
By restricting the input to a problem, it often becomes possible to design more accurate or more efficient algorithms to solve that problem. In this thesis we restrict our attention to planar graphs and achieve both these goals. Planar graphs exhibit many structural and combinatorial properties that enable the design of good algorithms. These properties include: corresponding to every planar graph there is a dual planar graph; the dual of the complement of the edges of a spanning tree form a spanning tree of the dual graph; a set of edges is a cycle if and only if the dual edges form a cut; cycles can be said to enclose edges, faces and vertices in the planar embedding; paths can be compared as to their relative embedding.

We capitalize on these properties to design (a) faster algorithms for polynomial-time solvable network flow problems and (b) algorithms with better approximation guarantees for NP-hard connectivity problems. We give a conceptually simple $O(n \log n)$-time algorithm for finding the maximum st-flow in a directed planar graph, proving a theorem that was incorrectly claimed over a decade ago. We also show how to compute the minimum cut between all pairs of vertices on a common face of a planar graph in linear time. We give the first polynomial-time approximation schemes for the Steiner-tree and 2-edge-connected subgraph problems. Both schemes are NP-hard in planar graphs and admit no PTAS in general graphs. Our schemes run in $O(n \log n)$ time.
Exploiting Planarity for Network Flow and Connectivity Problems

by

Glencora Borradaile

Honors B.Sc. Applied Mathematics, University of Western Ontario, 2002
Sc.M. Computer Science, Brown University, 2004

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy in the
Department of Computer Science at Brown University

Providence, Rhode Island
May 2008
This dissertation by Glencora Borradaile is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date ________________  
Philip Klein, Director

Recommended to the Graduate Council

Date ________________  
Claire Mathieu, Reader

Date ________________  
Robert Tarjan, Reader  
(Princeton University)

Approved by the Graduate Council

Date ________________  
Sheila Bonde  
Dean of the Graduate School
Biography

Glencora Borradaile was born during a snowstorm in Thunder Bay, Ontario on the twelfth of December, 1980. She is the daughter of a teacher and a professor and the sister of a biochemist and an economist. She attended the University of Western Ontario from 1998 to 2002, earning an Honours Bachelor of Science degree in Applied Mathematics and was awarded a Gold Medal as the top graduate of the program. She completed her Masters in 2004 while studying towards a doctorate in Computer Science at Brown University. She has received two undergraduate research awards, two graduate scholarships and a postdoctoral fellowship from the Natural Science and Engineering Research Council of Canada. While at Brown University, she has received a Kanellakis Graduate Fellowship, a Brown University Dissertation Fellowship and a Rowland Lloyd Graduate Award.

When she isn’t at her white board, she’s being either athletic or culinary.
Acknowledgments

Philip Klein helped me rediscover my love for research and I look forward to his advice in the years to come as I grow to become his colleague. Claire Mathieu has been more than a committee member; she has been a role-model. I thank Robert Tarjan, whose research is woven throughout this thesis, for serving on my committee.

I thank the theory-lunch crowd at Brown and the theory community at large for always providing a welcoming atmosphere. Special thanks go to Aparna Das, Bernard Haeupler, Nick Harvey, Crystal Kahn, Warren Schudy and Christopher Wilson who commented on parts of this dissertation.

I am grateful for the financial support of the Natural Science and Engineering Research Council of Canada, the Rosh Foundation, the National Science Foundation, the Canadian Scholarship Trust and General and Mrs. Kanellakis.

To my family, who supported me.
To my friends, who kept me sane.
Contents

List of Tables viii

List of Figures ix

1 Introduction 1
  1.1 Contributions and organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
  1.2 Preliminaries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
    1.2.1 Graphs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
    1.2.2 Vector spaces . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
    1.2.3 Planar graphs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
    1.2.4 Duals of planar graphs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
    1.2.5 Clockwise and leftmost . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
    1.2.6 Tree-cut operation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2 Network Flow Problems 11
  2.1 Maximum flow algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
  2.2 History of planar maximum flow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.2.1 Toward an $O(n \log n)$ algorithm . . . . . . . . . . . . . . . . . . . . . . . . . 14
  2.3 Leftmost-path algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
    2.3.1 Leftmost circulations and flows . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
    2.3.2 Unusability Theorem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
    2.3.3 Implementation of MAXFLOW . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
  2.4 Multiple minimum cuts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
    2.4.1 Minimum cuts between boundary vertices of a planar graph . . . . . . . . . . . . 31
    2.4.2 Running time of BOUNDARYCUTTREE . . . . . . . . . . . . . . . . . . . . . . . . 31
    2.4.3 Correctness of BOUNDARYCUTTREE . . . . . . . . . . . . . . . . . . . . . . . . . 31
    2.4.4 A more compact cut tree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
  2.5 Open Problems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
3 Connectivity Problems

3.1 Algorithms for connectivity problems .......................... 38
  3.1.1 Polynomial-time approximation schemes .................... 38
  3.1.2 Polynomial-time solvable cases .............................. 39
  3.1.3 New approximation schemes for planar connectivity problems .... 40

3.2 Brick decomposition .............................................. 40
  3.2.1 Construction of the mortar graph ........................... 41
  3.2.2 Running time of BRICKDECOMPOSITION ...................... 42
  3.2.3 Properties of the mortar graph .............................. 42

3.3 Structural properties of bricks ................................ 43
  3.3.1 Bricks ....................................................... 44
  3.3.2 Simplifying trees with leaves on ε-short paths .............. 46
  3.3.3 Simplifying forests inside bricks (Proof of Theorem 3.3.3) .... 52
  3.3.4 Simplifying subgraphs of 2-EC multi-subgraphs inside bricks .... 56

3.4 The Structure Theorem .......................................... 63
  3.4.1 Portals ..................................................... 63
  3.4.2 Portal-connected graph .................................... 64

3.5 Approximation via spanner construction ........................ 67
  3.5.1 Spanner Construction ...................................... 68
  3.5.2 Correctness ............................................... 69

3.6 Approximation via the brick decomposition ..................... 70
  3.6.1 Parcels ..................................................... 70
  3.6.2 New requirements .......................................... 73

3.7 Dynamic program .................................................. 75
  3.7.1 Defining the recursion tree ............................... 76
  3.7.2 The dynamic programming table ............................ 77
  3.7.3 Steiner tree ............................................... 78
  3.7.4 2-EC ....................................................... 80
  3.7.5 Correctness ............................................... 82

3.8 An exact algorithm for the boundary 2-EC problem ............. 83

3.9 Open Problems .................................................... 85

A Notation .......................................................... 88

Bibliography .......................................................... 89

* Parts of this thesis have appeared in print before [15, 16, 19]. I thank my coauthors, Philip Klein and Claire Mathieu, for their permission in using portions of these papers in this thesis.
## List of Tables

1.1 **TreeCut Algorithm** ........................................ 10

2.1 Planar Maximum-Flow and Minimum-Cut Algorithms .................. 14
2.2 **LeftmostCirculation Algorithm** .......................... 16
2.3 Abstract **MAXFLOW** Algorithm .............................. 17
2.4 Implemented **MAXFLOW** Algorithm .......................... 26
2.5 **BOUNDARYCutTREE** Algorithm .............................. 31
2.6 **GomoryHuEDGES** Algorithm ................................. 34

3.1 **BRICKDECOMPOSITION** Algorithm .......................... 42
3.2 **DECOMPOSEConnectivity** Algorithm ........................ 59
3.3 **PORTALSelection** Algorithm ............................... 64
3.4 A framework for designing planar PTAS .......................... 68
3.5 **CONNECTIVITYSpanner** Algorithm .......................... 69
3.6 **PARCELDECOMPOSITION** Algorithm ....................... 71
3.7 **NEWRequirements** Algorithm .............................. 74
3.8 The filling procedure for Steiner tree and 2-EC dynamic programs 78
3.9 **BOUNDARY2EC** Algorithm ................................. 83
List of Figures

1.1 A planar graph and its dual .............................................. 5
1.2 Interdigitating spanning trees .......................................... 6
1.3 Cycle-cut duality .............................................................. 6
1.4 Cycle orientation ............................................................... 7
1.5 Crossing paths and cycles .................................................... 8
1.6 A non-self-crossing cycle have faces bounded by subpaths of the cycle .... 10
1.7 Cutting open a planar graph along a tree ............................... 10

2.1 Flow can be removed from an arc in the leftmost-path algorithm .............. 18
2.2 An obstruction for proving the Unusability Theorem .......................... 19
2.3 Showing that there is no flow across an obstruction .......................... 21
2.4 Flow paths to and from an obstruction do not intersect ....................... 22
2.5 Using an obstruction to prove the Unusability Theorem ....................... 24
2.6 The creation of an obstruction .............................................. 24
2.7 Persistence of obstructions in proving the Unusability Theorem ............... 25
2.8 An iteration of MaxFlow ...................................................... 27
2.9 Termination of MaxFlow ...................................................... 28
2.10 A Gomory-Hu cut-equivalent tree ........................................ 30
2.11 A fundamental cut ............................................................ 33
2.12 A compact tree representing boundary-to-boundary minimum cuts ........... 35

