ can be rewritten as

$$\hat{D}_a = \left(\bigcup \left\{C(b) : b \in \hat{N}_2(a)\right\} \cup \hat{N}_2(a)\right) \cap U.$$

Since $\hat{E} \subseteq E$, we have $\hat{N}_2(a) \subseteq N_2(a)$. Therefore $\hat{D}_a \subseteq D_a$.

On the other hand, we have $N_2(a) \cap U \subseteq \hat{N}_2(a) \cap U$, because $\hat{\delta}(a, u) = \delta(a, u)$ for every $u \in N_2(a) \cap U$. To prove $D_a \subseteq \hat{D}_a$, it only remains to show that, for any

Algorithm 2.5 Verification

```
 1: procedure Verify-Subgraph(Ĝ, U, s)
 2:     if |U| > n_0 then
 3:         repeat
 4:             A ← Subset-Centers(Ĝ, U, s)
 5:         until |A| ≤ 4s log n and |Ĉ(w) ∩ U| ≤ 4|U|/s for every w ∈ V
 6:         for a ∈ A do
 7:             Query(N̂_2(a), U)
 8:             for b ∈ N̂_2(a) do
 9:                 Ĉ(b) ← {v ∈ V : δ̂(b, v) < δ̂(A, v)}
10:             D̂_a ← (⋃{Ĉ(b) : b ∈ N̂_2(a)} ∪ N̂_2(a)) ∩ U
11:             Verify-Subgraph(Ĝ, D̂_a, s)
12:     else
13:         Query(U, U)
```

vertex $u \notin N_2(a)$ such that $u \in C(b) \cap U$ for some $b \in N_2(a)$, we have $u \in C(x) \cap U$ for some $x \in \hat{N}_2(a)$. We choose x to be the vertex at distance 2 from a on a shortest a-to-u path in $\hat{G}$. By the assumption and the definition of x, we have:

$$\delta(x, u) = \hat{\delta}(x, u) = \hat{\delta}(a, u) - 2 = \delta(a, u) - 2.$$

By the triangle inequality, and using $b \in N_2(a)$ and $u \in C(b)$, we have:

$$\delta(a, u) \leq \delta(a, b) + \delta(b, u) \leq 2 + \delta(b, u) < 2 + \delta(A, u).$$

Therefore $\delta(x, u) < \delta(A, u)$. Thus $u \in C(x) \cap U$. □

The recursive algorithm Verify-Subgraph for non-edge verification is given in Algorithm 2.5. It receives a graph $\hat{G} = (V, \hat{E})$, a subset $U \subseteq V$, and an integer parameter s, and verifies all the non-edges of $\hat{G}[U]$. It first queries every $(u, v) \in \bigcup_{a \in A} \hat{N}_2(a) \times U$, and then recurses on each extended Voronoi cell $\hat{D}_a$, see Figure 2.2. The parameters s and n_0 are defined later. Correctness of the algorithm follows from Lemmas 2.3.2 and 2.3.3.

Query Complexity Analysis. To provide intuition, we first analyze an algorithm of 4 recursive levels, and show that its query complexity is $\tilde{O}(n^{4/3})$. To simplify the presentation, we assume $\Delta = O(1)$. Let $s = n^{1/3}$ and let n_0 be some well-chosen constant. Consider any recursive call Verify-Subgraph$(\hat{G}, U, s)$ where $|U| > n_0$. Let $A \subseteq V$ be the centers at the end of the **repeat** loop. By Lemma 2.3.1, the expected number of **repeat** loops is constant. For every $a \in A$,

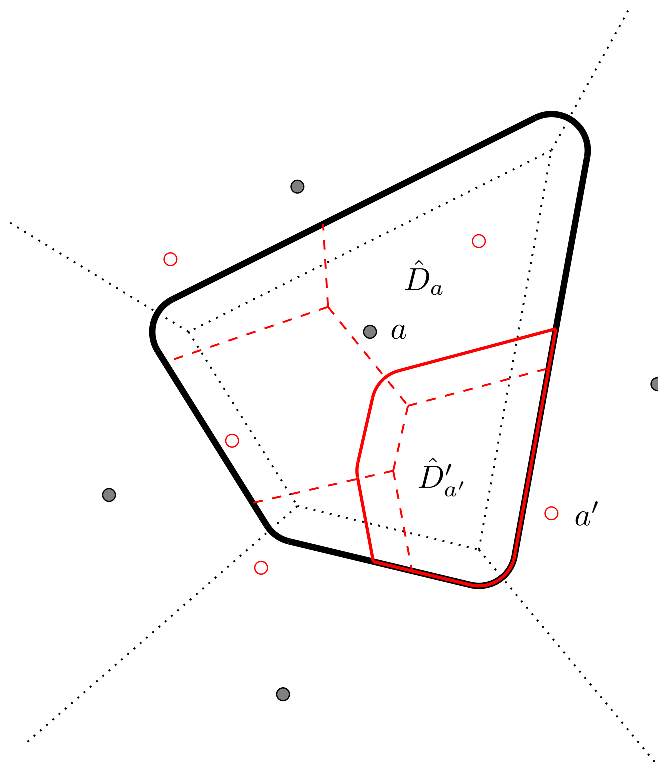


Figure 2.2: Two levels of recursive calls of VERIFY-SUBGRAPH($\hat{G}, V, s$). The solid points are top-level centers returned by SUBSET-CENTERS($\hat{G}, V, s$). The dotted lines indicate the partition of V into Voronoi cells by those centers. For a center a, $\hat{D}_a$ is the region that is a bit larger than the Voronoi cell of a. On the second level of the recursive call for $\hat{D}_a$, the hollow points are the centers returned by SUBSET-CENTERS($\hat{G}, \hat{D}_a, s$). Observe that some of those centers lie outside $\hat{D}_a$. The dashed lines indicate the partition of $\hat{D}_a$ into Voronoi cells by those centers. For a center a' on the second level, $\hat{D}'_{a'}$ is the region that is a bit larger than the Voronoi cell of a'.

$\hat{N}_2(a)$ has constant size, since $\Delta = O(1)$. Every $\hat{C}(w) \cap U$ has size $O(|U|/n^{1/3})$, so every $\hat{D}_a$ has size $O(|U|/n^{1/3})$. Since $|A| = \tilde{O}(n^{1/3})$, the number of recursive calls on the next level is $\tilde{O}(n^{1/3})$. Therefore during the recursion, on the second level, there are $\tilde{O}(n^{1/3})$ recursive calls, where every subset has size $O(n^{2/3})$; on the third level, there are $\tilde{O}(n^{2/3})$ recursive calls, where every subset has size $O(n^{1/3})$; and on the fourth level, there are $\tilde{O}(n)$ recursive calls, where every subset has size $O(1)$. Every recursive call with subset U uses $\tilde{O}(n^{1/3} \cdot |U|)$ queries. Therefore, the overall query complexity is $\tilde{O}(n^{4/3})$.

Next, we give the full proof of the complexity as stated in Theorem 1.4.2. Define

$$k_0 = \left\lfloor \sqrt{\frac{\log n}{\log\left(\log n \cdot 32(\Delta^2+1)^2\right)}} \right\rfloor.$$

Let $s = n^{1/k_0}$ and $n_0 = \left(4(\Delta^2+1)\right)^{k_0}$. Consider any recursive call when $|U| > n_0$. Let $A \subseteq V$ be the centers at the end of the **repeat** loop. Then $|A| \le 4s \log n$ and every $|\hat{C}(w) \cap U|$ is at most $4|U|/s$. By Lemma 2.3.1, the expected number of **repeat** loops is constant. Since the graph has maximum degree Δ, the size of every $\hat{D}_a$ is at most $(\Delta^2+1) \cdot \max(4|U|/s, 1)$. Therefore by induction, for any $1 \le k \le k_0+1$, any subset U on the kth level of the recursion has size at most $t_k := n\left(4(\Delta^2+1)/s\right)^{k-1}$, where $t_{k_0+1} = n_0$. Hence the maximum level of the recursion is at most k_0+1.

Consider the recursive calls with $|U| \le n_0$. There are at most $(2s\log n)^{k_0}$ such calls and each takes $|U|^2 \le \left(4(\Delta^2+1)\right)^{2k_0}$ queries. So their overall query complexity is at most $n \cdot \left(\log n \cdot 32(\Delta^2+1)^2\right)^{k_0} \le n^{1+1/k_0}$.

Consider the recursive calls with $|U| > n_0$ on the kth level of the recursion for some fixed $k \in [1, k_0]$.[1] There are at most $(2s\log n)^{k-1}$ such calls and each takes at most $(\Delta^2+1)|A| \cdot |U|$ queries, where $|U| \le t_k$. So their overall query complexity is at most $n^{1+1/k_0}\left(\log n \cdot 8(\Delta^2+1)\right)^{k}$. Summing over k from 1 to k_0, the query complexity of all recursive calls with $|U| > n_0$ is at most $2 \cdot n^{1+1/k_0}\left(\log n \cdot 8(\Delta^2+1)\right)^{k_0} \le 2 \cdot n^{1+2/k_0}$.

Therefore, the number of queries for non-edge verification is at most $3 \cdot n^{1+2/k_0}$, which is $n^{1+O\left(\sqrt{(\log\log n + \log\Delta)/\log n}\right)}$. Since it takes at most $\Delta n = n^{1+\log\Delta/\log n}$ queries to verify that $\hat{E} \subseteq E$, we obtain the overall query complexity as stated in Theorem 1.4.2.

[1] We note that there are no recursive calls on the $(k_0+1)^{\text{th}}$ level (i.e., last level) of the recursion with $|U| > n_0$.

Chapter 3

Greedy Algorithms

In this chapter, we first give a greedy verification algorithm using a distance oracle, and then extend it to a reconstruction algorithm using a shortest path oracle.

3.1 Verification via a Distance Oracle

In this section, we prove the following theorem.

Theorem 3.1.1. *If there is an algorithm for graph verification using $f(n, \Delta)$ distance queries, then a greedy algorithm for verification uses $O(\Delta n + \log n \cdot f(n, \Delta))$ distance queries.*

Theorem 3.1.1 implies Theorem 1.4.3, since each query to a distance oracle can be simulated by the same query to a shortest path oracle.

Let $\widehat{NE}$ be the set of the non-edges of $\hat{G}$. For each pair of vertices $(u, v) \in V^2$, we define $S_{u,v} \subseteq \widehat{NE}$ as follows:

$$S_{u,v} = \left\{ ab \in \widehat{NE} : \hat{\delta}(u, a) + \hat{\delta}(b, v) + 1 < \hat{\delta}(u, v) \right\}. \tag{3.1}$$

The following lemmas relate the sets $\{S_{u,v} : (u, v) \in V^2\}$ with non-edge verification.

Lemma 3.1.2. *Assume that $\hat{E} \subseteq E$. For every $(u, v) \in V^2$, if $\delta(u, v) = \hat{\delta}(u, v)$, then every pair $ab \in S_{u,v}$ is a non-edge of G.*

Proof. Consider any pair $ab \in S_{u,v}$. By the triangle inequality, $\delta(u, a) + \delta(a, b) + \delta(b, v) \geq \delta(u, v) = \hat{\delta}(u, v)$. By the definition of $S_{u,v}$ and using $\hat{E} \subseteq E$, we have $\hat{\delta}(u, v) > \hat{\delta}(u, a) + \hat{\delta}(b, v) + 1 \geq \delta(u, a) + \delta(b, v) + 1$. Thus $\delta(a, b) > 1$, i.e., ab is a non-edge of G. □

Lemma 3.1.3. *If a set of queries T verifies that every non-edge of $\hat{G}$ is a non-edge of G, then $\bigcup_{(u,v)\in T} S_{u,v} = \widehat{NE}$.*

Proof. Assume, for a contradiction, that some $ab \in \widehat{NE}$ does not belong to any $S_{u,v}$ for $(u,v) \in T$. Consider adding ab to the set of edges of $\hat{E}$: this will not create a shorter path between u and v, for any $(u,v) \in T$. Thus including ab in $\hat{E}$ is consistent with the answers of all queries in T. This contradicts the assumption that T verifies that ab is a non-edge of G. □

Proof of Theorem 3.1.1. From Lemmas 3.1.2 and 3.1.3, the non-edge verification is equivalent to the SET-COVER problem with the universe $\widehat{NE}$ and the sets $\{S_{u,v} : (u,v) \in V^2\}$. The SET-COVER instance can be solved using the well-known greedy algorithm [52], which gives a $(\ln n + 1)$-approximation. Hence our greedy algorithm for verification (Algorithm 3.1). For the query complexity, first, verifying that $\hat{E} \subseteq E$ takes at most Δn queries, since the graph has maximum degree Δ. The part of non-edge verification uses a number of queries that is at most $(\ln n + 1)$ times the optimum number of queries. □

Algorithm 3.1 Greedy Verification

```
1: procedure VERIFY(Ĝ)
2:     for uv ∈ Ê do
3:         QUERY(u, v)
4:     Y ← ∅
5:     while Ê ∪ Y does not cover all vertex pairs do
6:         choose (u, v) that maximizes |S_{u,v} \ Y|
                                  ▷ S_{u,v} defined in Equation (3.1)
7:         QUERY(u, v)
8:         Y ← Y ∪ S_{u,v}
```

3.2 Reconstruction via a Shortest Path Oracle

In this section, we prove Theorem 1.4.4 that we recall:

Theorem 1.4.4. *If there is an algorithm for graph verification using $f(n,\Delta)$ distance queries, then a greedy algorithm for reconstruction uses $O(\Delta n + \log n \cdot f(n,\Delta))$ shortest path queries.*

Algorithm 3.2 Greedy Reconstruction

1: **procedure** Reconstruct(V)
2: $u_0 \leftarrow$ an arbitrary vertex
3: **for** $u \in V \setminus \{u_0\}$ **do**
4: Query(u, u_0) to get a shortest u-to-u_0 path
5: $X \leftarrow$ the union of the above paths
6: $Y \leftarrow \emptyset$
7: **while** $X \cup Y$ does not cover all vertex pairs **do**
8: choose (u, v) that maximizes $|S^X_{u,v} \setminus Y|$ ▷ $S^X_{u,v}$ defined in Equation (3.2)
9: Query(u, v) to get a shortest u-to-v path
10: **if** $\delta_G(u, v) = \delta_X(u, v)$ **then**
11: $Y \leftarrow Y \cup S^X_{u,v}$
12: **else**
13: $e \leftarrow$ some edge of the above u-to-v path that is not in X
14: $X \leftarrow X \cup \{e\}$
15: **return** X

The algorithm (Algorithm 3.2) constructs an increasing set X of edges so that in the end $X = E$. At any time, the candidate graph is X.[1] Initially, X is the union of the shortest paths given as answers by $n - 1$ queries, so that X is a connected subgraph spanning V. At each subsequent step, the algorithm makes a query that leads either to the confirmation of many non-edges of G, or to the discovery of an edge of G.

Formally, we define, for every pair $(u, v) \in V^2$,

$$S^X_{u,v} = \Big\{ab \in \text{non-edges of } X : \delta_X(u, a) + \delta_X(b, v) + 1 < \delta_X(u, v)\Big\}. \tag{3.2}$$

This is similar to $S_{u,v}$ defined in Equation (3.1). From Lemma 3.1.2, the pairs in $S^X_{u,v}$ can be confirmed as non-edges of G if $\delta_G(u, v) = \delta_X(u, v)$. At each step, the algorithm queries a pair (u, v) that maximizes the size of the set $S^X_{u,v} \setminus Y$. As a consequence, either all pairs in $S^X_{u,v} \setminus Y$ are confirmed as non-edges of G, or $\delta_G(u, v) \neq \delta_X(u, v)$, and in that case, the query reveals an edge along a shortest u-to-v path in G that is not in X; we then add this edge to X.

To see the correctness, we note that the algorithm maintains the invariant that the pairs in X are confirmed edges of G, and that the pairs in Y are confirmed non-edges of G. Thus when $X \cup Y$ covers all vertex pairs, we have $X = E$.

For the query complexity, first, consider the queries that lead to $\delta_G(u, v) \neq$

[1] We identify X with the subgraph induced by the edges of X.

$\delta_X(u,v)$. For each such query, an edge is added to X. This can happen at most $|E| \le \Delta n$ times, because the graph has maximum degree Δ.

Next, consider the queries that lead to $\delta_G(u,v) = \delta_X(u,v)$. Define R to be the set of vertex pairs that are not in $X \cup Y$. We analyze the size of R during the algorithm. For each such query, the size of R decreases by $|S^X_{u,v} \setminus Y|$. To lower bound $|S^X_{u,v} \setminus Y|$, we consider the problem of non-edge verification using a distance oracle on the input graph X, and let T be an (unknown) optimal set of queries. Then $|T| \le f(n,\Delta)$, since there is a verification algorithm using $f(n,\Delta)$ distance queries. By Lemma 3.1.3, the sets $S^X_{u,v}$ for all pairs $(u,v) \in T$ together cover $R \cup Y$, hence R. Therefore, at least one of these pairs satisfies

$$|S^X_{u,v} \setminus Y| \ge |R|/|T| \ge |R|/f(n,\Delta).$$

Initially, $|R| \le n(n-1)/2$, and right before the last query, $|R| \ge 1$, thus the number of queries with $\delta_G(u,v) = \delta_X(u,v)$ is $O(\log n) \cdot f(n,\Delta)$.

Therefore, the overall query complexity is $O(\Delta n + \log n \cdot f(n,\Delta))$.

Remark. *Note that the above proof depends crucially on the fact that $f(n,\Delta)$ is a uniform bound on the number of distance queries for verifying* any *n-vertex graph of maximum degree Δ. Thus, even though the graph X changes during the course of the algorithm because of queries (u,v) such that $\delta_G(u,v) \ne \delta_X(u,v)$, each query (u,v) with $\delta_G(u,v) = \delta_X(u,v)$ confirms $1/f(n,\Delta)$ fraction of non-edges.*

Chapter 4

Decomposition by Separators

In this chapter, we consider special classes of graphs, and we give reconstruction and verification algorithms via a distance oracle. These algorithms all use separators to decompose the graph into subgraphs, and then apply recursion on each subgraph.

4.1 Preliminaries

We review the definition and properties of *separators*, *tree decomposition*, and *chordal graphs*.

Definition 4.1.1. *A subset $S \subseteq V$ is a β-*balanced separator *of the graph $G = (V, E)$ (for $\beta < 1$) if the size of every connected component of $G \setminus S$ is at most $\beta|V|$. In this case, the partition of $G \setminus S$ into connected components is called a β-*balanced partition *of the graph.*

Definition 4.1.2. *A* tree decomposition *of a graph $G = (V, E)$ is a tree T with nodes $n_1, n_2, \ldots, n_\ell$. Node n_i is identified with a* bag *$S_i \subseteq V$, satisfying the following conditions:*

1. *For every vertex v in G, the nodes whose bags contain v form a connected subtree of T.*
2. *For every edge uv in G, some bag contains both u and v.*

The width *of a tree decomposition is the size of the largest bag minus 1, and the* treewidth *of G is the minimum width over all possible tree decompositions of G.*

Lemma 4.1.3 ([71])**.** *Let G be a graph of treewidth k. Any tree decomposition of width k contains a bag $S \subseteq V$ that is a $(1/2)$-balanced separator of G.*

Lemma 4.1.4 ([18]). *Let G be a chordal graph. Then G has a tree decomposition such that every bag is a maximal clique*[1] *and every maximal clique appears exactly once in this decomposition.*

From Lemmas 4.1.3 and 4.1.4, we have:

Corollary 4.1.5. *Let G be a chordal graph of maximum degree Δ. Then G has treewidth at most Δ, and there exists a clique $S \subseteq V$ of size at most $\Delta + 1$ that is a $(1/2)$-balanced separator of G.*

Definition 4.1.6. *A subset of vertices $U \subseteq V$ is said to be* self-contained *if, for every pair of vertices $(x, y) \in U^2$, any shortest path in G between x and y goes through nodes only in U.*

4.2 Reconstruction of Chordal Graphs

In this section, we prove Theorem 1.4.6 that we recall:

Theorem 1.4.6. *For reconstruction of chordal graphs using a distance oracle, there is a randomized algorithm with query complexity $O\Big(\Delta^3 2^\Delta \cdot n(2^\Delta + \log^2 n) \log n\Big)$, which is $\tilde{O}(n)$ when the maximum degree Δ is $O(\log \log n)$.*

The algorithm (Algorithm 4.3) computes a vertex that is on many shortest paths in the sampling, and then grows a clique separator including this vertex. Next, it partitions the graph into subgraphs with respect to this separator, and then recursively reconstructs each subgraph. The main tools we need is computing a shortest path between a pair of vertices (Section 4.2.1) and partitioning the graph with respect to a set of vertices (Section 4.2.2). In what follows, the set U represents the set of vertices for which we are currently reconstructing the induced subgraph $G[U]$.

4.2.1 Subroutine: Computing a Shortest Path

In this section, we prove the following lemma.

Lemma 4.2.1. *Let U be a self-contained subset of V. Let a and b be vertices in U. The function* Shortest-Path(U, a, b) *(Algorithm 4.1) outputs a shortest path between a and b in $G[U]$. Its query complexity is $O(|U| \log |U|)$.*

The algorithm is based on dichotomy. First, it makes $2|U|$ queries to get $\delta(u, a)$ and $\delta(u, b)$ for every $u \in U$. Let c be the middle node of some shortest a-to-b path.

[1] A maximal clique is a clique which is not contained in any other clique.

Algorithm 4.1 Finding a Shortest Path (see Lemma 4.2.1)

```
 1: function SHORTEST-PATH(U, a, b)
 2:     if δ(a, b) > 1 then
 3:         QUERY(a, U); QUERY(b, U)
 4:         T ← {v ∈ U | δ(v, a) + δ(v, b) = δ(a, b)}
 5:         l ← ⌊δ(a, b)/2⌋
 6:         c ← an arbitrary node in T such that δ(c, a) = ℓ
 7:         U_1 ← {v ∈ T | δ(v, a) < ℓ}
 8:         U_2 ← {v ∈ T | δ(v, a) > ℓ}
 9:         P_1 ← SHORTEST-PATH(U_1, a, c)
10:         P_2 ← SHORTEST-PATH(U_2, c, b)
11:         return the concatenation of P_1 and P_2
12:     else
13:         return the path of a single edge ab
```

Then the algorithm recursively computes a shortest a-to-c path and a shortest c-to-b path. The concatenation of these two paths is a shortest a-to-b path.

During the recursion, the distance between the two given endpoints is reduced by half each time. So there are $O(\log |U|)$ levels of the recursion. The total number of queries at every level is $O(|U|)$, since the sets on the same level of the recursion are disjoint. Therefore, the overall query complexity is $O(|U| \log |U|)$.

4.2.2 Subroutine: Partitioning by a Set

In this section, we prove the following lemma.

Lemma 4.2.2. *Let U be a self-contained subset of V. Let S be a subset of U. The function* PARTITION(U, S) *(Algorithm 4.2) outputs the partition of $G[U] \setminus S$ into connected components. Its query complexity is $O(\Delta |S| \cdot |U|)$.*

Let $W = (N(S) \cap U) \setminus S$. For every $a \in W$, define the *cluster* of a as:

$$B(a) = \{x \in U \setminus S \mid \delta(a, x) \leq \delta(S, x)\}. \tag{4.1}$$

Since U is self-contained, every $x \in U \setminus S$ belongs to some cluster $B(a)$. The clusters may have overlaps. The PARTITION algorithm successively merges two clusters with overlaps. See Figure 4.1.

The query complexity of the algorithm is $O(|N(S)| \cdot |U|) = O(\Delta |S| \cdot |U|)$. Lemma 4.2.2 then follows directly from Lemmas 4.2.3 and 4.2.4.