3.1 The mortar graph and bricks ................................................ 41
3.2 Strips and columns of the brick decomposition ................................ 41
3.3 The set of bricks corresponding to a mortar graph ............................ 44
3.4 Construction of a brick ......................................................... 45
3.5 Tree replacements reduce the number of leaves ................................ 47
3.6 Three paths in a tree for payment of a tree replacement ...................... 47
3.7 Edge-sets of a tree needed for the analysis of Lemma 3.3.6 ..................... 48
3.8 Two paths in a tree for payment of a tree replacement ........................ 49
3.9 A tree with two roots .......................................................... 50
3.10 Tree transformation for Lemma 3.3.7 ........................................ 51
3.11 South-to-north paths in forest embedded in a brick ........................................... 54
3.12 Breaking a graph into trees with leaves on a single path ........................................ 55
3.13 Cycles block 2-EC edges .............................................................. 57
3.14 Crossing edge-disjoint paths imply more edge-disjoint paths ................................. 58
3.15 Decomposing a 2-EC graph into trees ............................................................. 61
3.16 Maintaining 2 edge-disjoint paths between boundary vertices ................................. 63
3.17 Construction of the portal connected graph ......................................................... 64
3.18 The brick insertion operation ................................................................ 65
3.19 Breadth-first search levels correspond to cycles ....................................................... 75
3.20 Brick contraction ..................................................................................... 76
3.21 A configuration for the Steiner-tree dynamic program ............................................... 79
3.22 A configuration for the 2-EC dynamic program ...................................................... 80
3.23 Existence of a tree enclosed by a cycle ................................................................. 84
3.24 Duplicate edges are required .......................................................................... 85
3.25 The Steiner forest problem .............................................................................. 86
Chapter 1

Introduction

Claire Mathieu once said that planar graphs are treacherous. Centuries earlier, Euler discovered the same thing. In 1750, Euler set out to prove what is now known as the Euler Characteristic for Polyhedra:

\[ n - m + f = 2 \]

where \( n \) is the number of vertices, \( m \) the number of edges, and \( f \) the number of faces of a polyhedron. Attempting to characterize the polyhedra for which it is true is daunting, and, indeed, forms the basis of Lakatos’ book on mathematical logic [76]. Euler, in the first of two papers on this matter, proposed the characteristic but then admits, “I have not been able to find a firm proof of this theorem.” [39, 94] In the second paper, Euler thought he proved the characteristic for all polyhedra, when in fact he only proved it for convex polyhedra [38, 95]. Unbeknownst to Euler, Descartes had written, a century earlier, the very proof that had eluded Euler. Descartes proved a rule that can easily be transformed into the Euler Characteristic [28] for the relevant polyhedra. Sadly, Descartes’ original manuscripts were lost, despite having been dropped into a river, rescued, dried and ultimately transcribed by Leibniz. The proof was unknown until the copy was rediscovered in 1860. Admittedly, it is much easier to prove the characteristic for connected planar graphs, where one need not worry about aberrant polyhedra.

Planar graphs had also been the focus of Euler’s 1741 Bridges-of-Königsberg paper, a paper that may have single-handedly launched the field of topology. The people of Königsberg wondered if they could walk through their town in such a way that each of the seven bridges is crossed exactly once. From the town’s map, Euler generated a (planar) graph with an edge for each bridge and proved that there exists a path on a graph which travels along each edge exactly once if and only if the graph is connected and has zero or two vertices of odd degree [37]. Of course, this property is also true for general graphs, but is perhaps the earliest motivation that studying planar graphs can lead to wonderful insights.
1.1 Contributions and organization

In this thesis we study ways in which planarity can be exploited to design algorithms that outperform their general-graph counterparts. The last 50 years has seen a wealth of research in this vein. For example, the existence of small, balanced separators [77] in planar graphs gives efficient divide-and-conquer algorithms for shortest paths [55, 40] (among other problems). Here we use other properties of planar graphs, such as the existence of a dual planar graph, the relative orientation of paths, and combinatorial embeddings to find better approximation algorithms than are possible in general graphs and to find more efficient exact algorithms than are known for general graphs.

The thesis is broken into two main parts: Chapter 2 concerns network flow problems and Chapter 3 concerns connectivity problem. Each of these chapters is self-contained and for that reason, we forgo a detailed introduction to the relevant chapter. All of our results are deterministic. We summarize our results here:

**Maximum directed st-flow** fifty years ago, Ford and Fulkerson gave an algorithm for finding the maximum st-flow in an st-planar graph [41]. We generalize this algorithm to handle any directed planar graph in $O(n \log n)$-time [16], proving a theorem incorrectly claimed by Weihe [108]. This improves on an $O(n \log^3 n \log C)$ running time using previously known techniques [40, 79] (where $C$ is the sum of the capacities). The best algorithm for general graphs requires $O(n^{3/2} \log^2 n)$ time for planar graphs [46]. Our algorithm is used as a black box for an algorithm that computes the global minimum cut in a planar graph [22]. It is also used a subroutine for finding a maximum st-flow in a non-planar graph that is embedded in the plane such that only $k$ pairs of edges are crossing [56].

**Multiple undirected minimum cuts** We show that, given a face $f$ of an undirected planar graph, one can compute the minimum cut between each pair of vertices adjacent to $f$ in linear time [14]. This improves on a quadratic running time using previously known techniques [48, 51, 55].

**Steiner tree** We give the first polynomial-time approximation scheme for the Steiner-tree problem in undirected planar graph [15, 19]. The algorithm runs in $O(n \log n)$ time. The previous best approximations are constant-factor approximations for general graphs. It has been shown that no PTAS exists for general graphs [11]. Our algorithm has been used as a black box to obtain an $O(n \log n)$-time approximation scheme for finding a Steiner tree in the Euclidean plane in the presence of obstacles [81].

**2 edge-connected multi-subgraphs** Given a set of connectivity requirements $r(x) \in \{0, 1, 2\}$ defined for each vertex $x$, the 2 edge-connected multi-subgraph problem is: find a minimum-weight multi-subgraph that contains at least $\min\{r(x), r(y)\}$ edge disjoint paths between every pair of vertices $x, y$. We give the first polynomial-time approximation scheme for this problem in undirected planar graph [17]. The algorithm is an extension of the previously mentioned Steiner-tree algorithm and has the same running time. Previous work only considered the case where the solution spans all
the vertices of the input graph (i.e. \( r(x) \in \{1, 2\} \)) [7, 6]. We also give an exact linear-time algorithm for the case when the vertices with non-zero requirements are on the boundary of a single face.

Throughout this document we state the date of the first publicly-available peer-reviewed publications but cite the most complete publications.

## 1.2 Preliminaries

We conclude the introduction with basic graph notation and properties of planar graphs that will be used throughout the paper. The reader familiar with these topics may feel free to skip the remainder of this chapter (Chapter 2 begins on page 11). However the comparator \textit{left of} and the operation \textsc{TreeCut} will be covered in this section and may not be familiar to most readers. The notation used in this paper is summarized in Appendix A (page 88).

### 1.2.1 Graphs

A graph \( G \) is either a directed graph \( \langle V, A \rangle \) or an undirected graph \( \langle V, E \rangle \), where \( A \) and \( E \) are the sets of \textit{arcs} and \textit{edges} respectively. For an arc \( a \in A \), we define two oppositely directed \textit{darts}, one in the same orientation as \( a \) (which we sometimes identify with \( a \)) and one in the opposite orientation. Similarly for an edge \( e \in E \), we define two oppositely directed \textit{darts}.

We define \( \text{rev} (\cdot) \) to be the function that takes each dart to the corresponding dart in the opposite direction. Formally, the dart set is \( A \times \{\pm 1\} \), and \( \text{rev} ((a, i)) = (a, -i) \). The head and tail of a dart \( d \) in a graph \( G \) (written \( \text{head}_G(d) \) and \( \text{tail}_G(d) \)) are such that the dart is oriented from tail to head. We may omit the subscript when doing so introduces no ambiguity. We may use \( uv \) to indicate a dart \( d \) such that \( u = \text{tail}(d) \) and \( v = \text{head}(d) \).

An \( x \)-to-\( y \) \textit{walk} is a sequence of darts \( d_1 \ldots d_k \) such that \( \text{head}_G(d_1) = x \), \( \text{tail}_G(d_k) = y \) and, for \( i = 2, \ldots, k \), \( \text{head}_G(d_{i-1}) = \text{tail}_G(d_i) \). We call a is walk in which no dart appears more than once a \textit{path}. We may interpret a dart as a path. An empty sequence represents the trivial path consisting of a single vertex; we will always associate a particular empty sequence with a particular vertex. We use \( \text{start}(P) \) to denote the first vertex, \( x \), of \( P \) and \( \text{end}(P) \) to denote the last vertex \( y \) of \( P \). If additionally \( \text{head}_G(d_k) = \text{tail}_G(d_1) \) then the walk is a cycle. A path/cycle of darts is \textit{simple} if no vertex occurs twice as the head of a dart in the path/cycle.