Lemma 4.2.3. *Let C be a connected component in $G[U] \setminus S$. Then $C \subseteq B$ for some set B in the output of the algorithm.*

Algorithm 4.2 Computing the Partition (See Lemma 4.2.2)

```
1: function PARTITION(U, S)
2:     QUERY(S, U) and obtain N(S) ∩ U
3:     QUERY(N(S) ∩ U, U)
4:     W ← (N(S) ∩ U) \ S
5:     𝓑 ← {B(a) | a ∈ W}                    ▷ B(a) defined in Equation (4.1)
6:     while ∃ B_1, B_2 ∈ 𝓑 s.t. B_1 ∩ B_2 ≠ ∅ do
7:         merge B_1 and B_2 in 𝓑
8:     return 𝓑
```

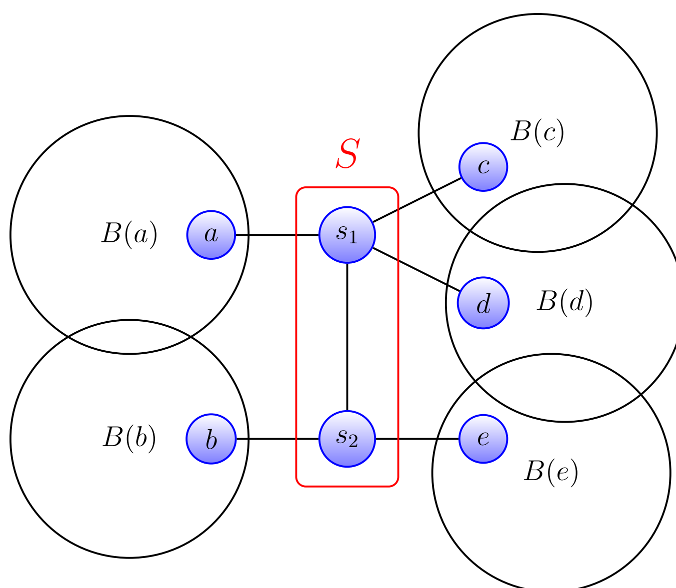


Figure 4.1: Here $S = \{s_1, s_2\}$ and $W = \{a, b, c, d, e\}$. The clusters $B(a)$, $B(b)$, $B(c)$, $B(d)$, $B(e)$ are indicated by the circles. Using their overlaps, the algorithm produces the partition $\mathcal{B} = \{B(a) \cup B(b)\,,\, B(c) \cup B(d) \cup B(e)\}$.

Proof. Let A be the set of vertices in $C \cap W$. Since U is self-contained, for every vertex $x \in C$, there exists some $a \in A$ such that $x \in B(a)$. Thus we only need to prove that all sets $\{B(a) : a \in A\}$ are eventually merged in our algorithm.

Define a weighed graph H whose vertex set is A, and such that for every $(a, b) \in A^2$, there is an edge ab in H, whose weight is defined as the distance between a and b in $G[C]$[2]. To show that all sets $\{B(a) : a \in A\}$ are eventually merged, we use an inductive proof that is in the same order that Prim's algorithm would construct a minimum spanning tree on H. Recall that Prim's algorithm

[2]This distance may be larger than $\delta(a, b)$.

initializes a tree $\mathcal{T}$ with a single vertex, chosen arbitrarily from A. Then it repeatedly chooses an edge $ab \in \mathcal{T} \times (A \setminus \mathcal{T})$ with minimum weight and add ab to $\mathcal{T}$. We will show that if an edge ab is added to $\mathcal{T}$, then $B(a)$ and $B(b)$ must be merged in our algorithm. Since Prim's algorithm finishes by providing a spanning tree including every $a \in A$, we thus have all sets $B(a)$ for $a \in A$ are merged in our algorithm.

Suppose that the i unions corresponding to the first i edges chosen by Prim's algorithm have been performed already, for $i \geq 0$. Let $\mathcal{T}$ be the tree in H after adding the first i edges.[3] Let ab be the $(i+1)^{\text{th}}$ edge chosen by Prim's algorithm. Thus $a \in \mathcal{T}$, $b \in A \setminus \mathcal{T}$, and weight$(ab)$ is minimized. Consider a shortest path $p_1, \ldots, p_k$ in $G[C]$ between a and b. Let $z = p_{\lceil k/2 \rceil}$ be the mid-point vertex of the path. We claims that both $B(a)$ and $B(b)$ contain z, thus $B(a)$ and $B(b)$ are merged in our algorithm. It is easy to see that $p_1, \ldots, p_{\lceil k/2 \rceil}$ and $p_{\lceil k/2 \rceil}, \ldots, p_k$ are shortest paths in G. Thus $\delta(a,z) = \lceil k/2 \rceil - 1$ and $\delta(b,z) = \lfloor k/2 \rfloor$. So we have $\delta(a,z) \leq \delta(b,z) \leq \delta(a,z) + 1$. To show $z \in B(a)$ and $z \in B(b)$, we only need to show that $\delta(b,z) \leq \delta(S,z)$. Choose the vertex $s \in S$ that minimizes $\delta(s,z)$ and consider a shortest z-to-s path P. Let c be the neighbor of s on P, and let P' be the shortest z-to-c path. We note that $c \in A$ and P' is in $G[C]$. Since $\delta(S,z) = \delta(s,z) = \delta(c,z)+1$, we only need to show that $\delta(b,z) \leq \delta(c,z)+1$. There are 2 cases:

Case 1: $c \in A \setminus \mathcal{T}$. Then the concatenation of $p_1, \ldots, p_{\lceil k/2 \rceil}$ and P' gives a path in $G[C]$ between a and c of length $\delta(a,z) + \delta(c,z)$, which is at least weight(ac) by the definition of the weight. From the choice of ab, weight$(ac) \geq$ weight$(ab) = \delta(a,z) + \delta(b,z)$. So we have $\delta(b,z) \leq \delta(c,z)$.

Case 2: $c \in \mathcal{T}$. Similarly, the concatenation of $p_k, p_{k-1}, \ldots, p_{\lceil k/2 \rceil}$ and P' gives a path in $G[C]$ between b and c of length $\delta(b,z) + \delta(c,z)$, which is at least weight(bc) by the definition of the weight. From the choice of ab, weight$(bc) \geq$ weight$(ab) = \delta(a,z)+\delta(b,z)$. So we have $\delta(a,z) \leq \delta(c,z)$. Thus $\delta(b,z) \leq \delta(a,z)+1 \leq \delta(c,z)+1$.

This completes the proof. □

Lemma 4.2.4. *Let B be a set in the output of the algorithm. Then $B \subseteq C$ for some connected component C in $G[U] \setminus S$.*

Proof. First we show that for every $a \in W$ and every $x \in B(a)$, a and x belong to the same component in $G[U] \setminus S$. Suppose there exists some $x \in B(a)$, such that x and a belong to different components in $G[U] \setminus S$. Any shortest path from a to x must pass through the separator S, so we have $\delta(a,x) \geq \delta(a,S) + \delta(S,x) = 1 + \delta(S,x)$. Contradiction with $x \in B(a)$.

Next we prove an invariant on $\mathcal{B}$ during the **while** loop (Line 6): *Every set $B \in \mathcal{B}$ is a subset of some component of $G[U] \setminus S$.* This invariant holds before the

[3]For the base case ($i = 0$), $\mathcal{T}$ contains a single vertex and no union operation is performed.

while loop starts. Suppose the invariant holds before the ith iteration of the **while** loop, and in this iteration $B_1, B_2 \in \mathcal{B}$ get merged. Since $B_1 \cap B_2 \neq \emptyset$, there exists $z \in B_1 \cap B_2$. All nodes in B_1 (resp. in B_2) are in the same component as z. Thus all nodes in $B_1 \cup B_2$ are in the same component as z. By induction, the invariant holds when the **while** loop terminates. This completes the proof. □

4.2.3 Algorithm and Analysis

The Reconstruct-Chordal algorithm (Algorithm 4.3) takes as input a self-contained subset $U \subseteq V$ of a chordal graph and returns the edge set of $G[U]$. The key function Balanced-Partition-Chordal finds a *β-balanced partition* of U. This function first computes a vertex that is on many shortest paths in the sampling, and then looks for a β-balanced clique separator including this vertex. It repeatedly takes samples until a β-balanced partition is found. We set $n_0 = 2^{\Delta+2}(\Delta+1)^2$; $C_1 = 36(\Delta+1)^2 \log |U|$; and

$$\beta = \max\left(1 - 1/(\Delta \cdot 2^{\Delta+1}), \sqrt{1 - 1/(4(\Delta+1))}\right).$$

Lemma 4.2.5. Reconstruct-Chordal(U) *returns the edge set of* $G[U]$.

Proof. By Lemma 4.2.2, $\{U_i\}_i$ is the partition of $G[U] \setminus K$ into connected components. Every edge of $G[U]$ belongs to some $G[U_i \cup K]$, since there is no edge between different U_i and U_j. So the edge set of $G[U]$ is the union of the edge sets of $G[U_i \cup K]$ over i. The statement follows by induction. □

To bound the query complexity, the key is the following lemma.

Lemma 4.2.6. *In every* ***repeat*** *loop of the function* Balanced-Partition-Chordal, *a β-balanced partition is found with probability at least* $2/3$.

To prove Lemma 4.2.6, we need Lemmas 4.2.7 and 4.2.8.

Lemma 4.2.7. *For every $v \in U$, let p_v denote the fraction of pairs $(a, b) \in U^2$ such that v is on some shortest path between a and b. Then* $\max_v p_v \geq 1/(2(\Delta+1))$.

Proof. By Corollary 4.1.5, there is some clique $S \subseteq U$ of size at most $\Delta + 1$ such that every connected component in $G[U] \setminus S$ has size at most $|U|/2$. Notice that for any pair of vertices a, b not from the same component, any shortest a-to-b path must go by some node in S. The number of such pairs is at least $|U|^2/2$. By Pigeonhole Principle, there exists some $z \in S$, such that for at least $1/|S| \geq 1/(\Delta+1)$ fraction of these pairs, their shortest paths go by z. Thus $p_z \geq 1/(2(\Delta+1))$. □

Algorithm 4.3 Reconstruction of Chordal Graphs (see Theorem 1.4.6)

```
1: procedure RECONSTRUCT-CHORDAL(U)
2:     if |U| > n_0 then
3:         ({U_i}_i, K) ←BALANCED-PARTITION-CHORDAL(U)
4:         return ⋃_i RECONSTRUCT-CHORDAL(U_i ∪ K)
5:     else
6:         return EXHAUSTIVE-QUERY(U)

7: function BALANCED-PARTITION-CHORDAL(U)
                                          ▷ outputs a β-balanced partition of U
8:     repeat
9:         {(a_i, b_i)}_{1≤i≤C_1} ← uniform and independent random pairs from U
10:        for i ← 1 to C_1 do
11:            P_i ← SHORTEST-PATH(U, a_i, b_i)              ▷ see Section 4.2.1
12:        x ← the node in U with the most occurrences among {P_i}_i
13:        QUERY(x, U) and obtain N(x) ∩ U
14:        QUERY(N(x) ∩ U, N(x) ∩ U) and obtain all cliques in U containing x
15:        for every clique K in U containing x do
16:            𝒫 ← PARTITION(U, K)                           ▷ See Algorithm 4.2
17:            if 𝒫 is β-balanced then return (𝒫, K)
18:    until a β-balanced partition is found
```

Lemma 4.2.8. *For every vertex $v \in U$, let $\tilde{p}_v$ denote the fraction of pairs (a_i, b_i) among $C_1 \log |U|$ uniform and independent random pairs from U such that v is on some shortest path between a_i and b_i. Let $x = \arg\max_v \tilde{p}_v$. If $C_1 \geq 9/(\max_v p_v)^2 \log |U|$, then with probability at least $2/3$, we have $p_x > (\max_v p_v)/2$.*

Proof. Let z be the node in U which maximizes p_z. We will show that $\mathbb{P}[\tilde{p}_y \geq \tilde{p}_z] < \frac{1}{3|U|}$ for any node y with $p_y \leq p_z/2$. This implies the lemma statement because we then have $\mathbb{P}[\exists y \in U, \text{ s.t. } p_y \leq p_z/2 \text{ and } \tilde{p}_y \geq \tilde{p}_z] < \frac{1}{3}$. Thus with probability at least $2/3$, the node x with the largest $\tilde{p}_x$ satisfies $p_x > p_z/2$.

Let y be any node with $p_y \leq p_z/2$. For every $i \leq C_1$, define a variable $Y_i \in \{0, 1\}$ such that $Y_i = 1$ if the node y is on a shortest a_i-to-b_i path and $Y_i = 0$ otherwise. Since $\{(a_i, b_i)\}_i$ are uniform and independent random pairs from U, $\{Y_i\}_i$ are independent random variables, and each Y_i equals 1 with probability p_y. We then have $\mathbb{E}[Y_i] = p_y \leq p_z/2$. Similarly, define a variable $Z_i \in \{0, 1\}$ such that $Z_i = 1$ if the node z is on a shortest a_i-to-b_i path and $Z_i = 0$ otherwise. Then $\mathbb{E}[Z_i] = p_z$.

For every i, define $T_i = Y_i - Z_i$. Let $\overline{T}$ be the average of all T_i's. Then $\mathbb{E}[\overline{T}] = \mathbb{E}[T_i] = \mathbb{E}[Y_i] - \mathbb{E}[Z_i] \leq -p_z/2$. We have

$$\mathbb{P}\left[\tilde{p}_y \geq \tilde{p}_z\right] = \mathbb{P}\left[\sum_i T_i \geq 0\right] \leq \mathbb{P}\left[\left|\overline{T} - \mathbb{E}[\overline{T}]\right| \geq p_z/2\right].$$

By Hoeffding's inequality, the last term is at most $2 \cdot \exp(-p_z^2 \cdot C_1/8)$, which is at most $1/(3|U|)$ by the definition of p_z and C_1. Therefore, we have $\mathbb{P}\left[\tilde{p}_y \geq \tilde{p}_z\right] < 1/(3|U|)$. □

Proof of Lemma 4.2.6. By Lemma 4.1.4, there is a *tree decomposition* T of $G[U]$ such that every bag of T is a unique maximal clique of $G[U]$. Let x be the node computed in Line 12 of Algorithm 4.3. Let T_x be the subtree of T induced by the bags containing x. Define F to be the forest after removing T_x from T. For any subgraph H of T, define $V(H) \subseteq U$ to be the set of vertices that appear in at least one bag of H.

Case 1: There exists some connected component T' in F with $(1-\beta)|U| \leq |V(T')| \leq \beta|U|$. Consider the (unique) edge K_1K_2 in T such that $K_1 \in T_x$ and $K_2 \in T'$. $K_1 \cap K_2$ is a β-balanced separator, since $V(T')$ is a component in $G[U] \setminus (K_1 \cap K_2)$. Thus $K_1 \supseteq K_1 \cap K_2$ is also a β-balanced separator. Since K_1 contains x, K_1 is one of the cliques checked on Line 15. The algorithm succeeds by finding a β-balanced separator.

Case 2: There exists some connected component T' in F with $|V(T')| > \beta|U|$. The algorithm then fails to find a β-balanced separator. We bound the probability of this case by at most 1/3. Again let K_1K_2 be the edge in T such that $K_1 \in T_x$ and $K_2 \in T'$. For any vertices $u, v \in V(T')$, any shortest u-to-v path cannot go by x. Since there are at least β^2 fraction of such pairs in U^2, we have $p_x \leq 1 - \beta^2$, which is at most $1/(4(\Delta+1))$ by the definition of β. This happens with probability at most 1/3 by Lemmas 4.2.7 and 4.2.8.

We argue that the two cases above are exhaustive. Suppose, for the sake of contradiction, that every component T' in F is such that $|V(T')| < (1-\beta)|U|$. The number of components in F is at most $\Delta \cdot 2^{\Delta}$, because every component has a bag that contains a neighbor of x, and all bags are unique. So $|V(F)| < \Delta \cdot 2^{\Delta} \cdot (1-\beta)|U|$, which is at most $|U|/2$ by the definition of β. On the other hand, every node $v \in U \setminus N(x)$ is covered by some clique in F, so $|V(F)| \geq |U| - (\Delta+1)$, which is greater than $|U|/2$ since $|U| > n_0$. Contradiction. □

Query Complexity Analysis. First, we analyze the complexity of Balanced-Partition-Chordal. Computing C_1 shortest paths takes $O(\Delta^2|U|\log^2|U|)$ queries, since a shortest path between two given nodes can be computed using $O(|U|\log|U|)$ queries (see Section 4.2.1). We note that the neighborhood $N(x)$ of x has size at most $\Delta+1$, and there are at most 2^{Δ} cliques containing x. For every clique K containing x, Partition(U, K) takes $O(\Delta|K| \cdot |U|)$ queries by Lemma 4.2.2,

where $|K| \leq \Delta + 1$. Therefore every **repeat** loop takes $O\left(\Delta^2 |U| (2^\Delta + \log^2 |U|)\right)$ queries. By Lemma 4.2.6, the expected number of **repeat** loops is constant. So the query complexity is $O\left(\Delta^2 |U| (2^\Delta + \log^2 |U|)\right)$.

Next, we analyze the complexity of RECONSTRUCT-CHORDAL(U). Let $q(m)$ be the number of queries when $|U| = m$. We have

$$q(m) = O\left(\Delta^2 m (2^\Delta + \log^2 m)\right) + \sum_i q(|U_i| + |K|),$$

where $\{U_i\}_i$ is a β-balanced partition of U. Hence

$$q(n) = O\left(\Delta^2 n (2^\Delta + \log^2 n) \log_{\frac{1}{\beta}} n\right) = O\left(\Delta^3 2^\Delta \cdot n (2^\Delta + \log^2 n) \log n\right).$$

This completes the proof of Theorem 1.4.6.

4.3 Reconstruction of Outerplanar Graphs

In this section, we prove Theorem 1.4.7 that we recall:

Theorem 1.4.7. *For reconstruction of outerplanar graphs using a distance oracle, there is a randomized algorithm with query complexity $O(\Delta^2 \cdot n \log^3 n)$, which is $\tilde{O}(n)$ when the maximum degree $\Delta = O(\mathrm{polylog}\, n)$.*

The algorithm again uses random sampling and statistic estimation, as used for reconstructing chordal graphs in Section 4.2. To obtain a balanced partition of an outerplanar graph, we need to partition the graph with respect to a *polygon* (Section 4.3.1).

4.3.1 Subroutine: Partitioning by a Polygon

Definition 4.3.1. *We say that the k-tuple $(x_1, \ldots, x_k) \in V^k$ (where $k \geq 3$) forms a* polygon *if $G[\{x_1, \ldots, x_k\}]$ has exactly k edges: $x_1x_2, x_2x_3, \ldots, x_kx_1$.*

In this section, we prove the following lemma.

Lemma 4.3.2. *Let U be a self-contained subset of V. Let $a, b, c \in U$ be consecutive nodes along some unknown polygon $(q_1, \ldots, q_l)$ in U. The function* PARTITION-BY-POLYGON(U, a, b, c) *outputs the partition of $U \setminus \{q_1, \ldots, q_l\}$ into connected components. Its query complexity is $O(\Delta |U| \log |U|)$.*

The function PARTITION-BY-POLYGON consists of the following two steps.

Step 1: Computing the Polygon. The algorithm is given in Algorithm 4.4. The key is to compute the a-to-c path along the polygon that does not go through b. First, the algorithm computes the middle point z of this path. Next, it computes a shortest path P_1 between a and z and a shortest path P_2 between z and c using $O(|U|\log|U|)$ queries (see Section 4.2.1). The polygon is the concatenation of the path P_1, the path P_2, the edge cb, and the edge ba. In the algorithm, PARTITION$(U, \{a, b\})$ and PARTITION$(U, \{b, c\})$ use $O(\Delta \cdot |U|)$ queries. So the overall query complexity of FIND-POLYGON is $O(\Delta \cdot |U|\log|U|)$.

Algorithm 4.4 Finding a Polygon

```
 1: procedure FIND-POLYGON(U, a, b, c)
 2:     (A_1, ..., A_{k_1}) ←PARTITION(U, {a, b})
 3:     Let A_i be the component containing c
 4:     (C_1, ..., C_{k_2}) ←PARTITION(U, {b, c})
 5:     Let C_j be the component containing a
 6:     T ← (A_i ∩ C_j) ∪ {a, c}
 7:     QUERY(a, T), QUERY(c, T)
 8:     d ← min_{u∈T}{δ(a, u) + δ(u, c)}
 9:     Let z ∈ T be such that δ(a, z) + δ(z, c) = d and δ(a, z) = ⌊d/2⌋
10:     P_1 ← SHORTEST-PATH(T, a, z)
11:     P_2 ← SHORTEST-PATH(T, z, c)
12:     return the concatenation of P_1, P_2, cb, ba
```

Step 2: Partitioning by the Polygon. Given a polygon $(q_1, \ldots, q_l)$, we want to compute the partition of U with respect to this polygon. We note that the PARTITION algorithm in Section 4.2.2 requires $O(\Delta \cdot l \cdot |U|)$ queries, which is $O(\Delta \cdot |U|^2)$ when $l = \Theta(|U|)$. In the following, we give an improved implementation that uses $O(\Delta \cdot |U|\log|U|)$ queries based on dichotomy.

Let $m = \lfloor l/2 \rfloor$. First we compute the partition of U into components with respect to the set $\{q_1, q_m, q_l\}$ using the PARTITION procedure. This takes $O(\Delta \cdot |U|)$ queries. Let Q_1 be the component containing q_2 (the endpoints q_1, q_m are included), and let Q_2 be the component containing q_{m+1} (the endpoints q_m, q_l are included). It is easy to see that both Q_1 and Q_2 are self-contained. We further decompose Q_1 with respect to the path $q_1 q_2 \ldots q_m$ and decompose Q_2 with respect to the path $q_m q_{m+1} \ldots q_l$, using a recursive procedure PARTITION-BY-PATH (Algorithm 4.5). This procedure receives a self-contained subset $Z \subseteq U$ and two integers s, t such that $1 \le s < t \le l$, and returns the partition of Z by the path $q_s q_{s+1} \ldots q_t$.

The number of queries of PARTITION$(Z, \{q_m\})$ or PARTITION$(Z, \{q_s, q_t\})$ is $O(\Delta \cdot |Z|)$. During the recursion, $(t - s)$ is reduced by half at every level, so there

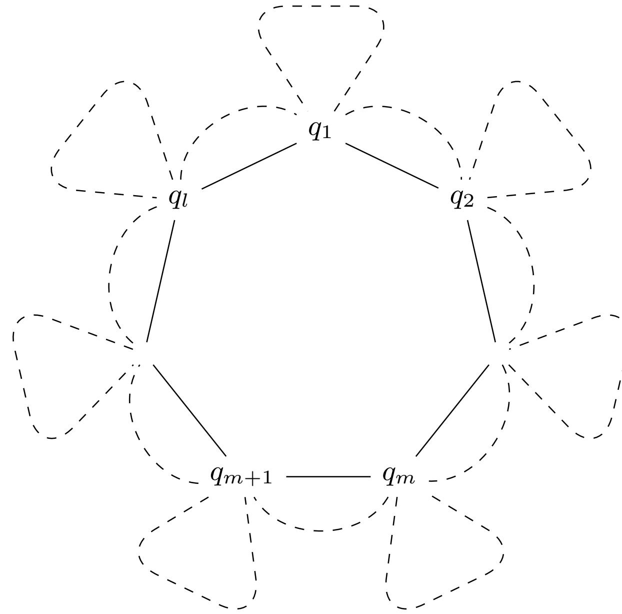


Figure 4.2: Partition by the polygon $q_1, \ldots, q_l$

Algorithm 4.5 Partition with respect to a path

```
1: procedure PARTITION-BY-PATH(Z, s, t)
2:     if t > s + 1 then
3:         m ← ⌊(s + t)/2⌋
4:         P ← PARTITION(Z, {q_m})
5:         Let Z_1 be the component in P containing q_s
6:         Let Z_2 be the component in P containing q_t
7:         P_1 ← PARTITION-BY-PATH(Z_1, s, m)
8:         P_2 ← PARTITION-BY-PATH(Z_2, m, t)
9:         return (P \ {Z_1, Z_2}) ∪ P_1 ∪ P_2
10:    else
11:        return PARTITION(Z, {q_s, q_t})
```

are at most $\log |U|$ levels of the recursion. The query complexity of each level is $O(\Delta \cdot |U|)$, since the sets Z's in the that level are disjoint (except at endpoints). So both PARTITION-BY-PATH$(Q_1, 1, m)$ and PARTITION-BY-PATH(Q_2, m, l) take $O(\Delta \cdot |U| \log |U|)$ queries. Therefore, the overall query complexity to partition U with respect to the polygon is $O(\Delta \cdot |U| \log |U|)$.

4.3.2 Algorithm and Analysis

The Reconstruct-Outerplanar algorithm (Algorithm 4.6) takes as input a self-contained subset $U \subseteq V$ of an outerplanar graph and returns the edge set of $G[U]$. Similar to Section 4.2, the function Balanced-Partition-Outerplanar computes a *β-balanced partition* of U. We set $n_0 = 20$; $C_1 = 324 \log |U|$, and $\beta = \sqrt{11/12}$.