If \( P = d_1 \ldots d_k \) and \( Q = e_1 \ldots e_\ell \) are walks such that \( \text{end}(P) = \text{start}(Q) \), we use \( P \circ Q \) to denote the walk \( d_1 \ldots d_k e_1 \ldots e_\ell \). If \( u \) and \( v \) are vertices in path \( P \) such that \( u = \text{tail}(d_i) \), \( v = \text{head}(d_j) \) and \( i \leq j \) (or, \( u = v \) for a vertex \( u \) in \( P \)), we use \( P[u, v] \) to denote the subpath \( P' \) such that \( \text{start}(P') = u \) and \( \text{end}(P') = v \). Since walks may visit vertices or darts multiple times, when we use this notation, we intend \( u \) and \( v \) to refer to specific occurrences of vertices within the path. If \( P \) is a cycle, and \( u \) occurs before \( v \) in \( P \) then \( P[u, v] \) denotes a subpath of the cycle. We use \( P[u, v] \) to denote the walk obtained from the walk \( P[u, v] \) by deleting the last dart; \( P(u, v) \) and \( P'(u, v) \) are defined similarly.
$P[\cdot,v]$ denotes the subpath $P'$ with $start(P') = start(P)$ and $end(P') = v$. $P[u,\cdot]$ is similarly defined. The reverse of $P$, $rev(P)$ is defined as the sequence $rev(d_k), rev(d_{k-1}), \ldots, rev(d_1)$.

Graphs are identified with sets of edges, thus a subgraph $H$ of a graph $G$ is also considered a subset of the edges of $G$. The set of vertices that are endpoints of edges in $H$ is denoted $V(H)$.

A spanning tree $T$ of $G$ is a connected, spanning subgraph of $G$ that contains no cycles of edges or arcs. (Note that $T$ will contain cycles of darts, but we regard $T$ primarily as a set of edges or arcs, disregarding orientation.) For vertices $u$ and $v$, $T[u,v]$ denotes the unique simple $u$-to-$v$ path through $T$. Given an identified vertex or root of $T$, $T[u]$ denotes the $u$-to-root path in $T$.

We denote the length of the shortest $x$-to-$y$ path in $G$ as $dist_G(x,y)$.

For a set $S$ of vertices, $\Gamma^+(S)$ is the set of darts whose tails are in $S$ and whose heads are not. Such a set of darts is called a cut. A cut is a simple cut if both $S$ and $V \setminus S$ are connected. A simple cut is also known as a bond.

### 1.2.2 Vector spaces

The arc space of a graph $G = \langle V,A \rangle$ is the vector space $\mathbb{R}^A$: a vector $\delta$ in arc space assigns a real number $\delta[a]$ to each arc $a \in A$. It is notationally convenient to interpret a vector $\delta$ in arc space as assigning real numbers to all darts. For a dart $\langle a,i \rangle$ $(i = \pm 1)$, we define

$$\delta[\langle a,i \rangle] = i \cdot \delta[a].$$

For each arc $a$, we define $\delta(a)$ to be the vector in arc space that assigns 1 to $a$ and zero to all other arcs:

$$\forall a' \in A, \delta(a)[a'] = \begin{cases} 1 & \text{if } a' = a, \\ 0 & \text{otherwise.} \end{cases}$$

For a multi-set $S$ of darts, we define $\delta(S) = \sum_{d \in S} \delta(d)$. For any set of edges, arcs or darts, $H$, we define $\delta(H) = \delta(S)$ where $S$ is the multi-set of darts comprising $H$. We equate a vertex $v$ with the set of darts whose tails are $v$, so $\delta(v)$ is $\sum_{\text{tail}(d) = v} \delta(d)$.

A vector $\eta$ in arc space specifies a set of darts of $G$, namely the set of darts assigned positive values by $\eta$. We say, for example, that a dart $d$ is in $\eta$ if $\eta[d] > 0$. We can similarly say that $\eta$ contains a path or a cycle.

The cycle space of $G$ is the subspace of the arc space spanned by

$$C = \{ \delta(C) : C \text{ a cycle of darts in } G \}.$$ 

We will refer to vectors in cycle space alternatively as circulations. This term will be overloaded when we discuss flow problems where we will impose constraints on the values in the circulation vector.

Given a spanning tree $T$ and for a dart $d \notin T$, the fundamental cycle of $d$ with respect to $T$ is the cycle formed by $d$ and the path in $T[\text{head}_G(d),\text{tail}_G(d)]$. Fundamental cycles are simple. The set of all fundamental cycles forms a basis for the cycle space.
1.2.3 Planar graphs

According to the geometric definition, a planar graph is a graph for which there exists a planar embedding. A planar embedding of a graph is a drawing of the graph on the plane (or the surface of a sphere) so that vertices are mapped to distinct points and edges are mapped to non-crossing curves. We call a set of contiguous points in the plane/on the sphere that are not in the image of the vertices or arcs a face. For an embedding on the plane, there is one infinite face. For an embedding on the sphere, an arbitrary face can be designated as the infinite face. We denote the infinite face $f_\infty$.

One can alternatively define embeddings combinatorially, without reference to topology [53, 30, 113]. A combinatorial embedding is sometimes called a rotation system. A combinatorial embedding is given by a permutation $\pi$ such that for each dart $d$, $\pi(d)$ is the dart $e$ such that $x = tail(d) = tail(e)$ and $e$ is the dart immediately after $d$ in the counterclockwise ordering of the darts around $x$. While such a formulation frequently makes the implementation of algorithms simpler, we will only use the permutation $\pi$ explicitly in a few places throughout this work. However, we do note that for all the algorithms contained herein, a combinatorial embedding is sufficient for implementation.

Planar graphs can also be characterized by the minors they do not contain: the complete graph with 5 vertices, $K_5$, and the complete bipartite graph with 6 vertices, $K_{3,3}$ [74, 107]. We will not use characterization in this thesis.

1.2.4 Duals of planar graphs

Corresponding to every connected planar embedded graph $G$ there is another connected planar embedded graph denoted $G^*$. The faces of $G$ are the vertices of $G^*$ and vice versa. The arcs (and hence darts) of $G$ correspond one-to-one with those of $G^*$. If $d$ is a dart of $G$, the tail of the corresponding dart of $G^*$ is the face to the left of $d$, and the head is the face to the right of $d$. Thus intuitively the geometric orientation in $G^*$ of the dart corresponding to $d$ is obtained by rotating the embedding of $d$ clockwise roughly 90 degrees. It is notationally convenient to equate the darts of $G$ with the darts of $G^*$. We call $G$ the primal graph and $G^*$ the dual and an example is given in Figure 1.1. The permutation corresponding to the combinatorial embedding for $G^*$ is denoted $\pi^*$ and is equal to $\pi \circ rev$.

![Figure 1.1: A planar graph and its dual: the primal is given by solid vertices and solid arcs and the dual is given by open vertices and dotted arcs.](image)
We will liberally use the following two classical results on planar graphs. These theorems are illustrated in Figures 1.2 and 1.3, respectively.

**Theorem 1.2.1** (Interdigitating Spanning Trees [34, 100]). For a spanning tree $T$ of $G$, the set of arcs or edges not in $T$ form a spanning tree of the dual $G^*$ which we denote $T^*$.

![Figure 1.2: The primal is given by solid arcs and the dual by dotted arcs. The dark bold edges form a spanning tree $T$ of the primal. The edges not in $T$ form a spanning tree $T^*$ of the dual.](image)

**Theorem 1.2.2** (Cycle-Cut Duality [109]). In a connected planar graph, a set of darts forms a simple directed cycle in the primal iff it forms a simple directed cut in the dual.

![Figure 1.3: The primal is given by solid edges and the dual by dotted edges. The dark bold (directed) darts form a simple directed cycle in the dual and a directed bond, $\Gamma^+(S)$, in the primal, where $S$ is the set of the lower 4 vertices.](image)

Recall that the cycle space of $G$ is the subspace of the arc space spanned by $\mathcal{C} = \{\delta(C) : C$ a cycle of darts in $G\}$. We equate a face $f$ with the set of darts forming the counterclockwise boundary of the face, so $\delta(f)$ is the sum of $\delta(d)$ over such darts. In a planar graph the set of vectors

$$\{\delta(f) : f$ a face of $G, f \neq f_\infty\}$$

is a basis for the cycle space of $G$. Therefore any vector $\eta \in \mathcal{C}$ can be represented as a linear combination of these basis vectors. We use $\phi$ to denote the vector of coefficients for this linear combination, so

$$\eta = \sum_{f \neq f_\infty} \phi[f] \delta(f)$$
We call $\phi$ a potential assignment, and we refer to $\phi[f]$ as the potential of face $f$. This use of potentials was introduced by Hassin [51] for $st$-planar graphs (a graph in which $s$ and $t$ are on the boundary of a common face) and by Miller and Naor [79] for general planar graphs. We adopt the convention that $\phi[f_\infty] = 0$.