Algorithm 4.6 Reconstruction of Outerplanar Graphs

```
 1: procedure Reconstruct-Outerplanar(U)
 2:     if |U| > n_0 then
 3:         (U_1, ..., U_ℓ) ← Balanced-Partition-Outerplanar(U)
 4:         return ⋃_i Reconstruct-Outerplanar(U_i)
 5:     else
 6:         return Exhaustive-Query(U)

 7: function Balanced-Partition-Outerplanar(U)
 8:     repeat
 9:         {(a_i, b_i)}_{1≤i≤C_1} ← uniform and independent random pairs from U
10:         for i ← 1 to C_1 do
11:             P_i ← Shortest-Path(U, a_i, b_i)              ▷ see Section 4.2.1
12:         x ← the node in U with the most occurrences among {P_i}_i
13:         Query(x, U) and obtain N(x) ∩ U
14:         𝒫 ← Partition(U, N(x) ∩ U)                       ▷ See Algorithm 4.2
15:         if 𝒫 is β-balanced then return 𝒫                  ▷ Figure 4.3
16:         Let W ∈ 𝒫 be the component with more than β|U| nodes
17:         if N(W) contains two neighbors of x (let them be y, y') then
18:             𝒫 ← Partition-by-Polygon(U, y, x, y')
19:             if 𝒫 is β-balanced then return 𝒫
20:     until a β-balanced partition is found
```

Correctness of the Reconstruct-Outerplanar algorithm is a trivial adaptation from Lemma 4.2.5. To bound the query complexity, the key is the following lemma.

Lemma 4.3.3. *In every **repeat** loop of the function* Balanced-Partition-Outerplanar*, a β-balanced partition is found with probability at least* $2/3$.

To prove Lemma 4.3.3, we need Lemma 4.3.4.

Lemma 4.3.4. *For every $v \in U$, let p_v denote the fraction of pairs $(a, b) \in U^2$ such that v is on some shortest path between a and b. Then* $\max_v p_v \geq 1/6$.

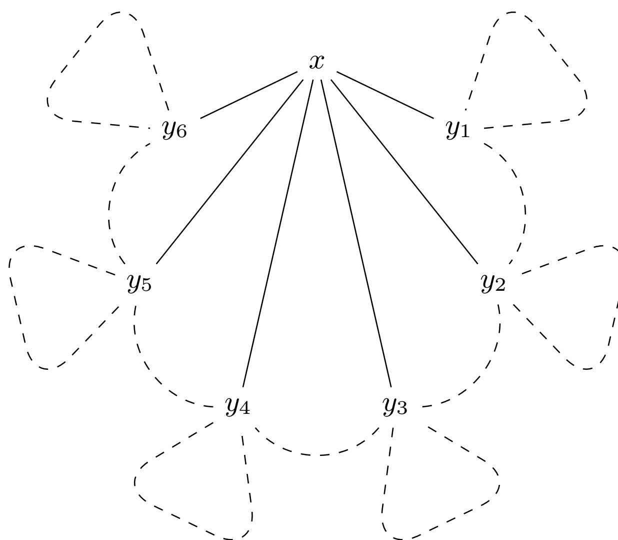


Figure 4.3: Partition by the neighbors $y_1, \ldots, y_6$ of x.

Proof. Since $G[U]$ is outerplanar, it has treewidth at most 2. From Lemma 4.1.3, there exists some set $S \subseteq U$ of size at most 3 such that every connected component in $G[U] \setminus S$ has size at most $|U|/2$. The following argument is similar to that in the proof of Lemma 4.2.7. Notice that for any pair of vertices a, b not from the same component, any shortest a-to-b path must go by some node in S. The number of such pairs is at least $|U|^2/2$. By Pigeonhole Principle, there exists some $z \in S$, such that for at least $1/|S| \geq 1/3$ fraction of these pairs, their shortest paths go by z. Thus $p_z \geq 1/6$. □

Proof of Lemma 4.3.3. Let x be the node computed in Line 12 of Algorithm 4.6. If the partition by the neighbors (Line 14) is β-balanced or the partition by the polygon (Line 18) is β-balanced, then the algorithm succeeds. We only need to bound the probability of the remaining case by 1/3. If no β-balanced partition is found, there must be a self-contained subset S of U such that $x \notin S$ and $|S| \geq \beta|U|$. For every $(a, b) \in S^2$, any shortest path between a and b does not go by x. So $p_x \leq 1 - (|S|/|U|)^2 \leq 1 - \beta^2 = 1/12$ by the definition of β. This happens with probability at most 1/3, by Lemmas 4.3.4 and 4.2.8. So a β-balanced partition is found with probability at least 2/3. □

Query Complexity Analysis. First, we analyze the complexity of Balanced-Partition-Outerplanar. Computing C_1 shortest paths takes $O(|U| \log^2 |U|)$ queries, since a shortest path between two given nodes can be computed using $O(|U| \log |U|)$ queries (see Section 4.2.1). We note that the neighborhood $N(x)$ of x

has size at most $\Delta+1$. By Lemma 4.2.2, Partition$(U, N(x))$ takes $O(\Delta|N(x)|\cdot|U|)$ queries. The procedure Partition-by-Polygon takes $O(\Delta|U|\log|U|)$ queries. Therefore every **repeat** loop takes $O\left(\Delta^2|U|\log^2|U|\right)$ queries. From Lemma 4.3.3, the expected number of **repeat** loops is constant. So the overall query complexity is $O\left(\Delta^2|U|\log^2|U|\right)$.

Next, we analyze the complexity of Reconstruct-Outerplanar. Let $q(m)$ be the number of queries when $|U| = m$. We have

$$q(m) = O\left(\Delta^2 m \log^2 m\right) + \sum_i q(|U_i|),$$

where $\{U_i\}_i$ is a β-balanced partition of U. Hence $q(n) = O\left(\Delta^2 n \log^3 n\right)$.

Thus we complete the proof of Theorem 1.4.7.

4.4 Verification of Treewidth Bounded Graphs

In this section, we prove Theorem 1.4.8 that we recall:

Theorem 1.4.8. *For verification of graphs of treewidth w using a distance oracle, there is a deterministic algorithm with query complexity $O(\Delta(\Delta + w\log n)n\log n)$, which is $\tilde{O}(n)$ when Δ and w are $O(\text{polylog}\, n)$.*

We only need to provide an algorithm for *non-edge verification*, because verifying that $\hat{E} \subseteq E$ can be done naively. The algorithm is by recursion. It first computes a $(1/2)$-balanced separator in $\hat{G}$ and use it to obtain a partition of the vertices. Then it verifies the non-edges of G between different components in the partition. Finally, it recurses to verify the non-edges inside each component.

There is a catch because of the query oracle: by querying a pair (u, v), we would like to get back their distance in the recursive subgraph H, but instead the oracle returns their distance in the entire graph G. It could well be that a shortest u-to-v path in G goes through two nodes s_1 and s_2 in the separator and the segment between s_1 and s_2 is outside H.

As a warmup, we first provide an algorithm for the special case of chordal graphs, because the above issue does not arise when the graph is chordal.[4] We then extend the algorithm to graphs of bounded treewidth: To get around that issue, we formulate the recursive subproblem by augmenting H and adding weighted edges between vertices of the separator.

[4]Since the separator is a clique, the shortest s_1-to-s_2 path is an edge, and thus belongs to H.

4.4.1 Warm up: Chordal Graphs

The Verify-Chordal algorithm (Algorithm 4.7) receives as input a chordal graph $\hat{G} = (V, \hat{E})$ such that $\hat{E} \subseteq E$ and a self-contained subset $U \subseteq V$, and verifies whether every non-edge of $\hat{G}[U]$ is a non-edge of $G[U]$.

Algorithm 4.7 Recursive Verification for Chordal Graphs

```
1: procedure Verify-Chordal(Ĝ, U)
2:     if |U| > 4(Δ + 1) then
3:         S ← (1/2)-balanced clique separator of Ĝ[U] of size at most Δ + 1
4:         Query(S, U) and obtain N(S) ∩ U
5:         Query(N(S) ∩ U, U)
6:         for every component C of Ĝ[U] \ S do
7:             Verify-Chordal(Ĝ, C ∪ S)
8:     else
9:         Query(U, U)
```

By Corollary 4.1.5, there is a (1/2)-balanced clique separator S of $\hat{G}[U]$.[5] To confirm the non-edges between different components of $\hat{G}[U] \setminus S$, it is sufficient to query every pair in $(N(S) \cap U) \times U$. This is shown in Lemma 4.4.1, which is a main idea of the algorithmic design. Then for each component C of $\hat{G}[U] \setminus S$, we recursively verify the non-edges inside $\hat{G}[C \cup S]$. The recursive call on the subset $C \cup S$ still uses the global query oracle. But because S is a clique in G, for any $u, v \in C \cup S$, any shortest u-to-v path in G stays inside $C \cup S$, so the value returned by Query(u, v) is the distance in $G[C \cup S]$.

Lemma 4.4.1. *Assume that $\hat{E} \subseteq E$. If $\delta(u, v) = \hat{\delta}(u, v)$ for every $(u, v) \in (N(S) \cap U) \times U$, then there is no edge in $G[U]$ between different components of $\hat{G}[U] \setminus S$.*

Proof. Let X and Y be any two different components in the partition of $\hat{G}[U] \setminus S$. Let x be any vertex in X and y be any vertex in Y. We show that xy is not an edge in $G[U]$. Let a (resp. b) be the vertex in $N(S)$ that is closest to x (resp. y) in $\hat{G}[U]$. Then $a \in X$ and $b \in Y$. Since $\hat{E} \subseteq E$, we have $\hat{N}(S) \subseteq N(S)$. It is then easy to see that $a, b \in (N(S) \cap U) \setminus S$. Without loss of generality, assume $\delta(a, x) \leq \delta(b, y)$.

Since $(a, y) \in (N(S) \cap U) \times U$, we have $\delta(a, y) = \hat{\delta}(a, y)$. Any shortest path in $\hat{G}[U]$ from a to y goes through S, so

$$\hat{\delta}(a, y) \geq \hat{\delta}(a, S) + \hat{\delta}(S, y) = \hat{\delta}(a, S) + 1 + \hat{\delta}(b, y) = 2 + \hat{\delta}(b, y).$$

[5] In addition, S can be computed in polynomial time and with no queries.

Since $(b, y) \in (N(S) \cap U) \times U$, we have $\hat{\delta}(b, y) = \delta(b, y)$. Therefore $\delta(a, y) \geq 2 + \delta(b, y) \geq 2 + \delta(a, x)$. By the triangle inequality, $\delta(x, y) \geq \delta(a, y) - \delta(a, x) \geq 2$. Thus xy is not an edge in $G[U]$. □

From Lemma 4.4.1, the correctness of Verify-Chordal follows by induction. We now analyze the query complexity. Since $\hat{G}[U]$ has maximum degree Δ and S has size at most $\Delta + 1$, Query(S, U) and Query$(N(S) \cap U, U)$ use $O(\Delta^2|U|)$ queries. Let $q(m)$ be the number of queries of Verify-Chordal$(\hat{G}, U)$ when $|U| = m$. We have

$$q(m) = O(\Delta^2 m) + \sum_C q(m + |S|),$$

where $m = |S| + \sum_C |C|$ and S is a (1/2)-balanced separator. Hence $q(n) = O(\Delta^2 n \log n)$.

Remark. *We note that there are simpler algorithms for verifying chordal graphs, but the algorithm presented here conveys ideas that can be extended to verify graphs of bounded treewidth.*

4.4.2 Extension: Graphs of Bounded Treewidth

We extend Algorithm 4.7 to graphs of treewidth w. The input specification is now the graph $\hat{G}$, a subset $U \subseteq V$, *plus* a set F of additional, new edges uv with weight $\delta(u, v)$. The set F is initially empty, and increases during the recursion. The algorithm verifies whether the metric of $(U, \hat{E}[U] \cup F[U])$ is identical to that of $(U, E[U] \cup F[U])$. Instead of S being a clique, now S is an existing bag of some tree decomposition of width w (see Lemma 4.1.3). Verifying the non-edges between different components is the same as before, because Lemma 4.4.1 still holds. To verify the non-edges inside a component C, we create new edges uv with weight $\delta(u, v)$ for all pairs $(u, v) \in S^2$, and add them to the set F of weighted edges. Then we make a recursive call for the vertex set $C \cup S$ and the updated set F. Every subgraph in the recursive call has treewidth at most w, since the new edges are added inside S. This concludes the description and correctness of the algorithm.

For the query complexity, we need to bound the size of the neighborhood $N(S)$ of S: it is with respect to the subgraph $E[U] \cup F[U]$, so the vertex degree is no longer bounded by Δ. However, for any vertex v, the number of weighted edges adjacent to v is bounded by the maximum bag size times the number of bags containing v that have been used as separators in the recursive calls. Since the graph has treewidth w, every bag has size at most $w + 1$. Since all separators are (1/2)-balanced, the recursion has depth $O(\log n)$, so v belongs to $O(\log n)$ such bags. Therefore, the degree of v is $O(\Delta + w \log n)$. The overall query complexity is $O(\Delta(\Delta + w \log n) n \log n)$.

Thus we proved Theorem 1.4.8.

Chapter 5

Side Results

5.1 Lower Bounds

In Section 5.1.1, we give lower bounds for general graphs where the maximum degree is unbounded; and in Section 5.1.2, we give a lower bound for graphs of maximum degree Δ.

5.1.1 General Graphs

Reyzin and Srivastava showed a lower bound as follows.

Lemma 5.1.1. *[73] For graph reconstruction using a distance oracle, any algorithm has query complexity $\Omega(n^2)$.*

This $\Omega(n^2)$ lower bound can be easily extended to the graph verification problem and/or to the shortest path oracle model as follows. Consider a graph G whose vertices are $v_1, \ldots, v_n$ and whose edges form a star: there is an edge $v_1 v_i$ for every $2 \leq i \leq n$. In addition, G may or may not contain a new edge $v_i v_j$, for $2 \leq i, j \leq n$. (In the graph verification problem, the star graph is given as $\hat{G}$.) To check whether G contains a new edge, we need to perform $\Omega(n^2)$ distance or shortest path queries.

5.1.2 Graphs of Bounded Degree

In this section, we prove Theorem 1.4.9 that we recall:

Theorem 1.4.9. *For graph reconstruction using a distance oracle, assuming the maximum degree $\Delta \geq 3$ is such that $\Delta = o\left(n^{1/2}\right)$, any algorithm has query complexity $\Omega(\Delta n \log n / \log(\log n / \log \Delta))$.*

To provide intuition, we first show a lower bound of $\Omega(\Delta n \log n / \log\log n)$, assuming that $n = 3t - 1$, where $t = 2^k$ for some integer k. Consider a family $\mathcal{G}$ of graphs G as follows: the vertex set is $\{v_1, \ldots, v_n\}$; the first $2t-1$ vertices form a complete binary tree of height k (with leaves $v_t, \ldots, v_{2t-1}$); the vertices $v_{2t}, \ldots, v_{3t-1}$ induce an arbitrary subgraph of maximum degree $\Delta - 1$; there is an edge between v_i and v_{i+t} for every $i \in [t, 2t-1]$ and there are no other edges. Then every vertex in G has degree at most Δ, and the diameter of the graph is at most $2k + 2 = O(\log n)$. Every distance query returns a number between 1 and $2k+2$, so it gives $O(\log\log n)$ bits of information. From information theory, the number of queries is at least the logarithm of the number of graphs in $\mathcal{G}$ divided by the maximum number of bits of information per query. The number of graphs in $\mathcal{G}$ is the number of different graphs of t vertices and of maximum degree $\Delta - 1$, which is $\Omega\left(n^{\Omega(\Delta n)}\right)$ when $\Delta = o(\sqrt{n})$ (see [69]). Therefore, we have a query lower bound:

$$\frac{\log\left(\Omega\left(n^{\Omega(\Delta n)}\right)\right)}{O(\log\log n)} = \Omega\left(\frac{\Delta n \log n}{\log\log n}\right).$$

To prove the bound as stated in Theorem 1.4.9, we only need to replace the above complete binary tree by a complete $(\Delta - 1)$-ary tree. The diameter of the graph is now $O(\log n / \log \Delta)$. The theorem statement follows trivially.

5.2 Approximate Reconstruction

In this section, we study the approximate version of the reconstruction problem using a distance oracle on general graphs (not necessarily of bounded degree). We first give a simple algorithm (Algorithm 5.1), and then show that this algorithm is optimal by providing a query lower bound of the same complexity.

Let $G = (V, E)$ be a connected, undirected, and unweighted graph. Let δ be the distance metric of G. Let f be any sublinear function of n. An *f-approximation* $\tilde{\delta}$ of the metric δ is such that, for every $(u, v) \in V^2$, $\tilde{\delta}(u, v) \le \delta(u, v) \le f \cdot \tilde{\delta}(u, v)$.

Algorithm 5.1 receives the vertex set V and outputs the approximate metric $\tilde{\delta}$. The algorithm repeatedly picks a node u such that the distances between u and other nodes are not yet estimated. It then makes queries between u and all other nodes and obtains an estimate $\tilde{\delta}(x, y)$ for every node x within distance $f/2$ from u and for every node y in the graph. The algorithm repeats the above until all distances are estimated.

Theorem 5.2.1. *The algorithm* APPROX-RECONSTRUCT(V) *computes an f-approximation of the graph metric δ using $O(n^2/f)$ distance queries.*

Algorithm 5.1

```
1: procedure APPROX-RECONSTRUCT(V)
2:     S ← V
3:     while S ≠ ∅ do
4:         u ← arbitrary node from S
5:         QUERY(u, S)
6:         Let B be the set of nodes whose distance to u is less than f/2
7:         Set δ̃(x, y) = 1 for all pairs (x, y) ∈ B × B with x ≠ y
8:         Set δ̃(x, y) = δ(u, y) − δ(u, x) for all pairs (x, y) ∈ B × (S \ B)
9:         Remove B from S
10:    return δ̃
```

Proof. First we prove that in the end of the algorithm, for every $(x, y) \in V^2$, we have $\tilde{\delta}(x, y) \le \delta(x, y) \le f \cdot \tilde{\delta}(x, y)$. For every **while** loop, consider any pair $(x, y) \in B \times B$ with $x \ne y$. We have

$$\tilde{\delta}(x, y) = 1 \le \delta(x, y) \le \delta(x, u) + \delta(u, y) < (f/2) + (f/2) = f = f \cdot \tilde{\delta}(x, y).$$

Next, consider any pair $(x, y) \in B \times (S \setminus B)$. On the one hand, by the triangular inequality,

$$\delta(x, y) \ge \delta(u, y) - \delta(u, x) = \tilde{\delta}(x, y).$$

On the other hand, by the triangular inequality,

$$\delta(x, y) \le (\delta(u, y) - \delta(u, x)) + 2\delta(u, x).$$

The first term is $\tilde{\delta}(x, y)$. The second term, by the definition of B, is at most $(f - 1)$. Since $x \in B$ and $y \notin B$, we have $\tilde{\delta}(x, y) \ge 1$, so the second term can be bounded by $f - 1 \le (f - 1) \cdot \tilde{\delta}(x, y)$. Adding completes the proof of the upper bound.

Next, we analyze the query complexity of the algorithm. Let $U \subseteq V$ be the set of nodes u's chosen in Line 4 during the algorithm. For every pair of distinct nodes $u, u' \in U$, we have $\delta(u, u') \ge f/2$. For every $u \in U$, define $N(u, f/4)$ as the neighborhood of u within distance $f/4$. Then we have $|N(u, f/4)| \ge f/4$, since G is connected. Observe that the sets $\{N(u, f/4)\}_u$ are disjoint. So there are at most $4n/f$ sets, i.e., $|U| \le 4n/f$. For every $u \in U$, the algorithm makes $O(n)$ queries. So the total number of queries is $O(n^2/f)$. □

For the lower bound, extending Lemma 5.1.1 gives the following theorem.

Theorem 5.2.2. *To compute an f-approximation of the graph metric δ using a distance oracle, any algorithm requires $\Omega(n^2/f)$ queries.*

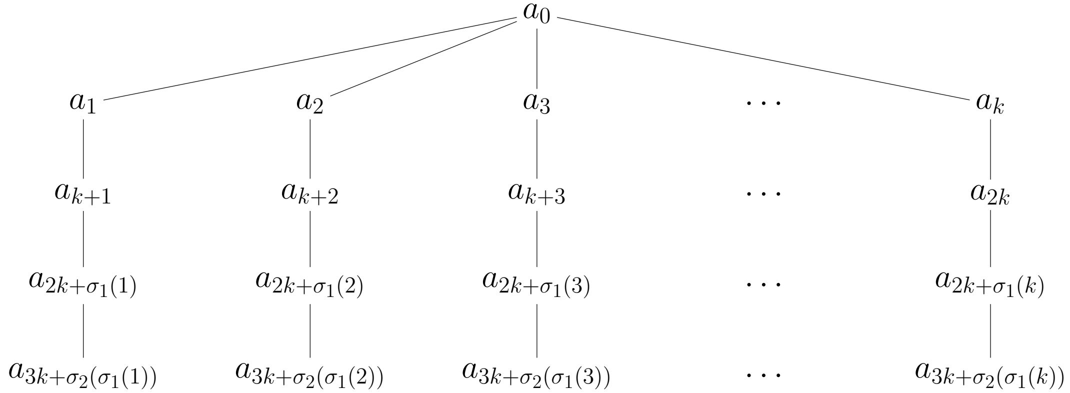


Figure 5.1: In this example, $f = 2$; The construction of the tree is based on two permutations σ_1 and σ_2.

Proof. To simplify the proof, we assume $n = 2fk + 1$, for $k \in \mathbb{N}$. We define a family of instances as follows. For each f-tuple $(\sigma_1, \ldots, \sigma_f)$, where every σ_i is a permutation of $\{1, \ldots, k\}$, we define an instance, which is a tree: it has one vertex a_0 as the root (on the first level), k vertices $a_1, \ldots, a_k$ on the second level, k vertices $a_{k+1}, \ldots, a_{2k}$ on the third level, $\cdots$, and k vertices $a_{n-k}, \ldots, a_{n-1}$ on the $(2f+1)^{\text{th}}$ level. For every $l \in [2, f]$ and every $i \in [1, k]$, there is an edge between the ith node on level l and the ith node on level $l+1$. For every $l \in [f+1, 2f]$ and every $i \in [1, k]$, there is an edge between the ith node on level l and the $\sigma_{l-f}(i)^{\text{th}}$ node on level $l+1$. See Fig. 5.1. We observe that every tree constructed above has k branches from the root, and every branch is a path of $2f+1$ nodes. We will show that any algorithm requires $\Omega(n^2/f)$ queries to compute an f-approximation of the metric on these instances.

First, notice that for these instances, any f-approximation $\tilde{\delta}$ of the metric can be transformed into the metric δ without queries: For every nodes u and v on consecutive levels between $f+1$ and $2f+1$, uv is an edge of G if and only if $\tilde{\delta}(u, v) < 2f$. This is because, if u and v are in the same branch, we have $\delta(u, v) = 1$, thus $\tilde{\delta}(u, v) \leq f$; and if u and v are in different branches, we have $\delta(u, v) = \delta(a_0, u) + \delta(a_0, v) \geq 2f$, thus $\tilde{\delta}(u, v) \geq 2f$. Therefore, we only need to prove that any algorithm for the exact reconstruction problem requires $\Omega(n^2/f)$ queries on these instances.

Let $\mathcal{A}$ be any algorithm that reconstructs these instances exactly. We assume that $\mathcal{A}$ does not make *redundant queries*, i.e., queries whose answers can be deduced in advance. Obviously, any query with the root is redundant. For any two node u and v, let l_u and l_v be their levels. The query (u, v) is redundant when $l_u \leq f+1$ and $l_v \leq f+1$, since the first $f+1$ levels are fixed. Thus every query (u, v) is such that $l_u > f+1$ and $l_v \geq 2$ (we assume $l_u \geq l_v$ without loss of generality). The

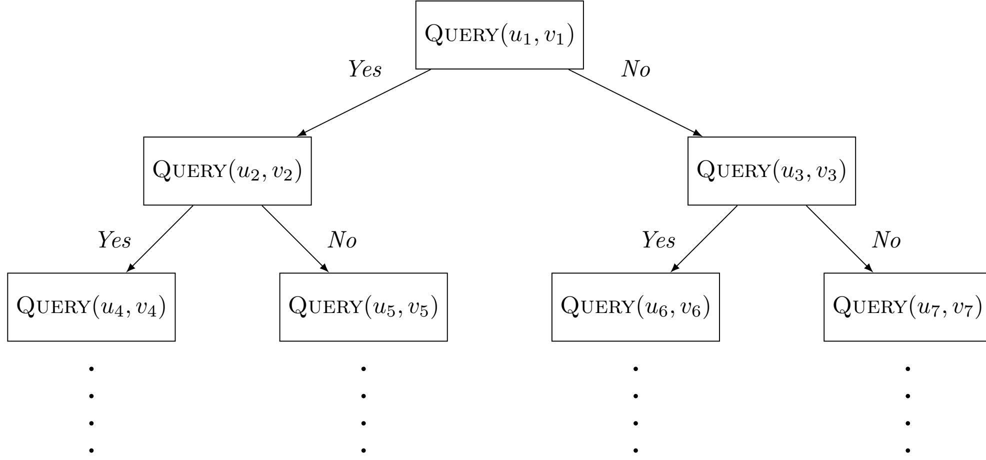


Figure 5.2: Decision tree of $\mathcal{A}$

answer is either $l_u - l_v$, if u and v are in the same branch; or $l_u + l_v - 2$, if u and v are in different branches. We can equivalently identify the answer as *Yes* or *No* to the question: *Are u and v in the same branch?* The key is to bound the number of *Yes* answers during the algorithm. We introduce the *component graph* H, which represents the information from all *Yes* answers received by $\mathcal{A}$. The vertex set of H is defined to be the set of all nodes of level between $f+1$ and $2f+1$. At the beginning, the edge set of H is empty. Each time when $\mathcal{A}$ receives a *Yes* answer to a query (u, v), we add an edge to H according to one of the two cases:

1. $l_u > f+1$ and $l_v \geq f+1$. Then we add the edge uv to H.
2. $l_u > f+1$ and $2 \leq l_v < f+1$. Then we add the edge uw to H, where w is the node on level $f+1$ that is in the same branch as v.

There could not be cycles in H, since otherwise there are redundant queries. The number of connected components in H is at least k, since every connected component in H contains nodes from the same branch of the tree and there are k branches. The number of edges in H is the number of vertices minus the number of connected components, which is at most $k(f+1) - k = kf$. Since every *Yes* answer adds an edge into H, we have $\mathcal{A}$ stops after at most kf *Yes* answers.

Next, we show the lower bound by a decision tree argument. See Figure 5.2. First, $\mathcal{A}$ queries some pair (u_1, v_1). If the answer is *Yes*, it queries some pair (u_2, v_2), otherwise it queries some pair (u_3, v_3), etc. $\mathcal{A}$ stops if and only if it arrives at a leaf of the decision tree. Let h be the depth of the decision tree. We only need to prove that $h = \Omega(n^2/f)$. A leaf of the decision tree is identified by its root-leaf

path, a word over $\{Yes, No\}$ of length at most h and with at most kf *Yes*'s.[1] The total number of leaves in the decision tree is at most

$$\sum_{0 \le j \le kf} \binom{h}{j} \le 2 \cdot \binom{h}{kf} \le \frac{2h^{kf}}{(kf)!}.$$

On the other hand, the number of leaves in the decision tree is the number of instances, which is $(k!)^f$. Therefore,

$$(k!)^f \le \frac{2h^{kf}}{(kf)!}.$$

Using Stirling's formula, we have $h = \Omega(k^2 f) = \Omega(n^2/f)$. □

[1]We assume that the length of this path is exactly h by appending unnecessary *No*'s.

Chapter 6

Conclusion

Main General Results. We have designed an algorithm for graph reconstruction using $\tilde{O}(n^{3/2})$ distance queries and an algorithm for graph verification using $O\left(n^{1+o(1)}\right)$ distance or shortest path queries. Both algorithms decompose the graph into subgraphs using the Voronoi cell decomposition, and then solve the problem in the subgraphs independently. We have also given a greedy algorithm for graph verification using either oracle and we have proved that its query complexity is again $O\left(n^{1+o(1)}\right)$. The greedy algorithm can be extended to graph reconstruction using a shortest path oracle with the same query complexity $O\left(n^{1+o(1)}\right)$.

Main Open Problem. For graph verification using either oracle and graph reconstruction using a shortest path oracle, we have provided algorithms with near-linear query complexity. However, we do not know whether graph reconstruction using a distance oracle is more difficult than these problems. Hence the central open problem: **Is there a reconstruction algorithm using a near-linear number of queries to a distance oracle?**

One Failed Attempt. To design a better-than-$\tilde{O}(n^{3/2})$ algorithm for graph reconstruction via a distance oracle, one might try to extend the Voronoi cell decomposition recursively, as in the algorithm for verification (Algorithm 2.5) with query complexity $O\left(n^{1+o(1)}\right)$. Recall that in Algorithm 2.5, the key subroutine is Subset-Centers (Algorithm 2.4), which roughly equipartitions a subset U into Voronoi cells. This subroutine requires no query in the verification problem, since $\hat{G}$ is given. However, in the reconstruction problem, it would require $\Omega(n)$ queries even if the subset U is small. Therefore, we cannot obtain an efficient recursive algorithm using this framework.

Another Failed Attempt. One might try a greedy approach, which has already been used for graph reconstruction via a shortest path oracle (Algorithm 3.2). Recall that Algorithm 3.2 first finds a connected subgraph spanning all vertices, and then greedily queries a pair (u, v). If the distances between u and v in the subgraph and in the graph G are the same, then it eliminates a large number of non-edges; otherwise it discovers an edge of G and adds it to the current subgraph. In the distance oracle model, finding a connected subgraph spanning all vertices can be done using $\tilde{O}(n)$ queries [63]. However, given a pair of (u, v) such that their distances in the current subgraph and in the graph G are different, $\Omega(n/\log n)$ distance queries are required in general to discover an edge [16]. Therefore, the greedy framework does not lead to an efficient algorithm for reconstruction via a distance oracle.

Future Directions. Let us consider two potential approaches for graph reconstruction. The first approach decomposes the graph into Voronoi cells and then applies recursion. This approach fails when there are many connections between different cells (since in this case, the distance between a pair of nodes in the cell is different from their distance in the entire graph). The second approach is *random elimination*: we select a set S of $\operatorname{polylog} n$ nodes at random, and query the distance between every selected node and every node in the graph. If a pair $uv \in V^2$ is such that $|\delta(u, s) - \delta(v, s)| > 1$ for some selected node $s \in S$, then uv is confirmed to be a non-edge of G. Next, we query all the pairs that are not yet confirmed. We have tested this algorithm on random Δ-regular graphs, where it only requires a near-linear number of queries. However, this approach fails when there are few connections between different parts of the graph. For example, if the graph is a complete binary tree, then the number of queries in the last step is quadratic.

Since the two approaches fail on opposite instances of the graphs, it might be possible to design an algorithm that combines the two approaches and has a near-linear query complexity.

Special Cases of Graphs. Although for general graphs, there is no reconstruction algorithm using a near-linear number of distance queries, when the graph is chordal or outerplanar, we have provided algorithms using $\tilde{O}(n)$ distance queries. These algorithms exploit the property that such graphs admit a small separator such that there is no connection between different sides of the separator. Note that when graphs have bounded degree, both chordal graphs and outerplanar graphs have bounded treewidth. For reconstruction using a shortest path oracle and verification using either oracle, we have further improved the query complexity from $O\left(n^{1+o(1)}\right)$ to $\tilde{O}(n)$ for graphs of bounded treewidth.

Intermediate Open Problem. We would like to know whether there is a reconstruction algorithm using a near-linear number of queries to a distance oracle, when the graph has bounded treewidth.

Other Open Problems. As noted in [14], we could consider other objectives of network inference, such as asking for the minimum number of queries to discover a fixed percentage of edges and non-edges, or determining the diameter of the network.

Part II

Planar Graph Optimization

Chapter 7

Introduction

We consider two problems in planar graphs: *correlation clustering* and *two-edge-connected augmentation*. We address them in the same work because they can be related via planar duality, which will be discussed later.

7.1 Correlation Clustering

7.1.1 The Problem

The *correlation clustering* problem takes as input a graph whose edges are labelled either $\langle + \rangle$ or $\langle - \rangle$. A $\langle + \rangle$ edge represents evidence that its endpoints belong to the same cluster, and a $\langle - \rangle$ edge represents evidence that its endpoints belong to different clusters. Each edge has a non-negative *weight* reflecting the strength of the evidence. The goal is to find a partition of the vertices into clusters that minimizes the total weight of the edges inconsistent with that evidence. See Fig. 7.1.

This problem was first considered by Ben-Dor, Shamir, and Yakhini [15], motivated by some computational biology questions. Bansal, Blum, and Chawla [11] also independently formulated and considered this problem, motivated by machine learning problems concerning document classification.

> [35] For example, the multiset of objects might consist of all authors of English literature, and two authors belong to the same category if they correspond to the same real person. This task would be easy if authors published papers consistently under the same name. However, some authors might publish under several different names such as William Shakespeare, W. Shakespeare, Bill Shakespeare, Sir Francis Bacon, Edward de Vere, and Queen Elizabeth I. Given

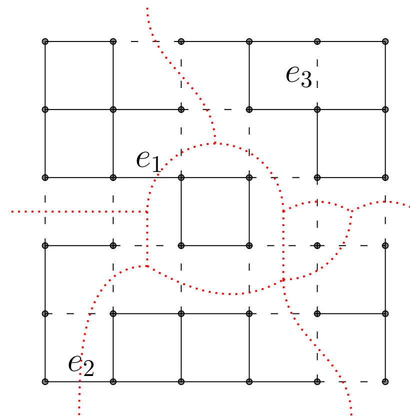


Figure 7.1: In this example, the instance is an unweighted grid graph. The $\langle + \rangle$ edges are solid, and the $\langle - \rangle$ edges dashed. Dotted lines indicate an optimal partition of vertices with inconsistent edges e_1, e_2, e_3.

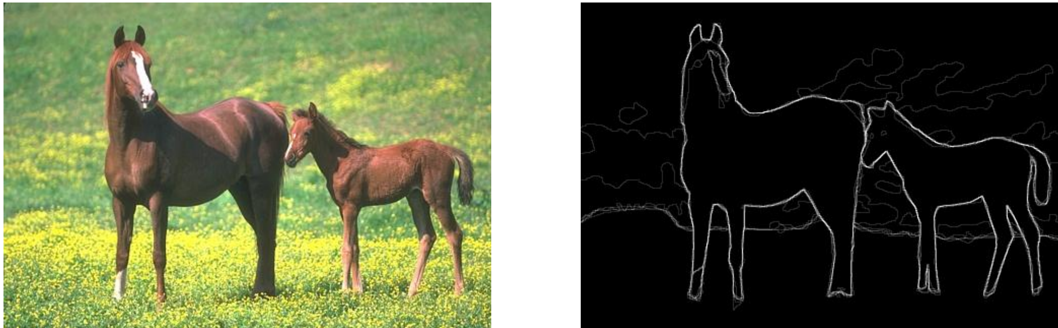


Figure 7.2: Image Segmentation

some confidence about the similarity and dissimilarity of the names, our goal is to cluster the objects to maximize the probability of correctness.

This problem on general graphs is APX-hard [11, 26, 35]. In this work, we study the case when the graph is *planar*. The motivation for planar graphs comes from *image segmentation*. The goal is to partition the image into regions representing different image components. See Fig. 7.2. An image is represented by a grid of *pixels*. For each pair of neighboring pixels, comparing the pixels' values yields an assessment of how likely the pixels are to belong to the same region. There can be spurious assessments. So global optimization is needed to find a good segmentation. When an image is large, it is common for a visual task to first coalesce coherent uniform neighborhoods of pixels into *superpixels*, using preprocessing based on local properties such as brightness, color, and texture, see [3, 66]. We then extract a local similarity measure on pairs of adjacent superpixels, and the goal is to find a

good segmentation of the superpixel graph under that measure. Researchers have formulated this problem as correlation clustering, e.g., [4, 5, 6, 58, 80]. They gave experimental results based on techniques such as integer linear programming or linear programming relaxation.

Note that the superpixel graph is planar. For correlation clustering on planar graphs, Bachrach et al. [10] showed NP-hardness, and prior to this work, the best algorithm with theoretical guarantee was a constant-factor approximation for minor-excluded graphs by Demaine et al. [35].

7.1.2 Related Work

Correlation clustering and its variants have been extensively studied because of their numerous applications, for example in computational biology [15, 19], data mining [31], machine learning [11], and computer vision [4, 5, 6, 58, 80].

General Graphs. Correlation clustering on general (weighted) graphs is APX-hard [11, 26, 35]. Charikar, Guruswami, and Wirth [26], and independently, Demaine et al. [35] gave an $O(\log n)$-approximation algorithm based on linear programming rounding and the *region-growing* technique. In [26, 35], the authors also noted that any $o(\log n)$-approximation algorithm for correlation clustering would require improving the state-of-art for approximating *minimum multicut*.

For the variant of the problem where the goal is to maximize the total weight of agreements, Charikar, Guruswami, and Wirth [26] showed APX-hardness and provided a 0.7664-approximation algorithm. Independently, Swamy [75] gave a 0.7666-approximation algorithm.

Complete Graphs. Bansal, Blum, and Chawla [11] studied the problem in an unweighted complete graph, i.e., every pair of vertices has a label of either *similar* or *dissimilar*. They gave a constant-factor approximation algorithm (for some large constant). Charikar, Guruswami, and Wirth [26] gave a 4-approximation algorithm and showed APX-hardness. For maximizing agreements, a PTAS was given in [11].

In the weighted setting, for each pair of vertices, the agreement weight and disagreement weight sum to one. Again, Bansal, Blum, and Chawla [11] gave a constant-factor approximation algorithm (for some large constant). Ailon, Charikar, and Newman [2] gave an algorithm that achieves expected approximation ratio 5. If in addition the weights obey the triangular inequality, the algorithm in [2] achieves expected approximation ratio 2, and an algorithm of worst case approximation ratio 3 was obtained by Gionis, Mannila, and Tsaparas [44].

Fixed Number of Clusters. Giotis and Guruswami [45] studied the variant when the number of clusters is limited to a constant, which might be an external constraint. They showed that both minimizing disagreement and maximizing agreement admit a PTAS and are NP-hard.

Noisy Input. Mathieu and Schudy [67] considered the variant where the input graph is generated from an arbitrary partition of the vertices into clusters, such that for each vertex pair, the similarity information is corrupted independently with some probability p. They showed that the clusters can be reconstructed exactly when all clusters are large or when p is small.

7.2 Two-Edge-Connected Augmentation

7.2.1 The Problem

In the field of telecommunications, an important task is to ensure that the network is resilient against link failures. Since failures are rare in real-life networks, it is sufficient for the network to be resilient against single-link failures. There has been much research on two-edge-connectivity problems, which require that there are two edge-disjoint paths between nodes in the network. See [72] for a survey.

The *two-edge-connected augmentation* problem takes as input a graph G with non-negative edge-weights and a subset R of edges of the graph. The goal is to find a minimum-weight subset S of edges of the graph such that for every edge $uv \in R$, u and v are two-edge-connected in the subgraph $R \cup S$. Without loss of generality, we assume that edges from R have weight 0, since any minimal two-edge-connected augmentation for (G, R) does not contain edges from R.

This problem is a generalization of the well-studied *tree augmentation* problem: given a graph G with non-negative edge-weights and given a spanning tree T of G, find a minimum-weight subset S of edges such that the subgraph $T \cup S$ is two-edge-connected. The condition is equivalent to requiring that for each edge uv of T, u and v are two-edge-connected in $T \cup S$. Kortsarz, Krauthgamer, and Lee [64] showed that tree augmentation is APX-hard. Thus two-edge-connected augmentation is also APX-hard. Frederickson and Ja'Ja' [41] gave a polynomial-time 2-approximation algorithm for *tree augmentation*. The running time of that algorithm was improved by Khuller and Thurimella [56], and further by Galluccio and Proietti [42].

In this work, we study two-edge-connected augmentation in *planar graphs*.

7.2.2 Related Work

Two-Edge-Connected Spanning Subgraph. In this problem, we want to find a minimum-weight subgraph of G in which every pair of vertices of G is two-edge-connected. This problem in general graphs was shown to be Max-SNP-hard by Czumaj and Lingas [33]. Frederickson and Ja'Ja' [41] gave a 3-approximation algorithm. The approximation ratio was improved to 2 (and 3/2 for unweighted graphs) by Khuller and Vishkin [57], and to 5/4 by Jothi, Raghavachari, and Varadarajan [53].

When the graph is planar, Eswaran and Tarjan [39] showed NP-hardness (by a reduction from Hamiltonian cycle), and Berger and Grigni [17] gave a PTAS. One might think that this would lead to a PTAS for two-edge-connected augmentation, but it is not the case, because the weight of a two-edge-connected augmentation can be much smaller than the weight of a two-edge-connected spanning subgraph.

Two-Edge-Connected Steiner Subgraph. In this problem, we are given a subset $Q \subseteq V$ of terminals, and we want to find a minimum-weight subgraph of G in which every pair of vertices of Q is two-edge-connected. This is a generalization of two-edge-connected spanning subgraph. Klein and Ravi [62] gave a 3-approximation algorithm. (In fact, they solved a more general problem where the connectivity requirements are specified for *pairs* of vertices.) This result was generalized to higher connectivity requirements by Williamson et al. [79] and Goemans et al. [46]. Edge-connectivity problems were subsumed by the work of Jain [51] on survivable network design.

When the graph is planar, Borradaile and Klein [21] gave a PTAS for a variant of the two-edge-connected Steiner subgraph problem, where a solution is allowed to include multiple copies of edges of the input graph. One might think that this would lead to a PTAS for two-edge-connected augmentation, but it is not the case, mainly because the *structure property* in [21] does not hold for two-edge-connected augmentation. This issue will be discussed later.

Other Work. There is a variety of other related work, see [65] for a survey.

7.3 Our Results

The results here have been published in [B].

First, we show that in planar graphs, correlation clustering can be reduced to two-edge-connected augmentation:

Theorem 7.3.1. *There is a polynomial-time approximation-preserving reduction from correlation clustering in weighted planar graphs to two-edge-connected augmentation in weighted planar graphs.*

Next, we give a *polynomial-time approximation scheme (PTAS)* for two-edge-connected augmentation when the graph is planar:

Theorem 7.3.2. *For any $\epsilon > 0$, there is a polynomial-time $(1+\epsilon)$-approximation algorithm for two-edge-connected augmentation in weighted planar graphs.*

From Theorems 7.3.1 and 7.3.2, we obtain a PTAS for correlation clustering:

Theorem 7.3.3. *For any $\epsilon > 0$, there is a polynomial-time $(1+\epsilon)$-approximation algorithm for correlation clustering in weighted planar graphs.*

Remark. *From the NP-hardness of correlation clustering in planar graphs [10] and the reduction (Theorem 7.3.1), we know that two-edge-connected augmentation in planar graphs is also NP-hard.*

Remark. *In practice, we may use an approximation algorithm for two-edge-connected augmentation that is different from the algorithm in Theorem 7.3.2, and then from the reduction (Theorem 7.3.1), we obtain an algorithm for planar correlation clustering with the same approximation factor.*

7.4 Notations and Definitions

Let G be a graph with non-negative edge-weights. Let $V[G]$ (or simply V) be its vertex set, and let $E[G]$ (or simply E) be its edge set. We allow G to have parallel edges. For a subset of edges $H \subseteq E[G]$, we identify H with the subgraph induced by edges from H. The weight of H is defined by $\sum_{e \in H} \text{weight}(e)$. For a subset of vertices $U \subseteq V[G]$, we define its *boundary* $\partial(U)$ as the set of edges $uv \in E[G]$ such that $u \in U$ and $v \in V[G] \setminus U$.

A *plane graph* is a planar graph together with a planar embedding. We use the phrases *plane graph* and *planar graph* interchangeably.

Next, we give the definitions of two optimization problems.

In the *Steiner tree* problem, we are given a weighted planar graph $G = (V, E)$ and a set $Q \subseteq V$ of terminals, and the goal is to find a minimum-weight connected subgraph connecting every terminal in Q.

In the *Steiner forest* problem, we are given a weighted planar graph $G = (V, E)$ and a set $\mathcal{D}$ of demands $(s,t) \in V^2$, and the goal is to find a minimum-weight forest F of G such that, for every demand $(s,t) \in \mathcal{D}$, s and t are connected in F.

For a given optimization problem in weighted graphs, we use $OPT(\mathcal{I})$ to denote the weight of an optimal solution for this problem on the instance $\mathcal{I}$. The parameter $\mathcal{I}$ is omitted when it is clear from the context.

7.5 Organization

In Chapter 8, we show the reduction from correlation clustering to two-edge-connected augmentation in planar graphs (Theorem 7.3.1). The proof is elementary and is based mainly on planar duality.

In Chapter 9, we review various techniques for designing approximation schemes in planar graphs. The techniques of *prize-collecting partition* [13] and *brick decomposition* [22] have been used to design approximation schemes for Steiner forest and Steiner tree in their original settings. Sections 9.1 and 9.2 review these two techniques respectively, and give slight adaptations for the two-edge-connected augmentation problem. Section 9.3 is a short survey on previous approximation schemes using brick decomposition. Section 9.4 presents a technical detail in the algorithmic design, which is unique for two-edge-connected augmentation. Section 9.5 reviews the *sphere-cut decomposition* technique from [37], which is useful for the dynamic program in Section 10.4.

Chapter 10 contains the main contribution of Part II of this thesis. In this chapter, we design an approximation scheme for two-edge-connected augmentation in planar graphs (Theorem 7.3.2). After a preprocessing step in Section 10.1, we give high-level ideas of the approximation scheme in Section 10.2. The difficulty in using the *brick decomposition* is that, the structure property on bricks that was used to design previous approximation schemes does not hold for two-edge-connected augmentation. Hence the novelty of our work: we give a new structure property on bricks (see Section 10.2). The proof of the new structure property is in Section 10.3. Using this property, Section 10.4 gives a dynamic program to compute a near-optimal solution. We complete the analysis of the approximation scheme in Section 10.5.

Finally, we discuss some open problems in Chapter 11.

Chapter 8

Reduction from Clustering to Augmentation

In this chapter, we prove Theorem 7.3.1 that we recall:

Theorem 7.3.1. *There is a polynomial-time approximation-preserving reduction from correlation clustering in weighted planar graphs to two-edge-connected augmentation in weighted planar graphs.*

Let G_0 be a plane graph with edge-labels in the correlation clustering problem. Let OPT_0 be the minimum weight of disagreements in correlation clustering. Next, we construct the graph G_1 as the abstract dual[1] of G_0, and let $R \subseteq E[G_1]$ be the duals of the $\langle - \rangle$ edges in G_0. Consider the problem of finding a minimum-weight subset $S_1 \subseteq E[G_1]$ such that $R \oplus S_1$ is a collection of two-edge-connected components in G_1. This problem is called the *intermediate problem.* Let OPT_1 be the minimum weight of a solution $S_1 \subseteq E[G_1]$ for the intermediate problem. Finally, we construct the graph G_2 from G_1 by adding a copy e' of e with the same weight, for every edge $e \in R$; the set R remains the same. Let OPT_2 be the minimum weight of a two-edge-connected augmentation $S_2 \subseteq E[G_2]$ for (G_2, R).

Theorem 7.3.1 follows directly from Lemmas 8.1.1 and 8.2.1, which correspond to two stages of the reduction.

8.1 First Stage

Lemma 8.1.1. $OPT_0 = OPT_1$. *Any solution for the intermediate problem can be transformed in polynomial time into a solution for correlation clustering in G_0 with the same weight.*

[1]See for example [36] for the definition of *abstract dual* and its properties.