We say a face $f$ is external to the circulation corresponding to a potential assignment $\phi$ if $\phi[f] = 0$, and is internal otherwise. We say that a dart $d$ is external if the faces to $d$’s left and right are external, and we say $d$ is internal if the faces to $d$’s left and right are internal. A dart can be neither internal nor external. We say a dart is contained by a circulation if the dart is neither internal nor external. A dart and its reverse have the same property (external, internal, or contained) with respect to a circulation. We say a vertex $v$ is external if every dart incident to $v$ is external, and is internal if every dart incident to $v$ is internal.

For a cycle $C$, we say $C$ encloses a face (dart or vertex, resp.) if the face (dart or vertex, resp.) is internal to or contained by $\delta(C)$. Likewise, we define strictly enclosed if the dart or vertex is internal to $\delta(C)$. For $C$ a simple cycle with a geometric embedding, this definition corresponds to the notion of a face, dart, or vertex being embedded inside $C$, with respect to $f_\infty$.

The boundary of a face $f$ of a planar embedded graph is the set of darts whose tail is $f$ in the dual and is denoted $\partial f$. This set of darts is non necessarily connected. If the set is connected, then the darts form a non-self-crossing cycle $C$; $C$ is a clockwise cycle if $f \neq f_\infty$ and a counterclockwise cycle if $f = f_\infty$. The boundary $\partial H$ of a planar embedded graph $H$ is equal to $\partial f_\infty$.

A circulation is counterclockwise (abbreviated c.c.w.) if the potential of every face is nonnegative [65]. A circulation is clockwise (abbreviated c.w.) if the potential of every face is non positive. A cycle $P$ is clockwise if $\delta(P)$ is clockwise. A cycle or circulation may be neither counterclockwise nor clockwise, but a simple cycle is either clockwise or counterclockwise. These definitions have geometric interpretations, as illustrated by Figure 1.4.

![Figure 1.4](image)

Figure 1.4: (a) A clockwise cycle. (b) A cycle that is neither clockwise nor counterclockwise.

Consider two paths $P$ and $Q$ that share a common vertex $x$. Suppose $x$ is the head of a dart $d$ of $P$ that is not a dart of $Q$. Path $P$ is said to enter path $Q$ at vertex $x$. One can similarly define leaves. Suppose additionally that $x = head(a)$ and $x = tail(b)$ where $a$ and $b$ are darts of $Q$. Consider the embedded graph induced by $a$, $b$ and $d$. If $\pi(b) = rev(d)$ and $\pi(rev(d)) = rev(a)$, then $P$ is said to enter $Q$ from the left. If $\pi(rev(d)) = b$ and $\pi(rev(a)) = rev(d)$ then $P$ is said to enter $Q$ from the right. Suppose path $P$ and $Q$ have a maximal subpath $R$ in common (where $R$ may be the trivial path of one vertex) such that $P$ enters $Q$ at $start(R)$ on the right and $P$ leaves $Q$ at $end(R)$.
on the left then \( P \) crosses \( Q \) at \( x \) where \( x \) is a vertex of \( R \). Similarly, \( P \) can cross \( Q \) from left to right. The notion of enters and crossing are illustrated in Figure 1.5.

If \( P \) and \( Q \) are paths that do not cross for any vertex, then they are non-crossing. A path is non-self-crossing if for every pair of subpaths \( P \) and \( Q \) of the path, \( P \) does not cross \( Q \). See Figure 1.5 for examples of these situations.

![Figure 1.5](image_url)

Figure 1.5: (a) \( P \) crosses \( Q \): \( P \) enters \( Q \) on the right at \( x \) and \( P \) leaves \( Q \) on the left at \( y \). (b) \( P \) and \( Q \) are non-crossing. (c) This is a self-crossing cycle. (d) This is a non-self-crossing but non-simple cycle, since vertex \( v \) occurs twice.

We will use the following lemma in Section 2.3.2 where we will use non-self-crossing cycles instead of simple cycles as one may be tempted to use. This lemma allows us to build non-self-crossing cycles from other non-self-crossing cycles.

**Lemma 1.2.3** (Composition Lemma). Let \( C \) be a non-self-crossing cycle and let \( A \) be a non-self-crossing path with endpoints on \( C \) such that no part of \( A \) is enclosed by \( C \). Then \( A \circ C[\text{end}(A), \text{start}(A)] \) is a non-self-crossing cycle.

**Proof.** Since no part of \( A \) is enclosed by \( C \), \( A \) does not cross \( C \). It follows that \( A \circ C[\text{end}(A), \text{start}(A)] \) is non-self-crossing.

1.2.5 Clockwise and leftmost

An \( x \)-to-\( y \) walk \( A \) is left of an \( x \)-to-\( y \) walk \( B \) if \( \delta(A) - \delta(B) \) is a clockwise circulation. (This definition was given by [68] for paths, but generalizes naturally to walks.) Likewise \( A \) is right of \( B \) if \( \delta(A) - \delta(B) \) is a counterclockwise circulation. Left of and right of are transitive, reflexive, antisymmetric relations. An \( x \)-to-\( y \) path \( A \) is the leftmost \( x \)-to-\( y \) path in a graph if, for every \( x \)-to-\( y \) path \( B \), \( A \) is left of \( B \). There is not necessarily a leftmost walk: suppose \( P = Q \circ C \) is the leftmost path where \( Q \) is an \( x \)-to-\( y \) path and \( C \) is a c.w. cycle then \( R = P \circ C \) is a walk that is left of \( P \) and \( R \circ C \) is left of \( R \) and so on. However, the following lemma allows us to consider only simple paths in graphs with no clockwise cycles.
**Lemma 1.2.4.** Let $G$ be a graph with no clockwise cycles. If $P$ is a leftmost walk, then $P$ is a simple path.

**Proof.** Suppose $P$ is not a simple path. Let $x$ be a vertex that occurs at least twice on $P$. Let $x_1$ be the first occurrence of $x$ on $P$ and let $x_2$ be the last. Then $C = P[x_1, x_2]$ is a cycle. Since $G$ has no clockwise cycles, $C$ must be counterclockwise. Let $P' = P[1, x_1] \circ P[x_2, \cdot]$ be the path that is obtained from $P$ by removing $C$. The circulation

$$\delta(P) - \delta(P') = \delta(C)$$

is counterclockwise, so $P$ is right of $P'$; $P$ is not leftmost, a contradiction. □

**Lemma 1.2.5.** Every subpath of a leftmost path is a leftmost path.

**Proof.** Let $P$ be a leftmost path. Let $Q$ be an $x$-to-$y$ subpath of $P$. Suppose there is another $x$-to-$y$ path $Q' \neq Q$ that is left of $Q$. Let $P' = P[\cdot, x] \circ Q' \circ P[y, \cdot]$. The circulation

$$\delta(P') - \delta(P) = \delta(P[\cdot, x] \circ Q' \circ P[y, \cdot]) - \delta(P[\cdot, x] \circ Q \circ P[y, \cdot])$$

$$= \delta(Q') - \delta(Q)$$

is clockwise since $Q'$ is left of $Q$. So $P' \neq P$ is left of $P$, a contradiction. □

**Theorem 1.2.6** (Non-crossing Theorem). If $P$ and $Q$ are unique non-self-crossing $x$-to-$y$ paths that do not cross each other, then $P$ is either right of or left of $Q$.

**Proof.** Let $C = Q \circ \text{rev}(P)$: $C$ is a non-self-crossing cycle. Let $G_C$ be the graph induced on $C$. Since $G_C$ is a connected graph, each face of $G_C$ has a connected boundary. We show that $\delta(C)$ is either a clockwise or counterclockwise circulation. Recall that we equate a face with the set of darts forming the clockwise boundary of the face.

First we show that each face of $G_C$ uses either darts of $C$ or darts of $\text{rev}(C)$ (but not both). Suppose for a contradiction that $f$ is a face that uses darts of both $C$ and $\text{rev}(C)$. Let $A$ and $B$ be maximal subpaths of $C$ such that $A \in \partial f$ and $\text{rev}(B) \in \partial f$ and $\text{end}(A) = \text{start}(\text{rev}(B))$. Let $A'$ and $B'$ be the subpaths of $C$ such that $C = A \circ A' \circ B \circ B'$. Since $f$ is a face, $C$ does not cross $\partial f$ and so $A \circ A'$ must leave $\partial f$ from the left (at $\text{end}(A)$, by the maximality of $A$). Likewise, $B \circ B'$ leaves $\partial f$ from the left. Since $C$ is non-self-crossing, $A \circ A'$ does not cross $B \circ B'$. However $B'$ is an $\text{end}(B)$-to-$\text{start}(A)$ path and $A'$ is a $\text{end}(A)$-to-$\text{start}(B)$; $A'$ crosses $B'$. See Figure 1.6 for an illustration.