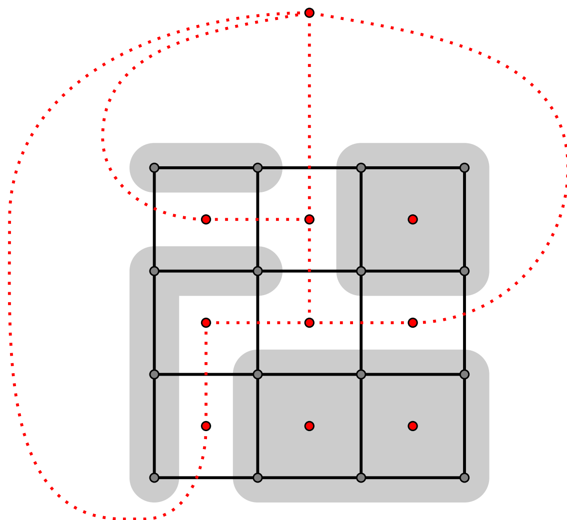


Figure 8.1: In this example, G_0 is represented by the grid graph. In its dual, the subgraph $R \oplus S_1$ is represented by the dotted edges. These edges separate the plane into four faces, which define four clusters of the vertices of G_0 (indicated by the gray areas).

Proof. It is sufficient to show that, a set of edges $S_0 \subseteq E[G_0]$ is the disagreements of some clustering in G_0 if and only if $R \oplus S_1$ is a collection of two-edge-connected components in G_1, where $S_1 \subseteq E[G_1]$ is the dual of S_0.

($\Longrightarrow$) Let $\{V_i\}_i$ be a clustering of G_0 with disagreements S_0. We observe that an edge belongs to some $\partial(V_i)$ if and only if it is a $\langle - \rangle$ edge and an agreement or is a $\langle + \rangle$ edge and a disagreement. Since R is the duals of the $\langle - \rangle$ edges in G_0 and S_1 is the dual of S_0, we have $R \oplus S_1$ is the dual of $\bigcup_i \partial(V_i)$. From the planar duality, the dual of every $\partial(V_i)$ is a union of cycles in G_1. Therefore, the dual of $\bigcup_i \partial(V_i)$, i.e. $R \oplus S_1$, is a collection of two-edge-connected components in G_1.

($\Longleftarrow$) Suppose $R \oplus S_1$ is a collection of two-edge-connected components in G_1. For every face F of that subgraph, define V_F as the set of vertices of G_0 inside this face. From the planar duality, $R \oplus S_1$ is the dual of $\bigcup_F \partial(V_F)$. See Fig. 8.1. Since S_1 is the dual of S_0, using the same argument as before, we deduce that $\{V_F\}_F$ is a clustering of G_0 with disagreements S_0. □

8.2 Second Stage

Lemma 8.2.1. $OPT_1 \geq OPT_2$*. Any two-edge-connected augmentation for* (G_2, R) *can be transformed in polynomial time into a solution for the intermediate problem with at most the same weight.*

The following lemma is the key to prove Lemma 8.2.1. It is also used to specify the dynamic programming table in Section 10.4.1.

Lemma 8.2.2. *Let G be a plane graph. Let R be a subset of $E[G]$. Let S be a minimal two-edge-connected augmentation for (G, R). Then every connected component in the subgraph $R \cup S$ is two-edge-connected.*

Proof. Suppose there is a bridge edge e in the subgraph $R \cup S$. Since S is a two-edge-connected augmentation for (G, R), e cannot belong to R, thus $e \in S$. Therefore, $S \setminus \{e\}$ remains a two-edge-connected augmentation, which contradicts the minimality of S. □

Proof of Lemma 8.2.1. Let $S_1 \subseteq E[G_1]$ be an optimal solution for the intermediate problem. We construct a subset $S_2 \subseteq E[G_2]$ as follows: for every $e \in R$, include its copy e' in S_2 if and only if $e \in S_1$; and for every $e \in E[G_1] \setminus R$, include e in S_2 if and only if $e \in S_1$. It is straightforward that S_2 is a two-edge-connected augmentation for (G_2, R) and has the same weight as S_1. Therefore $OPT_1 \geq OPT_2$.

Next, we show the second part of the statement. Let S_2 be any two-edge-connected augmentation for (G_2, R). By removing unnecessary edges, we may assume that S_2 is minimal. Thus S_2 does not contain edges from R. We construct a subset $S_1 \subseteq E[G_1]$ as follows: for every $e \in R$, include e in S_1 if and only if its copy e' is in S_2; and for every $e \in E[G_1] \setminus R$, include e in S_1 if and only if $e \in S_2$. By the construction, the weight of S_1 is at most the weight of S_2. We only need to show that $R \oplus S_1$ is a collection of two-edge-connected components in G_1.

By Lemma 8.2.2, $R \cup S_2$ is a collection of two-edge-connected components. The differences between $R \cup S_2$ and $R \oplus S_1$ are the pairs of edges $\{e, e'\}$ such that $e \in R$ and $e \in S_1$ (i.e., its copy e' belongs to S_2). Consider any such edge e. Since S_2 is minimal and contains e', the endpoints of e belong to different components in the subgraph $R \cup S_2 \setminus \{e, e'\}$. Therefore, removing $\{e, e'\}$ from $R \cup S_2$ remains a collection of two-edge-connected components. Since $R \oplus S_1$ can be obtained from $R \cup S_2$ by repeatedly removing such pairs $\{e, e'\}$, we have $R \oplus S_1$ is a collection of two-edge-connected components. □

Chapter 9

Techniques

In this chapter, we review various techniques that will be used for designing the approximation scheme in Chapter 10.

9.1 Prize-Collecting Partition

The *prize-collecting partition (PC partition)* technique helps to obtain approximation schemes in planar graphs, such as for Steiner forest [13] and multiway cut [12]. It is used as a preprocessing step to break down the input instance into separate subinstances which are easier to handle. In Section 9.1.1, we review this technique in its original settings, and in Section 9.1.2, we provide a slightly adapted version for two-edge-connected augmentation.

9.1.1 Steiner Forest

Bateni, Hajiaghayi, and Marx [13] used an algorithm, called *prize-collecting partition (PC partition)*, to partition the instance into subinstances such that the solution of each subinstance is connected. The central subroutine of this algorithm is *prize-collecting clustering (PC clustering)*, with the following properties.

Theorem 9.1.1 (Theorem 3.1 from [13])**.** *Let G be a graph with edge-weights where every vertex v has a potential ϕ_v. The PC clustering algorithm (Algorithm 2 in [13]) computes in polynomial time a forest F such that:*

- *The weight of F is at most $2\sum_{v\in V}\phi_v$;*

- *For any subgraph L of G, there is a set Q of vertices such that:*

Algorithm 9.1 PC-Partition$(G, \mathcal{D}, \epsilon)$, see Theorem 9.1.2

1: Use the algorithm of [47] to find a Steiner forest Y of $(G, \mathcal{D})$ with weight at most $2 \cdot OPT$
2: Contract each connected component of Y to build a graph $\tilde{G}$
3: **for** $v \in \tilde{G}$ **do**
4: $\quad \phi_v \leftarrow 1/\epsilon$ times the weight of the component corresponding to v
5: Apply PC clustering on $\tilde{G}$ and $\{\phi_v\}_v$, thus obtaining a forest F
6: **return** connected components of the subgraph $Y \cup F$ of G

- $\sum_{v \in Q} \phi_v$ *is at most the weight of* L*;*
- *If two vertices* $v_1, v_2 \notin Q$ *are connected by* L*, then they are in the same component of* F*.*

Built on PC clustering, the PC partition algorithm (Algorithm 9.1) proceeds as follows. It starts with a 2-approximate solution Y for Steiner forest, and contracts the edges of Y. Next, it assigns potentials to vertices, and then use PC-clustering to find a forest F. Consider the subgraph of the uncontracted graph consisting of the edges of the forest F together with the edges of the 2-approximate solution Y. The algorithm outputs the connected components in this subgraph. Bateni, Hajiaghayi, and Marx [13] showed the following theorem of PC partition for Steiner forest.

Theorem 9.1.2 (Theorem 3.1 from [13])**.** *Let* G *be a graph with edge-weights. Let* $\mathcal{D}$ *be a set of demand pairs. Let* $\epsilon > 0$ *be a parameter. The algorithm* PC-Partition$(G, \mathcal{D}, \epsilon)$ *(Algorithm 9.1) computes in polynomial time a set of connected subgraphs* $T_1, \ldots, T_k$ *with the following properties:*

- *For every demand* $(s, t) \in \mathcal{D}$*, there is some* T_i *containing both* s *and* t*;*
- $\sum_i \text{weight}(T_i) \leq (4/\epsilon + 2) OPT(G, \mathcal{D})$*;*
- $\sum_i OPT(G, \mathcal{D}_i) \leq (1 + \epsilon) OPT(G, \mathcal{D})$*, where* $\mathcal{D}_i$ *is the set of demands* $(s, t) \in \mathcal{D}$ *such that both* s *and* t *belong to* T_i*.*

9.1.2 Two-Edge-Connected Augmentation

In order to use the PC Partition framework for two-edge-connected augmentation, we first need to find a 2-approximate solution for this problem. This can be done in polynomial time using Jain's algorithm [51] (which solves a much more general problem). Jain [51] showed that there is polynomial-time algorithm that computes a 2-approximate solution for the following problem: Given a graph G

Algorithm 9.2 PC-Partition(G, R, ϵ), see Theorem 9.1.4

1: Use the algorithm of Lemma 9.1.3 to find a two-edge-connected augmentation Y of (G, R) with weight at most $2 \cdot OPT$
2: Contract each connected component of $R \cup Y$ to build a graph $\tilde{G}$
3: **for** $v \in \tilde{G}$ **do**
4: $\phi_v \leftarrow 1/\epsilon$ times the weight of the component corresponding to v
5: Apply PC clustering on $\tilde{G}$ and $\{\phi_v\}_v$, thus obtaining a forest F
6: **return** connected components of the subgraph $R \cup Y \cup F$ of G

with non-negative edge-weights, and requirements $r_{u,v} \in \mathbb{Z}$ for each pair (u, v) of vertices, find a minimum-weight subgraph of G such that, for each pair (u, v), the subgraph has at least $r_{u,v}$ edge-disjoint paths between u and v.

Let (G, R) be an instance of two-edge-connected augmentation. We construct an instance in Jain's problem: the graph G remains the same; for every pair of vertices $(u, v) \in V^2$, $r_{u,v}$ is set to 2 if $uv \in R$ and to 0 otherwise. The two problems are equivalent, since every edge of R has weight 0 in the two-edge-connected augmentation problem. Therefore, we have:

Lemma 9.1.3 (Corollary from [51])**.** *There is an algorithm that computes in polynomial time a two-edge-connected augmentation Y for (G, R) such that* $\text{weight}(Y) \leq 2 \cdot OPT$.

The PC partition algorithm for two-edge-connected augmentation is given in Algorithm 9.2. It is almost identical to that for Steiner forest (Algorithm 9.1), except that the connected components are defined in the subgraph $R \cup Y$, not in the subgraph Y. Using essentially the same proof as for Theorem 9.1.2, we obtain the following theorem of PC partition for two-edge-connected augmentation.

Theorem 9.1.4 (Partition Theorem)**.** *Let G be a graph with edge-weights. Let R be a subset of $E[G]$. Let $\epsilon > 0$ be a parameter. The algorithm* PC-Partition(G, R, ϵ) *(Algorithm 9.2) computes in polynomial time a set of connected subgraphs $T_1, \ldots, T_k$ with the following properties:*

- *For every edge $uv \in R$, there is some T_i containing the edge uv;*
- $\sum_i \text{weight}(T_i) \leq (4/\epsilon + 2) OPT(G, R)$*;*
- $\sum_i OPT(G, R_i) \leq (1 + \epsilon) OPT(G, R)$*, where R_i is the set of edges $uv \in R$ that are in T_i.*

9.2 Brick Decomposition

For non-local problems in weighted planar graphs in which the weight of the optimal solution can be much smaller than the weight of the graph, the *brick decomposition* technique by Borradaile, Klein, and Mathieu [22] has proved to be quite versatile: a planar embedded subgraph M (called the *mortar graph*) is selected, and the *bricks* are the subgraphs of G embedded in the faces of M. This technique has been used for designing approximation schemes for problems such as *Steiner tree* [20, 22], *Steiner forest* [13], *two-edge-connected survivability* [20, 21][1], *TSP* [20], and *multiway cut* [12]. In Section 9.2.1, we review the definition and properties of the brick decomposition in its original settings, and in Section 9.2.2, we provide a slightly adapted version for two-edge-connected augmentation.

9.2.1 Steiner Tree

Borradaile, Klein, and Mathieu [22] first introduced the *brick decomposition* technique to design approximation schemes for Steiner tree in planar graphs. They gave the following definition and properties of the brick decomposition.

Definition 9.2.1 ([22]). *Let G be a plane graph with edge-weights. Let $Q \subseteq V$ be a set of terminals. Let $\epsilon > 0$ be a parameter. Let M be a subgraph of G. For each face F of M, we define a* brick *B as the planar subgraph of G embedded inside the face, including the boundary edges of F. We denote the* interior *of B as the brick without the boundary edges of F. We call M a* mortar graph *of G if, for every brick B, its boundary in counter-clockwise order is the concatenation of four paths* West_B, South_B, East_B, North_B *(the subscript B is omitted when it is clear from the context), such that:*

1. *The interior of B is non-empty;*
2. *Every terminal of Q that is in B is on* North *or on* South*;*
3. North *is a path of minimum weight in B, and every proper subpath of* South *is a path of almost minimum weight in B, i.e., its weight is at most $(1 + \epsilon)$ times the minimum weight of a path in B between its endpoints;*
4. *There exists an integer $k = O(1/\epsilon^3)$ and vertices $s_0, \dots, s_k$ ordered from left to right along* South *to induce a partition of* South*, such that, for any vertex x on the segment* $\text{South}[s_i, s_{i+1})$*, the weight of the segment* $\text{South}[s_i, x]$ *is less than ϵ times the minimum weight of a path in B between x and* North.

[1] For the variant in which the solution is allowed to include multiple copies of edges of the input graph.

Lemma 9.2.2 ([22]). *Let G be a planar graph with edge-weights. Let $Q \subseteq V$ be a set of terminals. Let T be a tree in G that spans every terminal of Q. Let $\epsilon > 0$ be a parameter. There is a polynomial-time algorithm that computes a mortar graph M of G (see Definition 9.2.1), such that:*

1. $\text{weight}(M) = O(\text{weight}(T)/\epsilon)$;
2. $\sum_{brick\ B} \text{weight}(\text{East}_B \cup \text{West}_B) = O(\epsilon \cdot \text{weight}(T))$.

9.2.2 Two-Edge-Connected Augmentation

In the construction of the mortar graph for two-edge-connected augmentation, we take additional care because of the edges of R. Definition 9.2.3 and Lemma 9.2.4 are the counterparts of Definition 9.2.1 and Lemma 9.2.2.

Definition 9.2.3 (Mortar Graph and Bricks). *Let G be a plane graph with edge-weights. Let R be a subset of $E[G]$. Let $\epsilon > 0$ be a parameter. Let M be a subgraph of G. For each face F of M, we define a* brick *B as the planar subgraph of G embedded inside the face, including the boundary edges of F. We denote the* interior *of B as the brick without the boundary edges of F. We call M a* mortar graph *of G if, for every brick B, its boundary in counter-clockwise order is the concatenation of four paths West_B, South_B, East_B, North_B (the subscript B is omitted when it is clear from the context), such that:*

1. *The interior of B is non-empty;*
2. *Every edge of R that is in B is on North;*
3. South *is a path of minimum weight in B, and every proper subpath of* North *is a path of almost minimum weight in B, i.e., its weight is at most $(1+\epsilon)$ times the minimum weight of a path in B between its endpoints;*
4. *There exists an integer $k = O(1/\epsilon^4)$ and vertices $s_0, \ldots, s_k$ ordered from left to right along* South *to induce a partition of* South, *such that, for any vertex x on the segment $\text{South}[s_i, s_{i+1})$, the weight of the segment $\text{South}[s_i, x]$ is less than ϵ times the minimum weight of a path in B between x and* North.

Lemma 9.2.4 (Mortar-Graph Lemma). *Let G be a planar graph with edge-weights. Let R be a subset of $E[G]$. Let T be a connected subgraph of G that contains every edge of R. Let $\epsilon > 0$ be a parameter. There is a polynomial-time algorithm that computes a mortar graph M of G (see Definition 9.2.3), such that:*

1. $\text{weight}(M) = O(\text{weight}(T)/\epsilon)$;

2. $\sum_{brick\ B} \text{weight}(\text{East}_B \cup \text{West}_B) = O(\epsilon^2 \cdot \text{weight}(T))$.

Remark. *There are several differences between the mortar graph for Steiner tree and that for two-edge-connected augmentation:*

- *Property 2 of Definition 9.2.3 requires that edges of R appear only on the* North *boundaries of bricks. This can be achieved by requiring that edges of R appear only on the* North *boundaries of* strips *during the construction. See [22, 60] for the details of the decomposition into* strips.
- *Compared with Definition 9.2.1,* South *and* North *in Property 3 of Definition 9.2.3 are swapped. Indeed, in the construction of bricks in [22], the distinction between* South *and* North *is not important.*
- *The parameter k in Property 4 of Definition 9.2.3 is $O(1/\epsilon^4)$ instead of $O(1/\epsilon^3)$ in Definition 9.2.1. Therefore, the bound in Property 2 of Lemma 9.2.4 is $O(\epsilon^2 \cdot \text{weight}(T))$ instead of $O(\epsilon \cdot \text{weight}(T))$ in Lemma 9.2.2.*
- *The brick decomposition in [22] requires that T is a tree. Later, Klein and Mozes [61] generalized T to be any connected subgraph: Their approach is to do the brick decomposition in each face of T.*

9.3 Framework of Approximation Schemes

The approximation schemes in planar graphs for *Steiner tree* [20, 22], *Steiner forest* [13], *two-edge-connected survivability* [20, 21], *TSP* [20], and *multiway cut* [12] all use the brick decomposition technique, and have a similar framework which we now summarize. First, the algorithm finds an O(1)-approximate solution and builds a mortar graph. Next, it does Breadth-First Search (BFS) on the dual of the mortar graph, and selects a mod-η residue j^* such that edges whose levels are congruent to j^* have total weight at most $1/\eta$ times the weight of the mortar graph. It commits to including these edges in the ultimate solution; this decomposes the graph into subinstances each consisting of at most η levels of bricks. Note that a planar graph consisting of at most η BFS levels has branchwidth at most 2η, i.e., can be recursively decomposed into clusters of edges such that each cluster has at most 2η boundary vertices. For each subinstance, it finds a near-optimal solution by dynamic programming. Finally, it returns the union of the solutions for all subinstances.

Remark. *In the approximation scheme for Steiner forest [13] or for multiway cut [12], there is an additional preprocessing step of PC partition (see Section 9.1), which reduces the instance to subinstances. This step ensures that each subinstance*

Algorithm 9.3 AUGMENT-CONNECTED(G, Q, T, ϵ), see [22]

1: Compute a mortar graph M of G based on Q and T (Lemma 9.2.2).
2: Do BFS in the planar dual M^* starting from an arbitrary vertex r. Define the *level* of a vertex of M^* as its BFS distance from r. Let E_i denote the set of edges of M^* whose two endpoints are at level i and level $i+1$, respectively. Let $\eta = \Theta(1/\epsilon^3)$. For $j = 0, 1, \ldots, \eta - 1$, let $\mathcal{E}_j$ be the union of E_i, for all $i \equiv j \pmod{\eta}$. Let $j^* \in [0, \eta - 1]$ be the index which minimizes weight$(\mathcal{E}_j)$.
3: For every connected component C of $M^* \setminus \mathcal{E}_{j^*}$, let G_C be the subgraph of G consisting of the bricks corresponding to $V[C]$. Find a near-optimal Steiner tree in G_C by dynamic programming.
4: Return the union of the edge-sets of all Steiner trees in the previous step.[2]

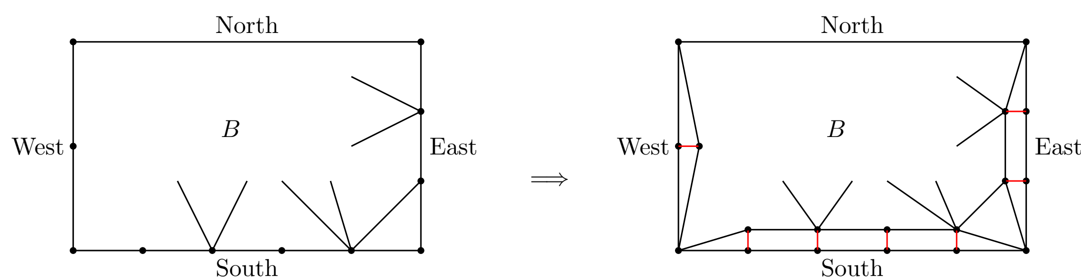


Figure 9.1: Doubling the West, South, and East boundaries of B. The new edges between vertices and their copies have weight 0.

admits a connected *subgraph T with relatively small weight, on which a mortar graph is built. The above framework is then applied to each subinstance.*

As a concrete example, we briefly review the approximation scheme for Steiner tree in [22]. It first computes a 2-approximate Steiner tree T using a *Minimum Spanning Tree (MST)*. Next, it uses the algorithm AUGMENT-CONNECTED (Algorithm 9.3), which receives a planar graph G, a set of terminals Q, a connected subgraph T, and $\epsilon > 0$, and outputs a near-optimal Steiner tree spanning Q in G.

9.4 Doubling Brick Boundaries

In this section, we describe the operation of doubling brick boundaries, which consists of adding to the graph artificial copies of the South, East, and West

[2]There are additional cares to ensure that the output is connected.

boundaries of bricks, and zero-weight edges between corresponding vertices. This is a technical detail to prevent annoying special cases in the Structure Theorem (Theorem 10.3.1).

Let G be a plane graph with edge-weights. Let M be its mortar graph. Let $P = p_0, \cdots, p_\ell$ ($\ell \geq 1$) be any boundary of a brick B. The operation of *doubling the boundary* P is defined as follows. For every vertex $u \in P \setminus \{p_0, p_\ell\}$, create a copy u', and add an edge uu' of weight 0;[3] for every edge uv on P, add an edge $u'v'$ of the same weight as that of uv; and for every edge $uv \in P \times (B \setminus P)$, replace the edge uv by an edge $u'v$ of the same weight. The result of doubling the West, South, and East boundaries of a brick B is given in Figure 9.1. We denote West$'$, South$'$, and East$'$ as the copies of West, South, and East.

Let H be the graph obtained from G by doubling the West, South, and East boundaries of every brick. By the definition of mortar graph (Definition 9.2.3), West, South, and East do not contain edges of R, so no edge of R is duplicated. We observer that the mortar graph of H is inherited from that of G.

Lemma 9.4.1 (Boundary-Doubling Lemma)**.** *A two-edge-connected augmentation for* (G, R) *can be transformed into a two-edge-connected augmentation for* (H, R) *in linear time without increasing the weight, and vice versa.*

Proof. A solution for (G, R) can be transformed into a solution for (H, R) by including all edges vv', which have weight 0. Conversely, a solution S' for (H, R) can be transformed into a solution S for (G, R) as follows: for every boundary edge uv of a brick, uv is included in S if at least one of uv and $u'v'$ is S'. □

9.5 Sphere-Cut Decomposition

We consider a special kind of branch decomposition of plane graphs, called a *sphere-cut decomposition* (see [37]): A *noose* of a plane graph is a Jordan curve that intersects only vertices of the graph and not edges. A *sphere-cut decomposition of width* w is a family of non-crossing nooses each intersecting at most w vertices; the nooses form a binary tree by the enclosure relation, each leaf noose encloses exactly one edge, and each edge is enclosed by a leaf noose. For each noose in the sphere-cut decomposition, we refer to the set of edges enclosed as a *cluster*.

Lemma 9.5.1 (Sphere-Cut Lemma)**.** *Let* G *be a plane graph whose dual graph has diameter* k*. Then* G *has a sphere-cut decomposition of width at most* $2k$*, and it can be computed in linear time.*

[3]To simplify the notation, We define $p'_0 := p_0$ and $p'_\ell := p_\ell$.

The proof of this lemma is a straightforward adaptation from the proof of Lemma 14.6.1 in [61]: We only need to replace the branch decomposition by the sphere-cut decomposition in that proof.

CHAPTER 10

Approximation Scheme

In this chapter, we prove Theorem 7.3.2 that we rewrite as follows:

Theorem 10.0.2 (Main Theorem)**.** *Let G be a plane graph with edge-weights. Let R be a subset of $E[G]$. Let $\epsilon > 0$ be a parameter. The algorithm* AUGMENT(G, R, ϵ) *(Algorithm 10.1) computes in polynomial time a two-edge-connected augmentation S for (G, R) such that* $\text{weight}(S) \leq (1+\epsilon)OPT(G, R)$.