Since each face of $G_C$ uses either darts of $C$ or $\text{rev}(C)$, the following potential assignment is valid:

$$\phi[f] = \begin{cases} 
-1 & \text{if } \partial f \subset C \\
0 & \text{otherwise}
\end{cases}$$

If $\phi[f_\infty] = 0$, $\phi$ is a valid potential assignment and corresponds to the circulation $\eta = \delta C$. Then $C$ is clockwise and $Q$ is left of $P$.

If $\phi[f_\infty] = -1$, $\phi + 1$ is a valid potential assignment (where $1$ is the all-ones vector). It follows that $C$ is counterclockwise and $Q$ is right of $P$. □
Figure 1.6: If $A \circ A' \circ B \circ B'$ is a cycle and $A \cup \text{rev}(B) \in \partial f$, then $A'$ must cross $B'$.

1.2.6 Tree-cut operation

The TreeCut operation [69] will be useful for proving a number of properties in Chapter 3. It is illustrated in Figure 1.7. The operation *cuts open* a planar graph along a tree, producing another planar graph with a new face enclosed by the duplicated tree edges.

**TreeCut**($G, T$)

1. Let $C$ be the clockwise non-self-crossing cycle that forms the Euler tour of $T$.
2. Duplicate the edges of $T$ and replicate the nodes of $T$ according to their degree such that $C$ visits each of these edges only once.
3. Create a new face $f_T$ that is left of every dart in $C$.
4. Return the new graph.

Table 1.1: **TreeCut** takes as input a planar embedded graph $G$ and a tree $T$ and returns a graph where the edges of $T$ are duplicated and form a simple cycle that is the boundary of a new face.

Figure 1.7: The process of cutting open a graph along a tree, $T$ (bold edges): duplicate edges, replicate vertices and create a new face $f_T$. 
Chapter 2

Network Flow Problems

The history of maximum-flow and minimum-cut problems [96] is tied closely to planar graphs. During the height of the cold war, the United States spent considerable effort analyzing the Soviet rail network: “The success of interdiction depends largely on [the] interdiction-program efforts on the enemy’s capability to move men and supplies.” [50]. Modelling the Soviet rail network as a planar graph (by, in fact, taking the dual of the planar graph composed of boundaries of administrative districts with edges representing transportation capacity between these districts), Harris and Ross, as members of the RAND corporation, studied the problem of determining the best way to interdict the Soviet rail network. That is, they found the minimum number of rail lines that must be cut in order to stop movement of supplies and men between tactically important locations: they found a minimum cut in the graph. Ford and Fulkerson picked up on this line of research, leading to their landmark paper proving the max-flow, min-cut theorem and formulating the augmenting-path algorithm [41].

We now give the formal statement of the maximum-flow and minimum-cut problems. Given a graph $G$, a source vertex $s$, a sink vertex $t$ and capacities on the darts $c(d)$, the maximum-flow problem is

$$\max \ f \cdot \delta(s)$$

s.t.  

$$f \cdot \delta(v) = 0, \ \forall v \in V \setminus \{s, t\}$$

$$f[d] \leq c(d), \ \forall \text{ darts } d$$

where $f$ is a vector in arc-space. Constraint (2.1) is the conservation constraint: the net flow at every non-source-or-sink vertex is zero. Constraint (2.2) is the capacity constraint. If a capacity function $c$ is given only in terms of the arcs, then we define a capacity function $c'$ on the darts as $c'(\langle a, 1 \rangle) = c(a)$ and $c'(\langle a, -1 \rangle) = 0$ for each arc $a$. A flow assignment $f$ or $st$-flow is called feasible if it satisfies these constraints. The goal is to maximize the value of the flow, $f \cdot \delta(s)$. A flow of value zero is called a circulation and is a vector in cycle space.
Given the same input, the minimum-cut problem is:

\[
\min \quad c(\Gamma^+(S)) \\
\text{s.t.} \quad s \in S \subseteq V \setminus \{t\} \tag{2.3}
\]

A set of vertices \(S\) satisfying Constraint (2.3) is called an \(st\)-cut. The value of a cut is given by the objective function.

**Theorem 2.0.7** (Max-Flow Min-Cut). The value of the maximum \(st\)-flow is equal to the value of the minimum \(st\)-cut.

The Max-Flow Min-Cut Theorem was discovered independently by Ford and Fulkerson [41], Kotzig [72], and Elias et al. [32].

### 2.1 Maximum flow algorithms

In the same paper that proved the Max-Flow Min-Cut Theorem, Ford and Fulkerson suggested an algorithm (actually, a paradigm) for finding a maximum flow called the augmenting path algorithm. The algorithm is iterative: find a path \(P\) from the source to the sink and push flow on this path. That is, the value of the flow for each dart in \(P\) is increased by an amount \(\Delta\).

More formally, a dart \(d\) is residual with respect to \(f\) and \(c\) if \(f[d] < c[d]\). Otherwise, \(d\) is non-residual. A path is residual if all its darts are residual. It follows from the Max-Flow Min-Cut Theorem that a feasible \(st\)-flow \(f\) is maximum if and only if there is no residual \(s\)-to-\(t\) path with respect to \(f\) and \(c\). Augmenting an \(st\)-flow \(f\) along a residual \(s\)-to-\(t\) path \(P\), means increasing \(f[d]\) by the same amount for each dart \(d\) in \(P\). We call this path the augmenting path. Suppose that \(f\) is feasible with respect to \(c\). If the amount of the increase is no more than

\[
\Delta = \min_{d \in P} c(d) - f[d],
\]

then after augmentation the \(st\)-flow \(f\) is still feasible. If the increase is exactly \(\Delta\), then we say the augmentation saturates the path \(P\). In this case, at least one dart of \(P\) becomes saturated or non-residual.

Dinitz [29] and Edmonds and Karp [31] showed that if the shortest (with respect to number of darts) augmenting path is chosen then there are at most \(nm\) iterations. Dinitz gave an \(O(n^2m)\) analysis for this using the notion of a blocking flow. In [46], Goldberg and Rao gave a clever implementation resulting in an \(O(\min(n^{2/3}, m^{1/2})m \log \frac{n}{m} \log U)\)-time algorithm (where \(U\) is the largest integral capacity) by using a different adaptive notion of distance that is related to the residual capacities. This is the fastest known algorithm for maximum flow in a general graph and results in an \(O(n^{3/2} \log n \log U)\)-time algorithm for planar graphs.

We briefly mention another type of maximum flow algorithm: the push-relabel algorithm, alternatively known as the preflow-push algorithm [47]. Rather than pushing flow along paths, flow is pushed on individual arcs. This algorithm does not maintain a feasible flow and selects arcs to augment in order to bring the flow closer to feasibility.
2.2 History of planar maximum flow

In [41], Ford and Fulkerson give a particular augmenting-path algorithm for the case of finding the maximum $st$-flow in a planar graph in which the source and the sink are on the boundary of a common face, the infinite face. Such a graph is termed $st$-planar. With the graph viewed with the source embedded on the left and the sink on the right, the algorithm iteratively augments the uppermost residual path. This algorithm has the property that the flow on an arc is never decreased. Since each augmentation makes at least one arc non-residual, the algorithm requires at most $m$ augmentations, where $m$ is the number of arcs. We will give more details in Section 2.3.

In 1979, Itai and Shiloach [58] showed that each iteration of the uppermost path algorithm could be implemented in $O(\log n)$ time, where $n$ is the number of vertices, using a priority queue of the residual darts. Consequently, the algorithm can be carried out in $O(n \log n)$ time (using the fact that a simple planar graph with $n$ vertices has at most $3n$ arcs).

In 1991, Hassin demonstrated that a maximum $st$-flow in an $st$-planar graph $G$ could be derived from shortest-path distances in the planar dual $G^*$ of $G$ where capacities in $G$ are interpreted as lengths in $G^*$. With this insight, it can be seen that the uppermost-path algorithm can be interpreted in the planar dual as Dijkstra’s algorithm. The fact that the uppermost path algorithm can be implemented to run in $O(n \log n)$ time corresponds to the observation, due to Johnson [60], that Dijkstra’s algorithm could be implemented to run in $O(n \log n)$ time by using a priority queue. Frederickson showed later that shortest-path distances in a planar graph with nonnegative lengths could be computed in $O(n^{\sqrt{\log n}})$ time, and Henzinger et al. [55] showed subsequently that the same problem could be solved in $O(n)$ time; combining this with Hassin’s result yields an $O(n)$-time algorithm for maximum $st$-flow in $st$-planar graphs.

There remained, however, the more general and more natural problem of $st$-flow in a planar graph in which $s$ and $t$ need not be on the boundary of a common face. In 1983, Reif [90] showed that the minimum $st$-cut (and so, via the Max-Flow Min-Cut Theorem, also the value of the max $st$-flow) could be found in $O(n \log^2 n)$ time for the special case of undirected planar graphs. This algorithm uses the observation that the edges crossing a min $st$-cut form a minimum length cycle $C$ that separates $s$ from $t$ in the planar dual graph (where $s$ and $t$ are faces). The algorithm finds a shortest path $P$ in $G^*$ from a vertex adjacent to $s$ to a vertex adjacent to $t$. Reif proves that $C$ only crosses $P$ once. A divide-and-conquer algorithm is given in which a minimum separating cycle is found that contains the middle vertex of $P$: this cycle corresponds to an $st$-planar min cut in the primal. This results in an $O(n \log^2 n)$-time algorithm, using the aforementioned $O(n \log n)$ $st$-planar flow algorithm of Itai and Shiloach. In 1985, Hassin and Johnson [52] draw on Reif’s technique to show that the flow assignment could also be found within the same time bound, again for undirected planar graphs. The shortest path algorithms of Henzinger et al. [55] or Klein [68] can be used to re-implement these algorithms in $O(n \log n)$ time.

Still the more general problem of $st$-flow in a planar directed graph remained open. This problem is more general since the problem of maximum $st$-flow in an undirected graph can be converted to a directed problem by introducing two oppositely oriented arcs of equal capacity for each edge. In 1982,
Johnson and Venkatesan gave a divide-and-conquer algorithm that finds a flow of input value $v$ in a directed planar graph in $O(n\sqrt{n}\log n)$ [61]. The algorithm divides the graph using $O(\sqrt{n})$ balanced separators, finding a flow in each side of value $v$. However, the flow on the $O(\sqrt{n})$-boundary edges of each subproblem might not be feasible. Each boundary edge is made feasible via an $st$-planar flow computation.

In 1989, Miller and Naor [79] showed that finding a directed $st$-flow of value $v$ could be reduced to computing shortest-path distances in a graph with positive and negatives lengths. Here, $v$ units of flow are routed (perhaps infeasibly) along any $s$-to-$t$ path $P$. The capacity of a dart $d$ on $P$ may now be violated. We are required to route this excess flow through the rest of the graph for each dart on $P$. This is a feasible circulation problem and can be solved using shortest-path distances in the dual graph, where lengths may be negative. In 2001, Fakcharoenphol and Rao [40] presented the first sub-quadratic algorithm for computing shortest-path distances in a graph with positive and negative lengths. In these algorithms, the value $v$ of the flow can be found by parametric search, resulting in a sub-quadratic running time depending on $C$, the sum of the capacities. This is the first sub-quadratic algorithm for the general problem of finding a maximum $st$-flow: previous algorithms are limited to $st$-planar or planar undirected graphs.

<table>
<thead>
<tr>
<th>Year</th>
<th>Restriction</th>
<th>Time</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1956</td>
<td>$st$-planar</td>
<td>$O(n^2)$</td>
<td>Ford and Fulkerson [41]</td>
</tr>
<tr>
<td>1979</td>
<td>$st$-planar</td>
<td>$O(n \log n)$</td>
<td>Itai and Shiloach [58]</td>
</tr>
<tr>
<td>1982</td>
<td>value</td>
<td>$O(n\sqrt{n}\log n)$</td>
<td>Johnson and Venkatesan [61]</td>
</tr>
<tr>
<td>1983</td>
<td>value, undirected</td>
<td>$O(n \log^2 n)$</td>
<td>Reif [90]</td>
</tr>
<tr>
<td>1985</td>
<td>undirected</td>
<td>$O(n \log^2 n)$</td>
<td>Hassin and Johnson [52]</td>
</tr>
<tr>
<td>1987</td>
<td>$st$-planar</td>
<td>$O(n\sqrt{\log n})$</td>
<td>Hassin [51] using Frederickson [42]</td>
</tr>
<tr>
<td>1997</td>
<td>$st$-planar</td>
<td>$O(n)$</td>
<td>Hassin [51] using Henzinger et al. [55]</td>
</tr>
<tr>
<td>1997</td>
<td>undirected</td>
<td>$O(n \log n)$</td>
<td>Hassin and Johnson [52] using Henzinger et al. [55]</td>
</tr>
<tr>
<td>2001</td>
<td></td>
<td>$O(n \log^3 n \log C)$</td>
<td>Miller and Naor [79] using Fakcharoenphol and Rao [40]</td>
</tr>
</tbody>
</table>

Table 2.1: Planar Maximum-Flow and Minimum-Cut Algorithms

2.2.1 Toward an $O(n \log n)$ algorithm

In 1994, Weihe [108] published an $O(n \log n)$ algorithm for planar directed maximum $st$-flow with a rather complicated proof of correctness. The algorithm, though inspired by Ford and Fulkerson’s uppermost-path algorithm, is also quite complicated. From the example included in the paper, it is clear that the uppermost path (as generalized to non-$st$-planar graphs) is not the augmenting path. In a preprocessing step, the input graph is transformed into one satisfying the following three requirements.

1. Each vertex but the source and sink has degree exactly three;

2. there are no clockwise cycles; and
3. each arc $uv$ belongs to a simple $s$-to-$v$ path and a simple $u$-to-$t$ path.

Satisfying Requirement 1 involves: splicing together every two successive arcs sharing an endpoint of degree one; and replacing each vertex of high degree by a cycle, increasing the number of vertices to $2m$, which is at most $6n$. Requirement 2 can be satisfied by using a reduction of Khuller, Naor, and Klein [65] to computing shortest-path distances in the dual (and so can be computed in $O(n)$ time using the algorithm of Henzinger et al. [55]). Details of this step will be given in Section 2.3.1.

Requirement 3 is problematic. Weihe states “To satisfy this assumption, simply remove all arcs that violate it. None of these arcs will help us solve our problem.” However, as pointed out by Biedl, Brejová, and Vinař [13], there is no known $O(n \log n)$-time algorithm to delete all such arcs. They give the best known algorithm to date, which runs in $O(n^2)$ time. To our knowledge, the dependence of Weihe’s proof of correctness on Requirement 3 has not been resolved. Although Weihe has claimed that his algorithm can be corrected, this has not been verified.

### 2.3 Leftmost-path algorithm

We revisit the uppermost-path algorithm of Ford and Fulkerson [41]. Recall that in each iteration, the uppermost residual path is saturated. We forgo a rigorous definition of uppermost as it is superseded by the more general leftmost (the uppermost path is the leftmost path). However, using an intuitive definition, we get the following lemma:

**Lemma 2.3.1.** Let $P_i$ be the uppermost residual path in the $i$th iteration of the algorithm. $P_{i+1}$ is below $P_i$.

**Proof.** For a contradiction, suppose that $P_{i+1}$ is not below $P_i$. Then there is an $x$-to-$y$ subpath of $P_{i+1}$ that is above an $x$-to-$y$ subpath of $P_i$. From this we can construct a path that was residual in the $i$th iteration that is above $P_i$. □

We present an algorithm to find a maximum $s$-$t$-flow in a directed planar graph that runs in $O(n \log n)$ time. The algorithm is a direct generalization of the uppermost-path algorithm. Ford and Fulkerson’s algorithm finds the uppermost flow: one in which no flow can be rerouted above the existing flow. Our generalization finds the leftmost flow (which we define in the next section, and is defined with respect to the infinite face). At the start of the algorithm, we start with a leftmost flow of value zero which is achieved via a preprocessing step equivalent to satisfying Requirement 2 of Weihe’s algorithm. The algorithm, at an abstract level, is:

Designate a face adjacent to $t$ as $f_\infty$.

Saturate the clockwise cycles. (LEFTMOSTCIRCULATION)

While there is a residual $s$-to-$t$ path, saturate the leftmost such path. (MAXFLOW)

We will first illustrate how the notions of clockwise and leftmost are related for circulations and give an algorithm for saturating the clockwise cycles, LEFTMOSTCIRCULATION. We will then extend this
definition to flows. The analysis of MaxFlow will assume that the input graph has no clockwise residual cycles.

2.3.1 Leftmost circulations and flows

In 1993, Khuller, Naor and Klein showed that circulations in planar graphs form a finite distributive lattice [65]. A distributive lattice is a partial order such that each pair of elements has a meet (greatest lower bound) and a join (least upper bound). Meet and join are distributive operators. We call the unique minimum of the circulation lattice the leftmost circulation. The leftmost circulation in the lattice is the vector in cycle space whose corresponding potential function is maximized subject to the capacity constraints:

\[
\max \phi \cdot 1 \\
\text{s.t. } \phi[\text{head}_{G^*}(d)] - \phi[\text{tail}_{G^*}(d)] \leq c(d), \forall \text{ darts } d
\]

This linear program exactly solves the shortest path problem in the dual planar graph: the potentials found correspond to shortest path distances from \( f_\infty \) to every face, interpreting capacities as distances. We are interested in the properties of the residual graph given by the circulation corresponding to these potentials. The algorithm LEFTMOSTCIRCULATION is expanded from [65] (Table 2.2).