10.1 Preprocessing

In this section, we reduce the instance of two-edge-connected augmentation to subinstances, such that every subinstance admits a connected skeleton of relatively small weight, and that the subinstances can be solved (almost) independently. To prove the Main Theorem, it is then sufficient to prove a related version (Theorem 10.1.1), where we are given in addition a connected subgraph T that contains every edge of R.

Theorem 10.1.1 (Augmentation Theorem)**.** *Let G be a plane graph with edge-weights. Let R be a subset of $E[G]$. Let T be a connected subgraph of G that contains every edge of R. Let $\epsilon > 0$ be a parameter. The algorithm* AUGMENT-CONNECTED(G, R, T, ϵ) *(Algorithm 10.2) computes in polynomial time a two-edge-connected augmentation S for (G, R) such that*

$$\text{weight}(S) \leq (1+\epsilon)OPT(G, R) + \epsilon^2 \cdot \text{weight}(T)\,.$$

We defer the proof of the Augmentation Theorem to later sections, and first show how it implies the Main Theorem. We note that a connected subgraph containing every edge of R might be much more expensive than OPT (see Figure 10.1). So

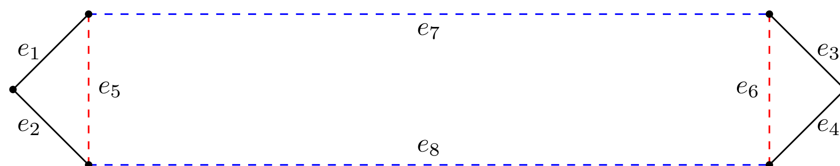


Figure 10.1: In the example, $R = \{e_1, e_2, e_3, e_4\}$. The optimal two-edge-connected augmentation consists of edges e_5 and e_6. However, any Steiner tree connecting the edges of R must include one of the edges e_7 and e_8, whose weight may be much higher than weight($\{e_5, e_6\}$).

Algorithm 10.1 Augment(G, R, T, ϵ), see Theorem 10.0.2

1: $(T_1, \ldots, T_k) \leftarrow$ PC-Partition$(G, R, \epsilon/7)$ ▷ Theorem 9.1.4
2: **for** $i \leftarrow 1$ to k **do**
3: $\quad R_i \leftarrow R \cap T_i$
4: $\quad S_i \leftarrow$ Augment-Connected$(G, R_i, T_i, \epsilon/7)$ ▷ Theorem 10.1.1
5: **return** $(\bigcup_i S_i) \setminus R$

applying the Augmentation Theorem directly would not lead to an approximation scheme. That is why we need to reduce the instance to almost independent subinstances. This is achieved by the PC-Partition algorithm in Section 9.1.2.

Proof of the Main Theorem using the Augmentation Theorem. The output of Algorithm 10.1 is a two-edge-connected augmentation for (G, R), from the Augmentation Theorem and the Partition Theorem (Theorem 9.1.4).

For each instance (G, R_i), by the Augmentation Theorem, we have

$$\text{weight}(S_i) \leq (1 + \epsilon/7)OPT(G, R_i) + (\epsilon/7)^2 \cdot \text{weight}(T_i)\,.$$

Summing the above inequality over i, and again using the Partition Theorem, we deduce that the weight of the output solution is at most $(1 + \epsilon)OPT(G, R)$. □

In the rest of this chapter, we will prove the Augmentation Theorem.

10.2 New Use of Brick Decomposition

For all previous approximation schemes using the brick decomposition, the key was the *structure property*, which says that there exists a near-optimal solution that crosses the boundary of any brick only a bounded number of times.[1] However, this

[1] The bound depends on ϵ.

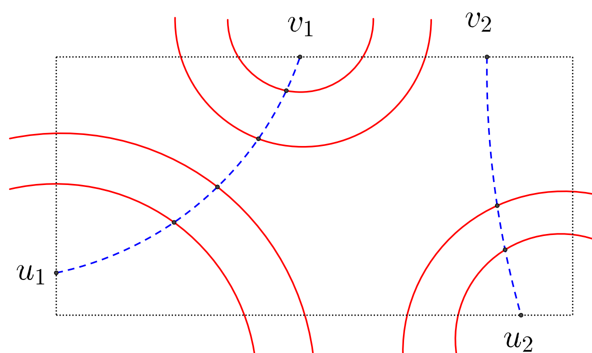


Figure 10.2: The rectangle is a brick. The solid curves represent parts of a near-optimal solution. The dashed curves represent a u_1-to-v_1 Jordan curve and a u_2-to-v_2 Jordan curve inside the brick.

property is unachievable for two-edge-connected augmentation, which is the main difficulty in using the brick decomposition.

Instead, we show that, after a transformation of the instance (namely, the boundary doubling operation in Section 9.4), we have:

New Structure Property. *There exists a near-optimal solution such that, for any brick and any two vertices u, v on the boundary of the brick, there is a u-to-v Jordan curve inside the brick that crosses the near-optimal solution only a bounded number of times.*[2]

This is illustrated in Figure 10.2. To prove the new structure property, we *add boundary cycles* and *reduce nesting* in each brick. See Section 10.3.

To make use of the new structure property, we combine it with the *sphere-cut decomposition* (see Section 9.5). Recall that in the framework of the approximation schemes of Section 9.3, each subinstance contains a bounded number of levels of bricks, thus its mortar graph has a branch decomposition of bounded width. We observe that the branch decomposition here has the special form of a sphere-cut decomposition: each cluster of edges is precisely the set of edges enclosed by a Jordan curve J that intersects the mortar graph a bounded number of times. This is where the new structure property comes in: each segment of J traversing a brick can be replaced by a curve that intersects the near-optimal solution a bounded number of times. This yields a new Jordan curve J' that has a bounded number of intersections with the near-optimal solution.

[2]The bound depends on ϵ.

Algorithm 10.2 AUGMENT-CONNECTED(G, R, T, ϵ), see Theorem 10.1.1

1: Compute a mortar graph M of G based on R and T (Lemma 9.2.4).
2: Do BFS in the planar dual M^* starting from an arbitrary vertex r. Define the *level* of a vertex of M^* as its BFS distance from r. Let E_i denote the set of edges of M^* whose two endpoints are at level i and level $i+1$, respectively. Let $\eta = \Theta(1/\epsilon^3)$. For $j = 0, 1, \ldots, \eta - 1$, let $\mathcal{E}_j$ be the union of E_i, for all $i \equiv j \pmod{\eta}$. Let $j^* \in [0, \eta - 1]$ be the index which minimizes weight$(\mathcal{E}_j)$.
3: Let H be the graph obtained from G by doubling East, South, and West boundaries of every brick (See Section 9.4).
4: For every connected component C of $M^* \setminus \mathcal{E}_{j^*}$, let H_C be the subgraph of H consisting of the bricks corresponding to $V[C]$. Find a near-optimal two-edge-connected augmentation in H_C by dynamic programming (See Section 10.4).
5: Return the union of the edge-sets of all two-edge-connected augmentations in the previous step.

The above properties enable us to design a *dynamic program (DP)*, see Section 10.4. For each cluster of the sphere-cut decomposition, the DP enumerates all possible intersections of the unknown near-optimal solution with the partially unknown Jordan curve J'. The DP also enumerates all possible connectivity structures of the part of the near-optimal solution inside J'. Note that there may be some edges of the graph that are in the parent cluster but not in the child clusters, so the DP must do a bit of extra work to go from tables for the children to the table for the parent (see Section 10.4.2).

The algorithm for Theorem 10.1.1 is given in Algorithm 10.2. Compared with Algorithm 9.3, the brick decomposition is slightly different (see Section 9.2); the boundary doubling operation is new (see Section 9.4); and the dynamic program is novel (see Section 10.4), because it is based on the new structure property.

10.3 Structure Theorem

The Structure Theorem is the key to the approximation scheme for two-edge-connected augmentation. It is a generalization of the *new structure property* in Section 10.2.

Theorem 10.3.1 (Structure Theorem)**.** *Let G be a plane graph with edge-weights. Let R be a subset of $E[G]$. Let $\epsilon > 0$ be a parameter. Let M be the mortar graph of G. Let H be the graph obtained from G by doubling the South, East, and West boundaries of every brick.*

For any two-edge-connected augmentation S_0 for (H, R), there is a two-edge-connected augmentation S for (H, R) such that:

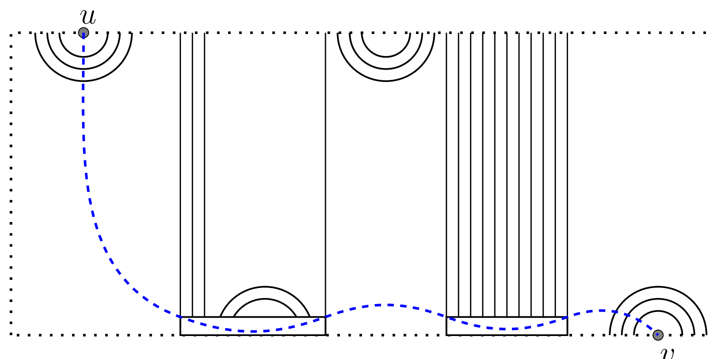


Figure 10.3: The rectangle is a brick. The solid segments represent the modified solution inside the brick. The dashed curve from u to v has few crossings with the modified solution.

- $\text{weight}(S) \leq (1+\epsilon)\text{weight}(S_0) + 4\sum_{brick\ B} \text{weight}(\text{East}_B \cup \text{West}_B)$;
- *For any brick and any two vertices u, v on the boundary of the brick, there is a u-to-v Jordan curve inside the brick that has $O(1/\epsilon^4)$ crossings with S, all occurring at vertices.*

The proof of the Structure Theorem consists in modifying the initial solution so that any pair of vertices on the boundary of a brick can be connected by a curve that has few crossings with the modified solution. Figure 10.3 shows the kind of curve we use. It starts at a given vertex u on the brick boundary, traverses nested paths, then bypasses South-to-North paths using *South cycles* (cycles formed by parts of the South boundary and their duplicates), and finally again traverses nested paths to reach the given vertex v on the brick boundary. In order to achieve a small number of crossings, we must ensure that the number of nested paths is small and that only a small number of South cycles are used to bypass the South-to-North paths.

The construction of the solution S works on each brick in turn, modifying the initial solution S_0 inside that brick. The key to prove the Structure Theorem is the following Structure Proposition, which can be viewed as a local version of the Structure Theorem.

Proposition 10.3.2 (Structure Proposition)**.** *Let S be any two-edge-connected augmentation for (H, R). Let B be a brick in H. Let F be the set of edges of S that are in the interior of B. Then there is a set of edges F_3 in B with the following properties:*

Feasibility: *$(S \setminus F) \cup F_3$ is a two-edge-connected augmentation for (H, R);*

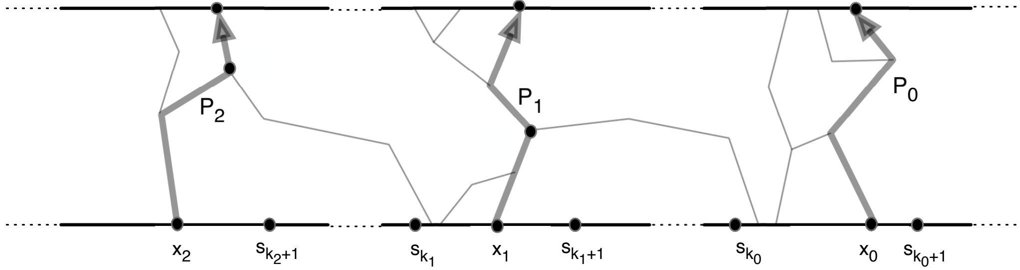


Figure 10.4: Extracted from [22]. The North and South boundaries are indicated by horizontal lines. The paths P_0, P_1, and P_2 are indicated by thick gray lines.

Near-Optimality: $\text{weight}(F_3) \leq (1+\epsilon)\text{weight}(F) + 4\text{weight}(\text{East}_B \cup \text{West}_B)$*;*

Bounded-Crossings Property: *For any two vertices u, v on the boundary of B, there exists a u-to-v Jordan curve inside the brick that has at most $O(1/\epsilon^4)$ crossings with F_3, all occurring at vertices.*

Proof of the Structure Theorem using the Structure Proposition. Let S be initialized as S_0. For each brick B of H in turn, we update S by $(S \setminus F) \cup F_3$, where F is the set of edges of S that are in the interior of B, and F_3 is the set of edges obtained from the Structure Proposition. The resulting S is a two-edge-connected augmentation. The weight of S is increased by at most

$$\sum_{\text{brick } B} \Big(\epsilon \cdot \text{weight}(F) + 4\text{weight}(\text{East}_B \cup \text{West}_B)\Big).$$

The Structure Theorem follows since the sets F's are disjoint subsets of S_0. □

In the rest of this section, we prove the Structure Proposition.

10.3.1 Construction

To construct F_3 from F in the Structure Proposition, there are three steps as follows.

Step 1: Modify F into F_1 by Adding East and West Cycles. We add to F all the edges on the two cycles $\text{East} \circ \text{East}'$ and $\text{West} \circ \text{West}'$, where East' and West' are copies of the East and West boundaries during the boundary doubling operation (see Section 9.4), and we remove from F all the edges inside the two cycles.

Prune the result by removing unnecessary edges that are in the interior of the brick, thus obtaining a forest. Let F_1 be the result.

Step 2: Modify F_1 into F_2 by Adding South Cycles. First, we greedily define a collection of disjoint South-to-North paths $P_0, \ldots, P_t$ (with associated indexes $k_0, \ldots, k_t$) using the approach in [22]: Let $s_0, \ldots, s_k$ be the vertices of South in the definition of mortar graph (Definition 9.2.3). Define P_0 as the East boundary. Assume P_i is a path in F_1 from some segment South$[s_j, s_{j+1})$ to North. Then P_{i+1} is defined to be the easternmost path in F_1 from South$[s_0, s_j)$ to North that does not go through any vertices of South or North or P_i. See Fig. 10.4.

Some associated notations: x_i is the start vertex of P_i; k_i is the integer j such that $x_i \in$ South$[s_j, s_{j+1})$; H_i is the subgraph of F_1 that is strictly enclosed by P_i, P_{i+1}, and the segments of South and North. Let t be the last index for which P_t is defined. Note that $t \leq k = O(1/\epsilon^4)$.

For each $i \in [0, t]$, let x'_i and s'_{k_i} be the copies of x_i and s_{k_i} during the boundary doubling operation (Section 9.4). In the construction of the solution, for each i, we add to F_1 the cycle $C_i = \text{South}[s_{k_i}, x_i] \circ x_i x'_i \circ \text{South}'[x'_i, s'_{k_i}] \circ s'_{k_i} s_{k_i}$. The cycles $\{C_i\}_i$ are called *South cycles*. See Figure 10.5. We remove from F_1 all the edges inside every C_i.

Prune the result by removing unnecessary edges that are in the interior of the brick and do not belong to any P_i, thus obtaining a forest. Let F_2 be the result.

Step 3: Modify F_2 into F_3 by Reducing Nesting. A *South arch* A is a path in F_2 whose endpoints u and v are on South and whose other vertices are all strictly in the interior of the brick. The u-to-v path along South is called the *base* of A. We define the *subgraph (strictly) enclosed by* A as the subgraph induced by the edges of F_2 that are (strictly) inside the cycle $A \circ \textit{base}(A)$. For a South arch A, the *South arch-emptying operation* is to remove from F_2 the edges in subgraph strictly enclosed by A, and to add to F_2 the edges on the base of A. We define the *depth* of South arches by induction: For every maximally enclosing South arch, its depth is 0; For every South arch A of depth d ($d \geq 0$), consider the subgraph strictly enclosed by A, and define the depth of every maximally enclosing South arch in this subgraph to be $d + 1$. In the construction of the solution, we apply the South arch-emptying operation to every South arch at depth $\kappa := \lceil 1/\epsilon \rceil$. See Figure 10.6.

Similarly, we define *North arch* and *North arch-emptying operation*, except that since the North boundary may contain edges from R, in the North arch-emptying operation, instead of adding all the edges of the u-to-v path along North, we add the edges of the u-to-v path along North that are not in R (since the solution is supposed to be an augmentation of R). Again, we apply the North arch-emptying operation to every North arch at depth κ.

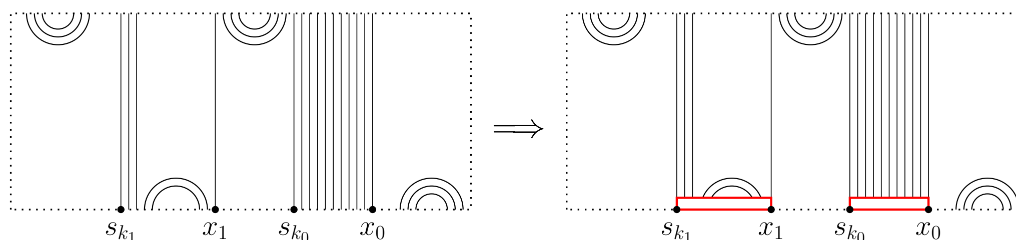


Figure 10.5: Adding South cycles (in bold) into the solution.

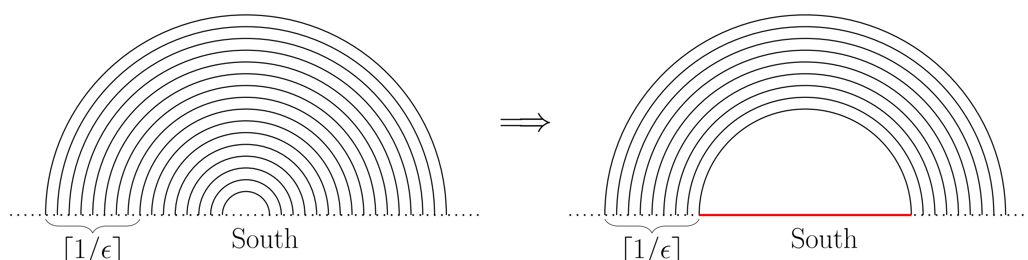


Figure 10.6: Reducing nesting: when the solution contains more than $\lceil 1/\epsilon \rceil$ nested paths, we add a piece of the South boundary (in bold) and empty the cycle thus created.

> Prune the result by removing unnecessary edges that are in the interior of the brick, thus obtaining a forest. Let F_3 be the result. We similarly define *arch* and *depth* in the subgraph F_3.

10.3.2 Analysis

In this section, we prove the Structure Proposition.

Proof of Feasibility. The edges that are removed during the construction of F_3 are either unnecessary edges, or edges inside cycles. By the following lemma, the result $(S \setminus F) \cup F_3$ is a two-edge-connected augmentation for (H, R).

Lemma 10.3.3 ([21])**.** *Let S be a two-edge-connected augmentation for (H, R). Let C be a non-self-crossing cycle of S strictly enclosing no edge of R. Let S' be the subset of S obtained by removing the edges of S that are strictly enclosed by C. Then S' is again a two-edge-connected augmentation for (H, R).*

Proof of Near-Optimality. The following lemma bounds the costs of arch-emptying operations with respect to some maximally enclosing arch A_0.

Lemma 10.3.4. *Let A_0 be a maximally enclosing arch in F_2. Let $F_2(A_0)$ (resp. $F_3(A_0)$) be the subgraph of F_2 (resp. F_3) enclosed by A_0. We have:*

$$\text{weight}(F_3(A_0)) - \text{weight}(F_2(A_0)) \leq 2\epsilon \cdot \text{weight}(F_2(A_0))\,.$$

Proof. For every $1 \leq i \leq \kappa$, define $\mathcal{A}_i$ as the set of depth-i arches in $F_2(A_0)$. Let $\mathcal{A} = \bigcup_i \mathcal{A}_i$. Since the arch emptying operations are applied to arches of depth κ, $F_3(A_0) - F_2(A_0)$ is at most $\sum_{A \in \mathcal{A}_\kappa} \text{weight}(base(A))$, which we want to bound.

Every $A \in \mathcal{A}_\kappa$ is enclosed by exactly κ arches from $\mathcal{A}$: one from each $\mathcal{A}_i$. We charge the weight of $base(A)$ to each of the κ arches. Thus the total charge for all $A \in \mathcal{A}_\kappa$ is $\kappa \sum_{A \in \mathcal{A}_\kappa} \text{weight}(base(A))$.

On the other hand, every arch $A' \in \mathcal{A}$ is charged by the arches $A \in \mathcal{A}_\kappa$ that are in the subgraph enclosed by A'. Notice that the bases of the arches charging A' are disjoint sub-segments of $base(A')$. So the total weight of these bases is at most the weight of $base(A')$, which is at most $(1+\epsilon)\text{weight}(A')$ by the definition of mortar graph (Definition 9.2.3). Thus we have:

$$\kappa \sum_{A \in \mathcal{A}_\kappa} \text{weight}(base(A)) \leq \sum_{A' \in \mathcal{A}} (1+\epsilon) \cdot \text{weight}(A') \leq (1+\epsilon) \cdot \text{weight}(F_2(A_0))\,.$$

Therefore, $\sum_{A \in \mathcal{A}_\kappa} \text{weight}(base(A)) \leq 2\epsilon \cdot \text{weight}(F_2(A_0))$, as required. □

Now we bound the weight of F_3. First, since F_1 is obtained from F by adding the East and West boundaries and their copies East$'$ and West$'$, we have:

$$\text{weight}(F_1) \leq \text{weight}(F) + 2\text{weight}(\text{East}_B \cup \text{West}_B)\,. \tag{10.1}$$

Next, from the definition of mortar graph, for every i, $\text{weight}(\text{South}[s_{k_i}, x_i]) \leq \epsilon \cdot \text{weight}(P_i)$. So the weight of the South cycle C_i is at most $2\epsilon \cdot \text{weight}(P_i)$. Since $\{P_i\}_i$ are disjoint paths in F_1, $\sum_i \text{weight}(P_i) \leq \text{weight}(F_1)$. Therefore, we have:

$$\text{weight}(F_2) \leq (1+2\epsilon)\text{weight}(F_1)\,. \tag{10.2}$$

Finally, note that every edge of F_2 is enclosed by at most two maximally enclosing arches.[3] Applying Lemma 10.3.4 to all maximally enclosing arches, and summing them up, we have:

$$\text{weight}(F_3) \leq (1+4\epsilon) \cdot \text{weight}(F_2)\,. \tag{10.3}$$

Combining Equations (10.1), (10.2), (10.3), we have:

$$\text{weight}(F_3) \leq (1+7\epsilon)(\text{weight}(F) + 2\text{weight}(\text{East}_B \cup \text{West}_B)).$$

The statement follows by replacing ϵ by $\epsilon' = \epsilon/7$.

[3] At most one maximally enclosing North arch and at most one maximally enclosing South arch.

Proof of Bounded-Crossing Property.

Definition 10.3.5. *For any two vertices u, v of the brick, consider a u-to-v Jordan curve inside the brick that has the minimum number of crossings with F_3, all occurring at vertices. Define the distance measure $\delta(u, v)$ as the number of crossing with F_3 on this curve (excluding u and v). For a vertex u of the brick and a subset X of vertices of the brick, define $\delta(u, X) = \min_{v \in X} \delta(u, v)$. For two subsets X and Y of vertices of the brick, define $\delta(X, Y) = \min_{u \in X, v \in Y} \delta(u, v)$.*

From the definition, $\delta(u, w) \leq \delta(u, v) + \delta(v, w) + 1$, for every vertices u, v, w of the brick.

We rewrite the *Bounded-Crossings Property* as follows: for every vertices u, v on the boundary of the brick, $\delta(u, v) = O(1/\epsilon^4)$.

Fact 10.3.6. *The nodes of the East cycle (resp. West cycle) are on the same face in F_3, and the nodes of any South cycle are on the same face in F_3.*

Step 1 of the construction enables us to reduce the case that $u, v \in \text{South} \cup \text{West} \cup \text{North} \cup \text{East}$ to the case that $u, v \in \text{South} \cup \text{North}$. To see this, let u, v be any boundary vertices. We note that u is on the same face with some vertex from $\text{South} \cup \text{North}$: when u is on East (resp. West), from Fact 10.3.6, u is on the same face with the intersection vertex of East (resp. West) and South. Thus $\delta(u, \text{South} \cup \text{North}) = 0$. Similarly, $\delta(v, \text{South} \cup \text{North}) = 0$. So $\delta(u, v)$ is at most 2 plus the distance between a pair of vertices from $\text{South} \cup \text{North}$. To prove the *Bounded-Crossing Property*, it only remains to prove Lemma 10.3.7 as follows.

Lemma 10.3.7. *For any vertices u, v on $\text{South} \cup \text{North}$, $\delta(u, v) = O(1/\epsilon^4)$.*

To prove Lemma 10.3.7, we need Lemmas 10.3.9, 10.3.11 and 10.3.12. Lemmas 10.3.9 and 10.3.12 are based on Lemma 10.3.8.

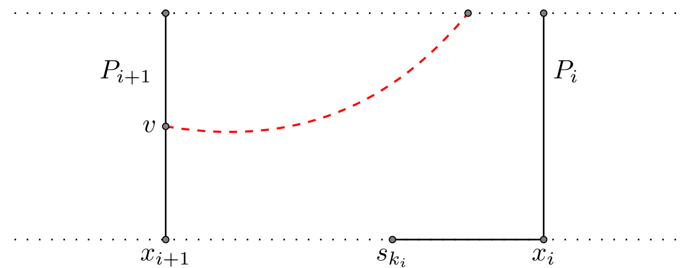
Lemma 10.3.8. *Let A be any arch in F_3. Let u be any vertex on the base of A. We have $\delta(u, A) = O(1/\epsilon)$.*

Proof. Consider a set of arches $\{A_i\}_{0 \leq i \leq \ell}$ in F_3, where $A_0 = A$, and every A_i $(i \geq 1)$ is the maximal enclosing arch in the subgraph strictly enclosed by A_{i-1} such that $u \in base(A_i)$. Let ℓ be the last index for which A_ℓ is defined. From Step 3 of the construction, $\ell = O(1/\epsilon)$. Let $u_\ell \in A_\ell$ be a vertex that is on the same face with u in F_3. For every $i = \ell, \ldots, 1$, let $u_{i-1} \in A_{i-1}$ be a vertex that is on the same face with u_i in F_3. The vertices $\{u_i\}_i$ always exist from the construction of $\{A_i\}_i$. Thus we obtain $u_0 \in A$ with $\delta(u, u_0) \leq \ell = O(1/\epsilon)$. □

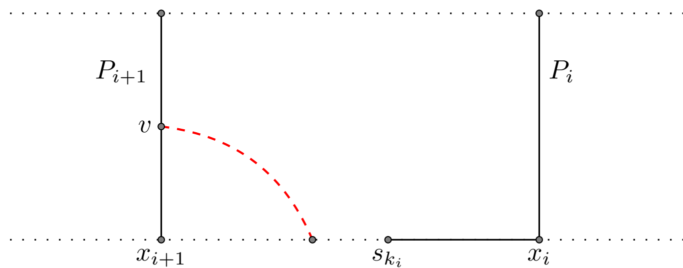
Lemma 10.3.9. *For every vertex $u \in \text{North}$, $\delta(u, \text{South}) = O(1/\epsilon)$.*

Proof. If u is not on the base of any North arch, then u is on the same face of F_3 with some vertex from South, because there is no path in F_3 that starts at an internal vertex of East or West from Step 1 of the construction. Thus $\delta(u, \text{South}) = 0$.

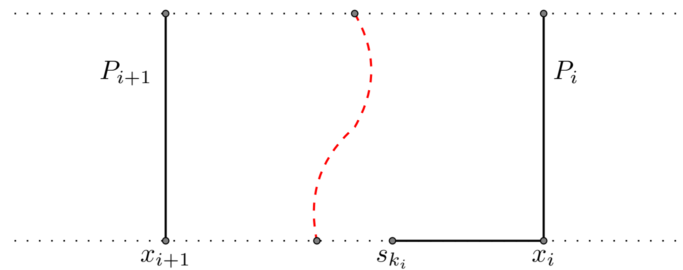
If u is on the base of some North arch, let A be the maximally enclosing North arch in F_3 such that $u \in base(A)$. By Lemma 10.3.8, there exists some vertex v on the arch A such that $\delta(u, v) = O(1/\epsilon)$. Using a similar argument as before, we deduce that v is on the same face of F_3 with some vertex from South. Therefore, $\delta(u, \text{South}) = O(1/\epsilon)$. □



(a)



(b)



(c)

Figure 10.7: The dashed paths are forbidden by the construction of $\{P_i\}_i$.

Fact 10.3.10. *For any $i < t$, the subgraph H_i has at most one path that goes eastwards starting from an internal vertex of P_{i+1}; and if such path exists, it must end in some node of* $\text{South}[s_{k_i}, x_i] \cup P_i$ *(see Figs. 10.7a and 10.7b). As a consequence, every vertex w on P_{i+1} has $\delta(w, x_{i+1}) \leq 1$.*

Lemma 10.3.11. *For every $i < t$, $\delta(x_{i+1}, x_i) \leq 2$.*

Proof. First, by definition of P_{i+1}, one can see that in H_i, x_{i+1} is on the same face as some node w of $\text{South}[s_{k_i}, x_i] \cup P_i$ (see Figs. 10.7b and 10.7c). There are two cases, depending on whether w is on $\text{South}[s_{k_i}, x_i]$ or on P_i.

Case 1: $w \in \text{South}[s_{k_i}, x_i]$. Then there is a South cycle including w and its copy w'. Since in F_3, x_{i+1} is on the same face as w', and w' is on the same face as x_i (Fact 10.3.6), we have $\delta(x_{i+1}, x_i) \leq 1$.

Case 2: $w \in P_i$. Then in F_3, the node x_{i+1} is on the same face as w. By Fact 10.3.10, $\delta(w, x_i) \leq 1$. Thus $\delta(x_{i+1}, x_i) \leq 2$. □

Lemma 10.3.12. *For every vertex $u \in$* South, $\min_i \delta(u, x_i) = O(1/\epsilon)$.

Proof. We observe that $x_t, \ldots, x_0$ induce a partition of South. So we only need to prove that, for every $i < t$ and every vertex $u \in \text{South}(x_{i+1}, x_i]$, $\delta(u, x_i) = O(1/\epsilon)$.

When $u \in \text{South}[s_{k_i}, x_i]$, $\delta(u, x_i) = 0$ by Fact 10.3.6. Next, consider a vertex $u \in \text{South}(x_{i+1}, s_{k_i})$. Since $\delta(x_{i+1}, x_i) \leq 2$ (Lemma 10.3.11) and every vertex w on P_i is such that $\delta(w, x_i) \leq 1$ (Fact 10.3.10), it is sufficient to show that either $\delta(u, x_{i+1}) = O(1/\epsilon)$, or $\delta(u, P_i) = O(1/\epsilon)$. This is done by a case-by-case analysis as follows (again using Fact 10.3.10).

When there is an internal vertex v of P_{i+1} such that there is a (unique) path in H_i between v and $\text{South}[s_{k_i}, x_i] \cup P_i$, there are two cases:

1. There is a path between v and $\text{South}[s_{k_i}, x_i]$. Define the arch A to be the concatenation of this path and a segment of P_{i+1}. See the first figure of Fig. 10.8.

2. There is a path between v and P_i. Define the arch A to be the concatenation of this path and segments of P_i and of P_{i+1}. See the second figure of Fig. 10.8.

When there is no such vertex v, consider the maximally enclosing South arch containing u. There are two cases:

3. This arch does not intersect P_i. Define A to be this arch. See the third figure of Fig. 10.8.

4. This arch intersects P_i. Define the arch A to be the concatenation of the part of this arch in H_i and a segment of P_i. See the fourth figure of Fig. 10.8.

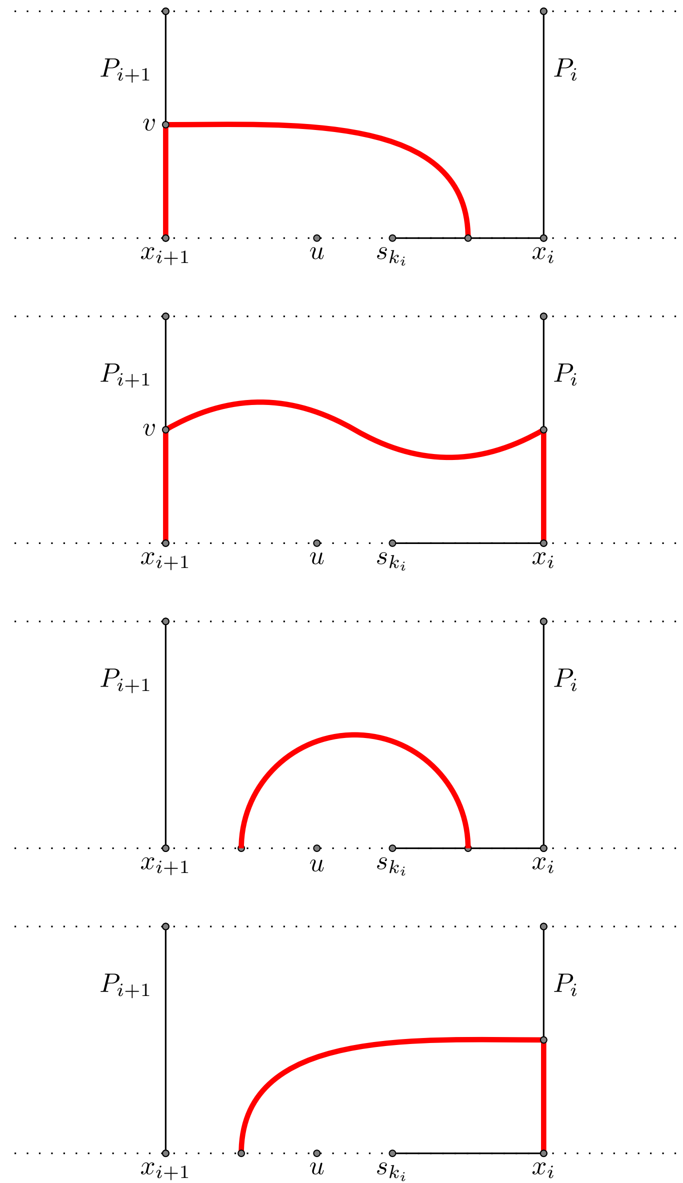


Figure 10.8: For each case, the arch A is in bold.

We note that in the first two cases, no internal vertex of A has connection to South in H_i. Therefore, every vertex of A is on the same face with x_{i+1}. In the third case, one can see that every vertex of A is on the same face with x_{i+1} (the face above the arch). In the fourth case, every vertex of A is either on the same face with x_{i+1} (the face above the arch), or belongs to P_i.

We conclude that in all cases, every vertex of A is either on the same face with

x_{i+1} or belongs to P_i. Since u is on the base of A, by Lemma 10.3.8, $\delta(u, A) = O(1/\epsilon)$. Therefore, we have either $\delta(u, x_{i+1}) = O(1/\epsilon)$, or $\delta(u, P_i) = O(1/\epsilon)$.

This completes the proof of the lemma. □

Proof of Lemma 10.3.7. Let u, v be any vertices on South ∪ North. Let u_1 (resp. v_1) be the vertex on South which minimizes $\delta(u, u_1)$ (resp. $\delta(v, v_1)$). Then

$$\delta(u, v) \leq \delta(u, u_1) + \delta(u_1, v_1) + \delta(v_1, v) + 2.$$

By Lemma 10.3.9, both $\delta(u, u_1)$ and $\delta(v, v_1)$ are $O(1/\epsilon)$. So we only need to show that $\delta(u_1, v_1) = O(1/\epsilon^4)$.

Let $i \leq t$ be the index that minimizes $\delta(u_1, x_i)$, and let $j \leq t$ be the index that minimizes $\delta(v_1, x_j)$. Then

$$\delta(u_1, v_1) \leq \delta(u_1, x_i) + \delta(x_i, x_j) + \delta(x_j, v_1) + 2,$$

and

$$\delta(x_i, x_j) \leq |j - i| + \sum_{\ell=\min(i,j)}^{\max(i,j)-1} \delta(x_\ell, x_{\ell+1}).$$

By Lemma 10.3.12, $\delta(u_1, x_i) = O(1/\epsilon)$, $\delta(v_1, x_j) = O(1/\epsilon)$, and by Lemma 10.3.11, $\delta(x_\ell, x_{\ell+1}) \leq 2$. Recall that $t = O(1/\epsilon^4)$. Therefore, $\delta(u_1, v_1) = O(1/\epsilon^4)$. □

10.4 Dynamic Programming

In this section, we design a *dynamic program (DP)* to solve the two-edge-connected augmentation problem on the instance (H, R), given that the dual of the mortar graph has bounded diameter. From the Structure Theorem, in order to get a near-optimal solution, we may restrict attention to solutions that satisfy the property defined there. A dynamic program computes the best among all such solutions.

Theorem 10.4.1 (Dynamic-Programming Theorem)**.** *Let G be a plane graph with edge-weights. Let R be a subset of $E[G]$. Let $\epsilon > 0$ be a parameter. Let M be the mortar graph of G. Let H be the graph obtained from G by doubling the South, East, and West boundaries of every brick.*

Assume that the dual graph of M has diameter $O(1/\epsilon^3)$. Then there is an algorithm that computes in polynomial time a two-edge-connected augmentation S for (H, R) such that:

$$\text{weight}(S) \leq (1 + \epsilon) OPT(H, R) + 4 \sum_{\textit{brick } B} \text{weight}(\text{East}_B \cup \text{West}_B)\,.$$

10.4.1 Specification of DP Table

In this section, we define the *index* of the DP table and the *value* at an index.

By the Sphere-Cut Lemma (Lemma 9.5.1), M has a sphere-cut decomposition $\mathcal{SC}$ of width $O(1/\epsilon^3)$ which can be computed in linear time. The first index of the DP table is a cluster E of $\mathcal{SC}$, which is a subset of edges of M.

Let $S_0 \subseteq E[H]$ be the optimal two-edge-connected augmentation for (H, R). Let $S \subseteq E[H]$ be defined in the Structure Theorem (Theorem 10.3.1). Then S is a near-optimal two-edge-connected augmentation. We remove unnecessary edges from S to make it minimal. By Lemma 8.2.2, every connected component in $R \cup S$ is two-edge-connected. For every cluster E of $\mathcal{SC}$, let J_E be the noose enclosing E and of minimum number of crossings with $R \cup S$ (all occurring at vertices), breaking ties by choosing the minimally enclosing one.[4] It is easy to show that the family of nooses $\{J_E\}_{E \in \mathcal{SC}}$ is non-crossing.

Lemma 10.4.2. *For every cluster E of $\mathcal{SC}$, J_E intersects $O(1/\epsilon^7)$ vertices of $R \cup S$.*

Proof. Since $\mathcal{SC}$ has width $O(1/\epsilon^3)$, there is a noose enclosing E that has $O(1/\epsilon^3)$ intersections with M. From one intersection to the next, it goes across a single brick, and by the Structure Theorem (Theorem 10.3.1), the part inside this brick can be chosen so as to have $O(1/\epsilon^4)$ intersections with S, hence $O(1/\epsilon^4)$ intersections with $R \cup S$, since no edge of R is in the interior of a brick. This results in a noose enclosing E that has $O(1/\epsilon^7)$ intersections with $R \cup S$. □

Let $Q^* \subseteq V[H]$ denote the (unknown) set of $O(1/\epsilon^7)$ intersection vertices of J_E with $R \cup S$. The second index of the DP table is a subset $Q \subseteq V[H]$ of size $O(1/\epsilon^7)$.

Next, we need a concise representation of the connectivity structure of the part of $R \cup S$ inside J_E. Let R_E (resp. Γ^*) denote the set of edges of R (resp. S) that are inside J_E. Define a forest F_0^* from $R_E \cup \Gamma^*$ by contracting every two-edge-connected component into a node. A node of F_0^* is called *internal* if its corresponding two-edge-connected component in $R_E \cup \Gamma^*$ does not contain any node from Q^*, i.e., the component is strictly inside J_E. We then define a forest F^* from F_0^* by removing internal nodes that are singletons and splicing internal nodes of degree 2. By the construction, F^* has at most $|Q^*|$ non-internal nodes, and every internal node has degree at least 3.[5] So F^* has at most $2|Q^*| - 2$ nodes. The third index of the DP table is a forest F of at most $2|Q| - 2$ nodes. Moreover,

[4] Since the noose is a geometric object, it is not uniquely defined, but a discrete formulation can be given using the *face-vertex incidence graph* (see [61]).

[5] There cannot be internal nodes of degree 1, because $R \cup S$ is a collection of two-edge-connected components.

there is a map ψ^* giving the natural many-to-one map from Q^* to nodes of F^*. The fourth index of the DP table is a map ψ from Q to $V[F]$. To summarize:

Definition 10.4.3 (DP index)**.** *An* index of the DP table*, also called a* DP index*, contains the following:*

- *E: a cluster of the sphere-cut decomposition $\mathcal{SC}$;*
- *Q: a subset of $V[H]$ of size $O(1/\epsilon^7)$;*
- *F: a forest of size at most $2|Q|-2$;*
- *ψ: a map from Q to $V[F]$.*

In addition, the triple (Q, F, ψ) as defined above is called a partial DP index.[6]

Before defining the value at a DP index, we need the concept of *consistency* to relate a solution $\Gamma \subseteq E[H]$ with a DP index.

Definition 10.4.4 (consistency)**.** *Let (E, Q, F, ψ) be an index of the DP table. We say that a subset Γ of $E[H]$ is* consistent *with (E, Q, F, ψ) if, for every node $a \in V[F]$, there exists a connected subgraph H_a of $\Gamma \cup R_E$ such that the endpoints of every edge in $R_E \cap H_a$ are two-edge-connected in H_a; and for every edge $ab \in E[F]$, there exists a simple path I_{ab} in $\Gamma \cup R_E$ connecting H_a and H_b, such that the following holds:*

1. *Every vertex $u \in Q$ belongs to $H_{\psi(u)}$;*
2. *For every edge $uv \in R_E$ such that u and v are not two-edge-connected in $R_E \cup \Gamma$, there is exactly one edge ab from F such that $uv \in I_{ab}$.*

Let $\mathcal{H} = \{H_a\}$ and $\mathcal{I} = \{I_{ab}\}$.

We observe that Γ^* is consistent with (E, Q^*, F^*, ψ^*), and that any Γ that is consistent with $(M, \emptyset, \emptyset, \emptyset^{\emptyset})$ is a two-edge-connected augmentation for (H, R).

For every DP index (E, Q, F, ψ), define its *value* $DP(E, Q, F, \psi)$ as the minimum weight among a collection of Γ's, such that:

1. Every Γ in this collection is consistent with (E, Q, F, ψ); and
2. If $(Q, F, \psi) = (Q^*, F^*, \psi^*)$, then Γ^* is in this collection.

In order to prove the Dynamic-Programming Theorem (Theorem 10.4.1), we only need to find a polynomial-time algorithm to fill in the DP table and to output the value $\mathrm{DP}(M, \emptyset, \emptyset, \emptyset^{\emptyset})$. [7]

[6] Note that the definition of (Q, F, ψ) is independent of E.

[7] The DP outputs the *value* of a solution, not the solution itself; but it is easy to enrich the DP in the standard manner so that it also outputs the solution achieving the value.

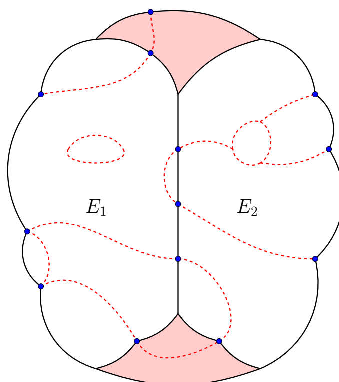


Figure 10.9: The solid curves represent J_E, J_{E_1}, and J_{E_2}: J_E is the outermost boundary; J_{E_1} is the boundary of the white face on the left, and J_{E_2} is the boundary of the white face on the right. The dark areas belong to the hole region. The dashed curves represent $R \cup S$ inside J_E. The solid points represent vertices from $Q^* \cup Q_1^* \cup Q_2^*$.

10.4.2 From Children to Parent

Let E be a cluster of $\mathcal{SC}$. Let E_1 and E_2 be its child clusters. Let $Q^*, Q_1^*, Q_2^* \subseteq V[H]$ be the sets of intersections of $R \cup S$ with J_E, J_{E_1}, J_{E_2}. The *hole region* is the area inside J_E but outside J_{E_1} and J_{E_2} in the plane.[8] See Figure 10.9. We observe that the hole region cannot contain edges from R.

Let $\hat{\Gamma}^*$ denote the set of edges of S in the hole region. Let $\hat{Q}^*$ denote the set of intersections of S with the boundary of the hole region. We have $\hat{Q}^* \subseteq Q^* \cup Q_1^* \cup Q_2^*$. Thus $|\hat{Q}^*| = O(1/\epsilon^7)$. Similar to Section 10.4.1, the connectivity structure of $\hat{\Gamma}^*$ can be represented by a forest $\hat{F}^*$ of size at most $2|\hat{Q}^*| - 2$, together with a map $\hat{\psi}^*$ from $\hat{Q}^*$ to $V[\hat{F}^*]$.

We use a side table T for the computation at hole regions. This table is indexed by a partial DP index $(\hat{Q}, \hat{F}, \hat{\psi})$. The concept of consistency can be extended to $(\hat{Q}, \hat{F}, \hat{\psi})$ by setting $E = \emptyset$ (therefore $R_E = \emptyset$). The *value* $T(\hat{Q}, \hat{F}, \hat{\psi})$ is defined as the minimum weight of any $\hat{\Gamma}$ that is consistent with $(\hat{Q}, \hat{F}, \hat{\psi})$.

Definition 10.4.5 (compatibility). *Let (Q_1, F_1, ψ_1), (Q_2, F_2, ψ_2), and $(\hat{Q}, \hat{F}, \hat{\psi})$ be partial DP indexes. Let (Q, F, ψ) be a partial DP index such that $Q \subseteq Q_1 \cup Q_2 \cup \hat{Q}$. We say that (Q, F, ψ) is* compatible *with (Q_1, F_1, ψ_1), (Q_2, F_2, ψ_2), and $(\hat{Q}, \hat{F}, \hat{\psi})$ if F and ψ can be constructed as follows:*

1. *F is initialized as $F_1 \cup F_2 \cup \hat{F}$;*

[8] Note that J_E, J_{E_1}, and J_{E_2} are non-crossing.

2. *For every vertex $u \in Q_1 \cap Q_2$, merge the nodes $\psi_1(u)$ and $\psi_2(u)$ in F (idem for every $u \in Q_1 \cap \hat{Q}$ and every $u \in Q_2 \cap \hat{Q}$);*

3. *Contract two-edge-connected components in F; Let $\psi : Q \to F_0$ be the natural extension of ψ_1, ψ_2, $\hat{\psi}$;*

4. *Remove internal nodes[9] of F that are singletons and splicing internal nodes of F of degree 2.*

Fact 10.4.6. *The partial DP indexes of the near-optimal solution are compatible: (Q^*, F^*, ψ^*) is compatible with (Q_1^*, F_1^*, ψ_1^*), (Q_2^*, F_2^*, ψ_2^*), and $(\hat{Q}^*, \hat{F}^*, \hat{\psi}^*)$.*

Lemma 10.4.7. *For $i \in \{1, 2\}$, let $\Gamma_i \subseteq E[H]$ be consistent with a DP index (E_i, Q_i, F_i, ψ_i). Let $\hat{\Gamma} \subseteq E[H]$ be consistent with a partial DP index $(\hat{Q}, \hat{F}, \hat{\psi})$. Let (Q, F, ψ) be a partial DP index that is compatible with (Q_1, F_1, ψ_1), (Q_2, F_2, ψ_2), and $(\hat{Q}, \hat{F}, \hat{\psi})$. Then $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \hat{\Gamma}$ is consistent with (E, Q, F, ψ).*

Proof. For $i \in \{1, 2\}$, Since Γ_i is consistent with (E_i, Q_i, F_i, ψ_i), there exists $\mathcal{H}_i$ and $\mathcal{I}_i$ according to Definition 10.4.4. Similarly, since $\hat{\Gamma}$ is consistent with $(\hat{Q}, \hat{F}, \hat{\psi})$, there exists $\hat{\mathcal{H}}$ and $\hat{\mathcal{I}}$ according to Definition 10.4.4. By the definition of compatibility, $Q \subseteq Q_1 \cup Q_2 \cup \hat{Q}$ and (F, ψ) can be constructed as in Definition 10.4.5. We modify $\mathcal{H}$ and $\mathcal{I}$ along the line of this construction. In each of the following steps, the graph F is defined in the corresponding step in Definition 10.4.5.