<table>
<thead>
<tr>
<th>LEFTMOSTCIRCULATION ((G_{in}, c_{in}, f_\infty))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Interpret capacities ( c_{in}(d) ) as lengths of darts in the dual graph ( G^*_{in} ).</td>
</tr>
<tr>
<td>2. Let ( \phi[f] = \text{dist}<em>{G^*</em>{in}}(f_\infty, f) ) for every face ( f ).</td>
</tr>
<tr>
<td>3. Let ( \eta[d] = \phi[\text{head}<em>{G^*</em>{in}}(d)] - \phi[\text{tail}<em>{G^*</em>{in}}(d)] ) for every dart ( d ).</td>
</tr>
<tr>
<td>4. Let ( G_{out} ) be the residual graph with respect to ( \eta ).</td>
</tr>
<tr>
<td>5. Let ( c_{out}(d) = c_{in}(d) - \eta[d] ) for every dart ( d ).</td>
</tr>
<tr>
<td>6. Return ((G_{out}, c_{out})).</td>
</tr>
</tbody>
</table>

Table 2.2: LEFTMOSTCIRCULATION takes as input a directed planar embedded graph, an identified face (the infinite face) and a dart capacity function. The algorithm returns the residual graph that has no clockwise residual cycle.

**Lemma 2.3.2.** The residual graph returned by LEFTMOSTCIRCULATION \((G_{in}, c_{in}, f_\infty)\) has no residual clockwise cycle.

**Proof.** Let \((G_{out}, c_{out}) = \text{LEFTMOSTCIRCULATION} \((G_{in}, c_{in}, f_\infty)\). Let \( C \) be a clockwise cycle of darts in \( G_{out} \) and let \( T \) be the tree representing the shortest-path distances computed in LEFTMOSTCIRCULATION. There is a path in \( T \) from \( f_\infty \) to every face enclosed by \( C \), so at least one dart of \( C \) is
in the shortest path tree. Let $d$ be such a dart:

$$c_{\text{out}}(d) = c_{\text{in}}(d) - \eta[d]$$

$$= c_{\text{in}}(d) - \left(\text{dist}_{G_{\text{in}}}(f_{\infty}, \text{tail}_{G_{\text{in}}}(d)) - \text{dist}_{G_{\text{in}}}(f_{\infty}, \text{head}_{G_{\text{in}}}(d))\right)$$

$$= c_{\text{in}}(d) - c_{\text{in}}(d) \quad \text{since } d \text{ is in the shortest-path tree}$$

$$= 0$$

Since $d$ is not residual, $C$ is not residual. □

Two $st$-flows $f_1$ and $f_2$ of the same value differ by a circulation. That is, $f_1 - f_2$ is a circulation. If this circulation is clockwise, then $f_1$ is left of $f_2$ [108]. If this circulation is counterclockwise, then $f_1$ is right of $f_2$. A flow $f$ is a leftmost flow of its value if for every other flow $f'$ of the same value, $f$ is left of $f'$. Alternatively, a $f$ is a leftmost flow if the residual graph has no clockwise residual cycle.

**Lemma 2.3.3.** Let $f$ be a leftmost flow and let $P$ be the leftmost residual path. Saturating $P$ results in a leftmost flow $f'$.

**Proof.** Suppose for a contradiction that $f'$ is not a leftmost flow. By the definition of leftmost, there must then be a clockwise residual cycle $C$ with respect to $f'$. Assume w.l.o.g. that $C$ is simple. Since $f$ was leftmost, $C$ was not residual prior to the augmentation, and so $P$ must have a dart $d$ in common with $\text{rev}(C)$.

Let $P' = P[, \text{tail}(d)] \circ C[\text{tail}(d), \text{tail}(d)] \circ P[\text{tail}(d), .]$ where $C[\text{tail}(d), \text{tail}(d)]$ is the cycle starting at $\text{tail}(d)$. $P'$ is a walk and $\delta(P') - \delta(P) = \delta(P) + \delta(C) - \delta(P) = \delta(C)$ is a clockwise circulation and so $P'$ is left of $P$. Therefore $P$ is not the leftmost path. □

### 2.3.2 Unusability Theorem

We will now concentrate on an abstract version of MAXFLOW (Table 2.3). Let $(G_0, c_0) = (G_{\text{out}}, c_{\text{out}})$

|Table 2.3: MAXFLOW takes as input a directed planar embedded graph $G_0$ and capacities $c_0$ with no clockwise residual cycles (as output by LEFTMOSTCIRCULATION$(G_{\text{in}}, c_{\text{in}}, f_{\infty})$ where $f_{\infty}$ is any face adjacent to $t$) and returns the maximum flow from $s$ to $t$. |

<table>
<thead>
<tr>
<th>(Abstract) MAXFLOW$(G_0, c_0, s, t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Initialize $f = 0$.</td>
</tr>
<tr>
<td>2. While there is a residual $s$-to-$t$ path w.r.t. $f$ and $c_0$, saturate the leftmost $s$-to-$t$ path, modifying $f$.</td>
</tr>
<tr>
<td>3. Return $f$.</td>
</tr>
</tbody>
</table>

be the output of LEFTMOSTCIRCULATION$(G_{\text{in}}, c_{\text{in}}, f_{\infty})$ where $f_{\infty}$ is any face adjacent to $t$. We assume that $G_0$ and $c_0$ are the input to MAXFLOW. To aid in the analysis, we will use the interpretation that every arc of $G_0$ has nonzero capacity: for each arc $a$ of $G_{\text{out}}$, if $c_{\text{out}}(a) = 0$ then interpret $\text{rev}(a)$ as an arc of $G_0$, else interpret $a$ as an arc. In what follows, $arc$ always refers to an arc of $G$, anti-arc

---

---
refers to a dart whose reverse is an arc. Since the output of \textsc{LeftmostCirculation} is a graph with no clockwise residual cycles (Lemma 2.3.3), we have the following property.

**Property 2.3.4.** The input graph to \textsc{MaxFlow} has no clockwise cycle of arcs.

The following invariant of \textsc{MaxFlow} follows from Lemma 2.3.3:

**Invariant 2.3.5.** During the execution of \textsc{MaxFlow}, $G_0$ has no clockwise residual cycles with respect to $f$.

Recall that in Ford and Fulkerson’s uppermost-path algorithm, once flow is pushed on an arc, flow can never be removed from that arc. This is not the case if the graph is not $st$-planar, as illustrated in Figure 2.1. However, we show that while you can remove flow from an arc, you cannot push flow back onto that arc, as stated in the following Unusability Theorem.

![Figure 2.1: A simple example illustrating that flow can be removed from an arc in MaxFlow, even in the case of unit capacities. On the left, the leftmost residual path (dotted) pushes flow along the bottom arc. On the right is the resulting residual graph. The leftmost residual path (dotted) removes flow from the bottom arc.](image)

**Theorem 2.3.6** (Unusability Theorem). Suppose an arc $a$ is augmented and some time later $\text{rev}(a)$ is augmented. Then arc $a$ cannot be augmented again.

The remainder of this section will be devoted to proving the Unusability Theorem. The structure of the proof is as follows. We show that if an arc $a$ is augmented and later $\text{rev}(a)$ is augmented, then a structure in the residual graph arises called an *obstruction* (Lemma 2.3.17). We show that this structure persists under leftmost augmentations (Lemma 2.3.18) and is a witness to $a$ never belonging to a leftmost residual path (Lemma 2.3.16).

Since the augmentations of the algorithm are always along leftmost residual paths, we have a number of properties which we give in the following lemma and which will be used in the remainder of this section.

**Lemma 2.3.7** (Prohibited augmentations). The following situations are not permitted if $A$ is a leftmost augmentation and the given vertex indices are well-defined:

1. $A[x,y]$ is right of a residual path $R[x,y]$.
3. A has a dart that enters a simple t-to-s residual path R from the right.

Proof. We prove each part separately.

1. \(A[s, x] \circ R[x, y] \circ A[y, t]\) is left of \(A\). This contradicts the requirement that \(A\) is the leftmost residual path.

2. This contradicts Invariant 2.3.5.

3. Suppose \(uv\) is a dart of a leftmost augmentation path and suppose \(uv\) enters a t-to-s residual path \(R\) from the right. As such, \(uv \notin R\) and \(rev(uv) \notin R\). \(A[s, u]\) must intersect \(R\) at some vertex: let \(x\) be the last intersection of \(A[s, u]\) with \(R\) (possibly \(x = s\)). If \(x \in R(v, s]\), then \(A[x, v] \circ R[v, x]\) is a clockwise residual cycle, contradicting Invariant 2.3.5. If \(x \in R(t, v)\) then \(R[x, v]\) is a residual path that is left of \(A[x, v]\), which is a prohibited augmentation of the second kind. \(\square\)

Now we define what it means to be unusable. Unusability is given by a structure in the residual graph called an obstruction.

**Definition 2.3.8 (Usable Arc).** An obstruction cycle (or more simply, an obstruction) is a clockwise non-self-crossing cycle \(L \circ M\) where \(L\) is residual and \(M\) consists entirely of arcs. We say it is an obstruction for an arc \(a\) if \((a, 1)\) is the first dart of \(L\). We say an arc \(a\) is usable if there is an obstruction for \(a\).

We give a second, equivalent representation for an obstruction which will be useful in proving the Usability Theorem. Both are illustrated in Figure 2.2.