1. $\mathcal{H}$ is initialized as $\mathcal{H}_1 \cup \mathcal{H}_2 \cup \hat{\mathcal{H}}$, and $\mathcal{I}$ is initialized as $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \hat{\mathcal{I}}$.

2. For every vertex $u \in Q_1 \cap Q_2$, let H' be the concatenation of $H_{\psi_1(u)}$ and $H_{\psi_2(u)}$ at vertex u. For every edge uv in $R_E \cap H'$, u and v are either two-edge-connected in $H_{\psi_1(u)}$, or two-edge-connected in $H_{\psi_2(u)}$. Therefore, u and v are two-edge-connected in H'. We update $\mathcal{H}$ by $\mathcal{H} \cup \{H'\} \setminus \{H_{\psi_1(u)}, H_{\psi_2(u)}\}$. Idem for every $u \in Q_1 \cap \hat{Q}$ and every $u \in Q_2 \cap \hat{Q}$.

3. For every two-edge-connected component A in F, consider the subgraphs H_a for all nodes a in A and the subgraphs I_{ab} for all edges ab in A, and let H' be the union of these subgraphs. We show that for every edge uv in $R_E \cap H'$, u and v are two-edge-connected in H'. When uv belongs to a subgraph H_a for some node a in A, u and v are two-edge-connected in H_a, which is a subgraph of H'. Next, consider the case when uv belongs to a path I_{ab} for some edge ab in A. Since A is two-edge-connected, there exists cycle in A that contains the edge ab, let it be $a_1, \ldots, a_k$. Define $a_{k+1} := a_1$ and $a_{k+2} := a_2$. We construct a cycle C in H' that contains the edge uv: for every $i \in [1, k]$, there is a path

[9] A node a in F is internal if $a \notin \psi(Q)$.

$I_{a_i a_{i+1}}$ connecting H_{a_i} and $H_{a_{i+1}}$. For every $i \in [2, k+1]$, let x_i be the end vertex of $I_{a_{i-1}a_i}$ and let y_i be the start vertex of $I_{a_i a_{i+1}}$. Both x_i and y_i are in H_{a_i}. Since H_{a_i} is connected, there is a path P_i in H_{a_i} connecting x_i and y_i. Let C be the concatenation of $I_{a_1 a_2}, P_2, I_{a_2 a_3}, P_3, \cdots, I_{a_k, a_{k+1}}, P_{k+1}$. Then C is a cycle (not necessary simple) in H' that contains uv. Note that uv appears in none of $\{H_a\}_{a \in A}$ and in exactly one of $\{I_{ab}\}_{ab \in A}$, and that every I_{ab} is a single path. Thus uv appears exactly once on C. By removing unnecessary edges on C, we obtain a simple cycle in H' that contains uv. So u and v are two-edge-connected in H'. Update $\mathcal{H}$ by $\mathcal{H} \cup \{H'\} \setminus \{H_a\}_{a \in A}$ and update $\mathcal{I}$ by $\mathcal{I} \setminus \{I_{ab}\}_{ab \in A}$.

4. For every internal node of F that is a singleton, we remove H_a from $\mathcal{H}$. For every internal node of F of degree 2, let b and c be its neighbors. Let u be the end vertex of the path I_{ba} and let v be the start vertex of the path I_{ac}. Both u and v are in H_a. Since H_a is connected, there is a path P in H_a connecting u and v. Let $I_{bc} = I_{ba} \circ P \circ I_{ac}$. We remove unnecessary edges from I_{bc} to make it a simple path. Then we remove H_a from $\mathcal{H}$, and update $\mathcal{I}$ by $\mathcal{I} \cup \{I_{bc}\} \setminus \{I_{ba}, I_{ac}\}$.

We prove that Γ is consistent with (E, Q, F, ψ) using $\mathcal{H}$ and $\mathcal{I}$ constructed above. The non-trivial part is to show that for every edge $uv \in R_E$ such that u and v are not two-edge-connected in $\Gamma \cup R_E$, there is exactly one edge ab from F such that $uv \in I_{ab}$. Assume without loss of generality that $uv \in R_{E_1}$. Then there is some edge e in F_1 such that $uv \in I_e$. Since I_e belongs to $\mathcal{I}_1$, it belongs to $\mathcal{I}$ in Step 1. I_e cannot be merged into some H' in Step 3, since otherwise u and v are two-edge-connected in $H' \subseteq \Gamma \cup R_E$. Therefore, after the construction, either I_e remains the same in $\mathcal{I}$, or I_e is a subsegment of some $I_{e'} \in \mathcal{I}$, which is the concatenation of I_e and other segments in Step 4. The statement holds in both case. □

10.4.3 Implementation

Preprocessing. First, the algorithm fills in the side table T during the preprocessing. Notice that any minimal $\hat{\Gamma} \subseteq E[H]$ that is consistent with $(\hat{Q}, \hat{F}, \hat{\psi})$ contains no cycles. Therefore, every node a in $\hat{F}$ corresponds to a vertex u_a in the graph H, and every edge ab in $\hat{F}$ corresponds to a path between u_a and u_b in $\hat{\Gamma}$. To compute the value $T(\hat{Q}, \hat{F}, \hat{\psi})$, the algorithm enumerates, for every $a \in \hat{F}$, the vertex u_a among $V[H]$. Next, for every $ab \in \hat{F}$, it computes a minimum-weight path between u_a and u_b in $H \setminus R$. The union of all these paths defines the current $\hat{\Gamma}$. The value $T(\hat{Q}, \hat{F}, \hat{\psi})$ is the minimum weight of all $\hat{\Gamma}$'s during the enumeration. The overall running time of the preprocessing is thus polynomial.

Base Case in DP. First, consider a cluster $E = \{uv\}$. Then (Q^*, F^*, ψ^*) must be one of the two configurations:

$$\begin{cases} (Q^A, F^A, \psi^A), & \text{if edge } uv \text{ belongs to exactly one of } S \text{ and } R, \\ (Q^B, F^B, \psi^B), & \text{otherwise.} \end{cases}$$

Here $Q^A = \{u, v\}$; F^A is a forest containing two nodes a and b and an edge ab; ψ^A maps u to a and v to b; $Q^B = \emptyset$; $F^B = \emptyset$; and $\psi^B = \emptyset^\emptyset$.

Remark. *When an edge uv belongs to both S and R, we have $S \setminus \{uv\}$ is a two-edge-connected augmentation for $(H, R \setminus \{uv\})$, since S is minimal. Therefore, we identify this case (and its DP state) with the case that uv belongs to neither S nor R (and its DP state).*

If $uv \in R$, we set $DP(E, Q^A, F^A, \psi^A) = 0$ and set $DP(E, Q^B, F^B, \psi^B)$ to be the minimum weight of a u-to-v path in $H \setminus R$.

If $uv \notin R$, we set $DP(E, Q^B, F^B, \psi^B) = 0$ and set $DP(E, Q^A, F^A, \psi^A)$ to be the minimum weight of a u-to-v path in $H \setminus R$.

By inspection, both properties of the DP value are satisfied.

Recursive Case in DP. Next, the algorithm fills in the DP table in the order of the index E from bottom up in $\mathcal{SC}$. Consider a cluster $E = E_1 \cup E_2$, where $E_1, E_2 \in \mathcal{SC}$. To compute the value at a DP index (E, Q, F, ψ), the algorithm enumerates every combination of (Q_1, F_1, ψ_1), (Q_2, F_2, ψ_2), and $(\hat{Q}, \hat{F}, \hat{\psi})$ that are compatible with (Q, F, ψ), and let $DP(E, Q, F, \psi)$ be

$$\min \left\{ DP(E_1, Q_1, F_1, \psi_1) + DP(E_2, Q_2, F_2, \psi_2) + T(\hat{Q}, \hat{F}, \hat{\psi}) \right\}.$$

From Lemma 10.4.7 and Fact 10.4.6, both properties of the DP value follow by induction.

10.5 Putting Them Together

In this section, we prove the Augmentation Theorem that we recall:

Theorem 10.1.1 (Augmentation Theorem)**.** *Let G be a plane graph with edge-weights. Let R be a subset of $E[G]$. Let T be a connected subgraph of G that contains every edge of R. Let $\epsilon > 0$ be a parameter. The algorithm* AUGMENT-CONNECTED(G, R, T, ϵ) *(Algorithm 10.2) computes in polynomial time a two-edge-connected augmentation S for (G, R) such that*

$$\text{weight}(S) \leq (1 + \epsilon) OPT(G, R) + \epsilon^2 \cdot \text{weight}(T).$$

By the Boundary Doubling Lemma (Lemma 9.4.1), the two-edge-connected augmentation problems for (G, R) and for (H, R) are equivalent. Thus we only need to prove the statement for the instance (H, R).

Let C be a connected component of $M^* \setminus \mathcal{E}_{j^*}$. Let H_C be the subgraph of H consisting of the bricks corresponding to $V[C]$. Let M_C be the mortar graph of H_C, i.e., it consists of the boundaries of the bricks corresponding to $V[C]$. (M_C is called a *parcel* in the terminology of [22].) From Lemma 8.9 in [22], the dual of M_C has a spanning tree of depth at most $\eta + 1 = O(1/\epsilon^3)$. Therefore, we can apply the Dynamic-Programming Theorem (Theorem 10.4.1) and obtain in polynomial time a two-edge-connected augmentation S_C for (H_C, R_C) such that:

$$\text{weight}(S_C) \leq (1+\epsilon)OPT(H_C, R_C) + 4 \sum_{\text{brick } B \text{ in } H_C} \text{weight}(\text{East}_B \cup \text{West}_B) \,.$$

Summing the above inequality over C, we have the weight of the output

$$\text{weight}\left(\bigcup_C S_C\right) \leq (1+\epsilon) \sum_C OPT(H_C, R_C) + 4 \sum_{\text{brick } B \text{ in } H} \text{weight}(\text{East}_B \cup \text{West}_B) \,.$$

By the Mortar-Graph Lemma (Lemma 9.2.4), $\text{weight}(M) = O(\text{weight}(T)\,/\epsilon)$, therefore

$$\text{weight}(\mathcal{E}_{j^*}) \leq (1/\eta)\text{weight}(M) = O(\epsilon^2 \cdot \text{weight}(T)).$$

So we have

$$\sum_C OPT(H_C, R_C) \leq OPT(H, R) + O(\epsilon^2 \cdot \text{weight}(T)).$$

Again by the Mortar-Graph Lemma, we have

$$\sum_{\text{brick } B \text{ in } H} \text{weight}(\text{East}_B \cup \text{West}_B) = O(\epsilon^2 \cdot \text{weight}(T)).$$

Therefore, the weight of the output is at most

$$(1+\epsilon)\,OPT(H, R) + O(\epsilon^2 \cdot \text{weight}(T)).$$

The statement follows by replacing ϵ by $\epsilon' = \epsilon/K$ for some absolute constant K that is large enough.

Chapter 11

Conclusion

Main Results. For planar graphs, we have provided a reduction from correlation clustering to two-edge-connected augmentation, mainly based on planar duality. Next, we have designed a polynomial-time approximation scheme for the latter problem. The scheme is based on the *brick decomposition* from [22]. In order to design a dynamic program to compute a near-optimal solution, we have proved a new structure property on bricks that we recall:

New Structure Property. *There exists a near-optimal solution such that, for any brick and any two vertices u, v on the boundary of the brick, there is a u-to-v Jordan curve inside the brick that crosses the near-optimal solution only a bounded number of times.*[1]

We hope that the new structure property can be used to give approximation schemes for other problems.

An Open Problem. Recall the *two-edge-connected Steiner subgraph* problem mentioned in Section 7.2.2. In this problem, we are given a subset $Q \subseteq V$ of terminals, and we want to find a minimum-weight subgraph such that all terminals from Q are two-edge-connected in this subgraph. For the special case when Q is the set of all vertices, this becomes the *two-edge-connected spanning subgraph* problem, which, in planar graphs, is NP-hard [39] and admits a PTAS [17]. Berger and Grigni [17] raised the question of whether there is a PTAS for the Steiner version of the problem in planar graphs. Borradaile and Klein [21] solved the relaxed version when the solution is allowed to contain multiple copies of each edge:

[1] The bound depends on ϵ.

> [21] We answer that question in the affirmative for the relaxed version. The question in the case of the strict version is still open.

Let us focus on the strict version of the problem, i.e., when every edge can be included at most once in the solution. We try to solve this problem using the new structure property. The obstacle is that, if we only specify the set of crossing vertices of the Jordan curve with the near-optimal solution as an DP index (as was the case in Section 10.4.1), there is no way to ensure that the solutions on different sides of the Jordan curve do not share edges. This does not harm in the two-edge-connected augmentation problem, because the solution is allowed to contain several two-edge-connected components. However, in this problem, the solution should be a single two-edge-connected component. So sharing edges is not allowed on different sides of a Jordan curve. Therefore, for every u and v on the boundary of the same brick, we need to encode the u-to-v Jordan curve more precisely. Of course, taking the complete Jordan curve requires too much memory. Hence the open question: Is there a concise encoding of the u-to-v Jordan curve inside the brick so that the solutions on different sides of the curve are disjoint?

Other Open Problems. There are many problems that we do not know whether they have approximation schemes in planar graphs, such as *facility location*, *vehicle routing*, *vertex-weighted Steiner tree*, and *directed Steiner tree*. It would be interesting to try to design approximation schemes for these problems by developing new structure properties on bricks, as has been done in our work.

List of Publications

[A] Sampath Kannan, Claire Mathieu, and Hang Zhou. Near-linear query complexity for graph inference. In *proceedings of the International Colloquium on Automata, Languages and Programming (ICALP),* 2015.

[B] Philip Klein, Claire Mathieu, and Hang Zhou. Correlation clustering and two-edge-connected augmentation for planar graphs. In *proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS),* 2015.

[C] Marc Lelarge and Hang Zhou. Sublinear-time algorithms for monomer-dimer systems on bounded degree graphs. In *Theoretical Computer Science (TCS), 548:68-78*, 2014. A preliminary version appears in *proceedings of the International Symposium on Algorithms and Computation (ISAAC)*, 2013.

[D] Claire Mathieu and Hang Zhou. Graph reconstruction via distance oracles. In *proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2013.

[E] Fabrice Benhamouda, Tancrède Lepoint, and Hang Zhou. Optimization of Bootstrapping in Circuits. Manuscript in preparation.

Bibliography

[1] Dimitris Achlioptas, Aaron Clauset, David Kempe, and Cristopher Moore. On the bias of traceroute sampling: or, power-law degree distributions in regular graphs. *Journal of the ACM (JACM)*, 56(4):21, 2009.

[2] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5), 2008.

[3] Sharon Alpert, Meirav Galun, Ronen Basri, and Achi Brandt. Image segmentation by probabilistic bottom-up aggregation and cue integration. In *Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[4] Amir Alush and Jacob Goldberger. Ensemble segmentation using efficient integer linear programming. *Pattern Analysis and Machine Intelligence*, 34(10):1966–1977, 2012.

[5] Amir Alush and Jacob Goldberger. Break and conquer: Efficient correlation clustering for image segmentation. In *Similarity-Based Pattern Recognition*, volume 7953, pages 134–147. Springer, 2013.

[6] Bjoern Andres, Jörg H. Kappes, Thorsten Beier, Ullrich Kothe, and Fred A. Hamprecht. Probabilistic image segmentation with closedness constraints. In *International Conference on Computer Vision*, pages 2611–2618. IEEE, 2011.

[7] Dana Angluin and Jiang Chen. Learning a hidden graph using O(log n) queries per edge. In *Learning Theory*, pages 210–223. Springer, 2004.

[8] Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Improved routing strategies with succinct tables. *Journal of Algorithms*, 11(3):307–341, 1990.

[9] Baruch Awerbuch and David Peleg. Routing with polynomial communication-space trade-off. *SIAM Journal on Discrete Mathematics*, 5(2):151–162, 1992.

[10] Yoram Bachrach, Pushmeet Kohli, Vladimir Kolmogorov, and Morteza Zadimoghaddam. Optimal coalition structure generation in cooperative graph games. In *Conference on Artificial Intelligence*, 2013.

[11] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[12] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Philip N. Klein, and Claire Mathieu. A polynomial-time approximation scheme for planar multiway cut. In *Symposium on Discrete Algorithms*, pages 639–655. SIAM, 2012.

[13] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *Journal of the ACM*, 58(5):21, 2011.

[14] Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matús Mihal'ak, and L. Shankar Ram. Network discovery and verification. *IEEE Journal on Selected Areas in Communications*, 24(12):2168–2181, 2006.

[15] Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999.

[16] Fabrice Benhamouda. Personal communication, 2014.

[17] A. Berger and M. Grigni. Minimum weight 2-edge-connected spanning subgraphs in planar graphs. In *International Colloquium on Automata, Languages and Programming*, volume 4596, pages 90–101, 2007.

[18] Jean R. S. Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.

[19] Sebastian Böcker and Jan Baumbach. Cluster editing. In *The Nature of Computation. Logic, Algorithms, Applications*, volume 7921, pages 33–44. Springer, 2013.

[20] Glencora Borradaile, Erik D Demaine, and Siamak Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014.

[21] Glencora Borradaile and Philip N. Klein. The two-edge connectivity survivable network problem in planar graphs. *International Colloquium on Automata, Languages and Programming*, pages 485–501, 2008.

[22] Glencora Borradaile, Philip N. Klein, and Claire Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3):31, 2009.

[23] Mathilde Bouvel, Vladimir Grebinski, and Gregory Kucherov. Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. In *Graph-Theoretic Concepts in Computer Science*, pages 16–27. Springer, 2005.

[24] Gerth Stølting Brodal, Rolf Fagerberg, Christian N.S. Pedersen, and Anna Östlin. The complexity of constructing evolutionary trees using experiments. In *International Colloquium on Automata, Languages and Programming*, volume 28, page 140. Springer, 2001.

[25] Rui Castro, Mark Coates, Gang Liang, Robert Nowak, and Bin Yu. Network tomography: recent developments. *Statistical Science*, 19:499–517, 2004.

[26] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.

[27] Gary Chartrand and Frank Harary. Planar permutation graphs. *Annales de l'institut Henri Poincaré (B) Probabilités et Statistiques*, 3(4):433–438, 1967.

[28] Daniel Chen, Leonidas J. Guibas, John Hershberger, and Jian Sun. Road network reconstruction for organizing paths. In *SODA*, pages 1309–1320, 2010.

[29] Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. In *STOC*, pages 749–758. ACM, 2008.

[30] F. Chung, M. Garrett, R. Graham, and D. Shallcross. Distance realization problems with applications to internet tomography. *Journal of Computer and System Sciences*, 63:432–448, 2001.

[31] William W. Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 475–480. ACM, 2002.

[32] Lenore J. Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001.

[33] Artur Czumaj and Andrzej Lingas. On approximability of the minimum-cost k-connected spanning subgraph problem. In *Symposium on Discrete Algorithms*, pages 281–290. SIAM, 1999.

[34] Luca Dall'Asta, Ignacio Alvarez-Hamelin, Alain Barrat, Alexei Vázquez, and Alessandro Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. *Theoretical Computer Science*, 355(1):6–24, 2006.

[35] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2):172–187, 2006.

[36] Reinhard Diestel. *Graph Theory*. Electronic library of mathematics. Springer, 2006.

[37] Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.

[38] Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. *Journal of Algorithms*, 46(2):97–114, 2003.

[39] Kapali P. Eswaran and R. Endre Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.

[40] Pierre Fraigniaud and Cyril Gavoille. Memory requirement for universal routing schemes. In *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, pages 223–230. ACM, 1995.

[41] Greg N. Frederickson and Joseph Ja'Ja'. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

[42] Anna Galluccio and Guido Proietti. A faster approximation algorithm for 2-edge-connectivity augmentation. In *International Symposium on Algorithms and Computation*, pages 150–162, 2002.

[43] Cyril Gavoille and Marc Gengler. Space-efficiency for routing schemes of stretch factor three. *Journal of Parallel and Distributed Computing*, 61(5):679 – 687, 2001.

[44] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007.

[45] Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. In *Theory of Computing*, pages 1167–1176. ACM, 2006.

[46] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, É. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Symposium on Discrete Algorithms*, SODA '94, pages 223–232. SIAM, 1994.

[47] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[48] Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000.

[49] Jotun J Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of Mathematical Biology*, 51(5):597–603, 1989.

[50] Shinichi Honiden, Michael E. Houle, and Christian Sommer. Balancing graph voronoi diagrams. In *ISVD*, pages 183–191. IEEE, 2009.

[51] Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[52] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

[53] Raja Jothi, Balaji Raghavachari, and Subramanian Varadarajan. A 5/4-approximation algorithm for minimum 2-edge-connectivity. In *Symposium on Discrete Algorithms*, pages 725–734. SIAM, 2003.

[54] Sampath K. Kannan, Eugene L. Lawler, and Tandy J. Warnow. Determining the evolutionary tree using experiments. *Journal of Algorithms*, 21(1):26 – 50, 1996.

[55] Ming-Yang Kao, Andrzej Lingas, and Anna Östlin. Balanced randomized tree splitting with applications to evolutionary tree constructions. In *STACS 99*, pages 184–196. Springer, 1999.

[56] Samir Khuller and Ramakrishna Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993.

[57] Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214–235, 1994.

[58] Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang Dong Yoo. Higher-order correlation clustering for image segmentation. In *Advances in Neural Information Processing Systems*, pages 1530–1538, 2011.

[59] Valerie King, Li Zhang, and Yunhong Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *SODA*, pages 444–453. SIAM, 2003.

[60] Philip N. Klein. A subset spanner for planar graphs: with application to subset TSP. In *Symposium on Theory of Computing*, pages 749–756. ACM, 2006.

[61] Philip N. Klein and Shay Mozes. Optimization algorithms for planar graphs. In preparation, manuscript at http://planarity.org.

[62] Philip N. Klein and R. Ravi. When cycles collapse: A general approximation technique for constrained two-connectivity problems. In *Integer Programming and Combinatorial Optimization*, pages 39–55, 1993.

[63] Mathias Bæk Tejs Knudsen. Personal communication, 2014.

[64] Guy Kortsarz, Robert Krauthgamer, and James R. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM Journal on Computing*, 33(3):704–720, 2004.

[65] Guy Kortsarz and Zeev Nutov. Approximating minimum cost connectivity problems. *Approximation Algorithms and Metahueristics*, 2007.

[66] David R. Martin, Charless C. Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *Pattern Analysis and Machine Intelligence*, 26(5):530–549, 2004.

[67] Claire Mathieu and Warren Schudy. Correlation clustering with noisy input. In *Symposium on Discrete Algorithms*, pages 712–728, 2010.

[68] Hanna Mazzawi. Optimally reconstructing weighted graphs using queries. In *SODA*, pages 608–615. SIAM, 2010.

[69] Brendan D McKay and Nicholas C Wormald. Asymptotic enumeration by degree sequence of graphs with degrees $o(n^{1/2})$. *Combinatorica*, 11(4):369–382, 1991.

[70] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM (JACM)*, 36(3):510–530, 1989.

[71] Bruce A. Reed. Algorithmic aspects of tree width. In *Recent advances in algorithms and combinatorics*, pages 85–107. Springer, 2003.

[72] Mauricio Resende and Panos Pardalos. *Handbook of optimization in telecommunications*. Springer, 2008.

[73] Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Algorithmic Learning Theory*, pages 285–297. Springer, 2007.

[74] Lev Reyzin and Nikhil Srivastava. On the longest path algorithm for reconstructing trees from distance matrices. *Information processing letters*, 101(3):98–100, 2007.

[75] Chaitanya Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *Symposium on Discrete Algorithms*, pages 526–527. SIAM, 2004.

[76] Fabien Tarissan, Matthieu Latapy, and Christophe Prieur. Efficient measurement of complex networks using link queries. In *INFOCOM Workshops*, pages 254–259. IEEE, 2009.

[77] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Symposium on Parallel Algorithms and Architectures*, pages 1–10. ACM, 2001.

[78] M.S. Waterman, T.F. Smith, M. Singh, and W.A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64(2):199 – 213, 1977.

[79] David P. Williamson, Michel X. Goemans, Milena Mihail, and Vijay V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15(3):435–454, 1995.

[80] Julian Yarkony, Alexander Ihler, and Charless C Fowlkes. Fast planar correlation clustering for image segmentation. In *European Conference on Computer Vision*, volume 7577, pages 568–581. Springer, 2012.