![Figure 2.2]({image-url})

Figure 2.2: (a) An obstruction for arc \(a\) as given by Definition 2.3.8. (b) The obstruction for arc \(a\) as given by Lemma 2.3.9 with the obstruction from (a) shaded in the background. \(L\), \(Q^2\) and \(R\) are residual; \(M\), \(Q^1\) and \(Q^2\) consist of arcs; \(a\) is the first arc of \(L\) and \(Q^2\).

**Lemma 2.3.9.** An obstruction for arc \(a\) is equal to the clockwise cycle \(Q^1 \circ Q^2 \circ R\) where

1. \(Q^1 \circ Q^2\) consists entirely of arcs,

2. \(Q^2 \circ R\) is residual,

3. \(a\) is the first dart of \(Q^2\),
4. there is flow through the vertex start(R), and
5. there is flow through the vertex end(R).

Proof. Let \( L \circ M \) be an obstruction for \( a \). Since \( G_0 \) has no clockwise cycles, \( L \circ M \) cannot consist entirely of arcs. Let \( b \) be the first anti-arc of \( L \). By Invariant 2.3.5, \( L \circ M \) cannot consist entirely of residual darts and so \( M \) cannot consist entirely of residual darts. Let \( c \) be the first non-residual dart of \( M \).

Let \( Q^1 = M[\text{tail}(c), \cdot] \), let \( Q^2 = L[\cdot, \text{tail}(b)] \), and let \( R = L[\text{tail}(b), \cdot] \circ M[\cdot, \text{tail}(c)] \). By choice of \( b \), \( L[\cdot, \text{tail}(b)] \) consists entirely of arcs, so property 1 holds. By choice of \( c \), \( M[\cdot, \text{tail}(c)] \) is residual, so property 2 holds. Since \( a \) is the first dart of \( L \), property 3 holds. Since \( b \) is a residual anti-arc, \( \text{rev}(b) \) carries flow, so property 4 holds. Since \( c \) is a non-residual arc, it carries flow, so property 5 holds.

Definition 2.3.10. For an unusable arc \( a \), let \( \Delta_a \) denote the obstruction for \( a \) that encloses the minimum number of faces (breaking ties arbitrarily). Write \( \Delta_a \) as \( Q^1_a \circ Q^2_a \circ R_a \), and let \( Q_a \) denote \( Q^1_a \circ Q^2_a \).

The definition of a minimally enclosing obstruction provides a number of properties of the residual graph.

Property 2.3.11. Suppose \( a \) is unusable. There is no residual path enclosed by \( \Delta_a \) from a vertex in \( Q^2_a[\cdot, \cdot] \) to a vertex in \( Q^1_a \).

Proof. Assume for a contradiction that \( W \) is such a residual path. Then \( W \circ Q^1_a[\text{end}(W), \cdot] \circ Q^2_a[\cdot, \text{start}(W)] \) is an obstruction for \( a \) that encloses fewer faces than \( \Delta_a \) does. That is, \( W \) can be used to replace \( R_a \) in the obstruction.

Property 2.3.12. Suppose \( a \) is unusable. \( Q^2_a \) belongs to a \( t \)-to-\( s \) residual path.

Proof. Since \( \Delta_a \) is a clockwise cycle, it cannot be residual, so \( Q^1_a \) cannot be residual. Let \( b \) be the last non-residual dart of \( Q^1_a \). Since \( Q^1_a \) contains only arcs, \( b \) carries flow and this flow must be routed to \( t \). Let \( F_t \) be any head(\( b \))-to-\( t \) flow path and let \( F_s \) be any s-to-start(\( R_a \)) flow path. Since the reverse of a flow path is residual, the path \( \text{rev}(F_t) \circ Q^1_a[\text{head}(b), \cdot] \circ Q^2_a \circ \text{rev}(F_s) \) is a residual \( t \)-to-\( s \) path.

Property 2.3.13. There are no flow paths enclosed by \( \Delta_a \) between vertices on the boundary of \( \Delta_a \).

Proof. Assume for contradiction that \( F \) is such a simple flow path. Let \( \alpha = \text{start}(F) \) and \( \beta = \text{end}(F) \). Then \( C_1 = \Delta_a[\alpha, \beta] \circ \text{rev}(F) \) and \( C_2 = F \circ \Delta_a[\beta, \alpha] \) are clockwise non-self-crossing cycles, each enclosing fewer faces than \( \Delta_a \). See Figure 2.3.

We will refer to the following:

Argument 1 Note that \( \text{rev}(F) \) is residual. If \( \Delta_a[\alpha, \beta] \) were residual then \( C_1 \) would be a residual clockwise cycle, contradicting Invariant 2.3.5. Since all non-residual darts of \( \Delta_a \) are in \( Q^1_a \), we infer that \( \Delta_a[\alpha, \beta] \) must include at least one dart of \( Q^1_a \).
Figure 2.3: An illustration of the cycles $C_1$ and $C_2$ for the the proof of Property 2.3.13. By Argument 1, there is an arc (bold) of $Q_a^1$ in $\Delta_a[\alpha, \beta]$. By Argument 2, there is an arc (grey) of $R_a$ in $\Delta_a[\beta, \alpha]$.

Argument 2 Note that $F$ consists entirely of arcs. If $\Delta_a[\beta, \alpha]$ consisted entirely of arcs then $C_2$ would be a clockwise cycle of arcs in $G_0$, contradicting Property 2.3.4. Since all anti-arcs of $\Delta_a$ are in $R_a$, we infer that $\Delta_a[\beta, \alpha]$ must include at least one dart of $R_a$.

There are two cases to consider:

Case 1 $\text{start}(R_a)$ is a vertex of $\Delta_a[\alpha, \beta]$: By Argument 1, $Q_a^1$ is not a subpath of $\Delta_a[\beta, \alpha]$. If $a$ is in $\Delta_a[\alpha, \beta]$ then $C_1$ is an obstruction enclosing fewer faces than $\Delta_a$. If $a$ is in $\Delta_a[\beta, \alpha]$ then $C_2$ is an obstruction enclosing fewer faces than $\Delta_a$.

Case 2 $\text{start}(R_a)$ is a vertex of $\Delta_a(\alpha, \beta)$: By Argument 2, $R_a$ is not a subpath of $\Delta_a[\alpha, \beta]$, so $\text{end}(R_a)$ is outside $\Delta_a[\alpha, \beta]$. By Argument 1, $Q_a^1$ is not a subpath of $\Delta_a[\beta, \alpha]$, so $a$ is a vertex of $Q_a^1$. Therefore the first arc of $Q_a^2$, which is $a$, is in $\Delta_a[\alpha, \beta]$, so $C_1$ is an obstruction enclosing fewer faces than $\Delta_a$.

Each case contradicts the minimality condition of $\Delta_a$. \hfill \Box

**Corollary 2.3.14.** If $\Delta_a$ encloses a flow-carrying dart $d$, then $\Delta_a$ encloses the source $s$ and every $s$-to-head($d$) flow path.

**Proof.** Suppose for a contradiction that there is a flow-carrying dart $d$ enclosed by $\Delta_a$ such that there is an $s$-to-head($d$) flow path $P$ such that $e$ is a dart of $P$ and $e$ is not contained by $\Delta_a$. Let $Q$ be a head($d$)-to-$t$ flow path. The path $P \circ Q$ contains a subpath that starts at a vertex (head($e$)) not strictly enclosed by $\Delta_a$, goes through a dart ($d$) enclosed by $\Delta_a$, and ends at a vertex ($t$) not strictly enclosed by $\Delta_a$. Such a flow path violates Property 2.3.13. \hfill \Box

For an unusable arc $a$ there is a $\text{start}(R_a)$-to-$t$ flow path and an $s$-to-end$(R_a)$ flow path by Parts 4 and 5 of Lemma 2.3.9.
Corollary 2.3.15. For an unusable arc $a$, any start($R_a$)-to-$t$ flow path does not intersect any s-to-end($R_a$) flow path.

Proof. Let $F_t$ be any start($R_a$)-to-$t$ flow path and let $F_s$ be any s-to-end($R_a$) flow path. Suppose for a contradiction that $F_t$ and $F_s$ share a vertex. Let $w$ be the first such vertex in $F_t$. Let $F'_s$ be the maximal suffix of $F_s$ that is not internal to $\Delta_a$. By Corollary 2.3.14, $F'_s$ is the only part of $F_s$ that is not internal to $\Delta_a$. Since no arc of $F_t$ is interior to $\Delta_a$, $w$ must be a vertex of $F'_s$.

Let $F = F_t[start(R_a), w] \circ F'_s[w, end(R_a)]$. $F$ is a start($R_a$)-to-end($R_a$) flow path that is not internal to $\Delta_a$. See Figure 2.4. By the Non-crossing Theorem, $F$ is either right of or left of $R_a$. There are two cases.

Case 1 If $F$ is right of $R_a$, $R_a \circ rev(F)$ is a clockwise residual cycle, contradicting Invariant 2.3.5.

Case 2 If $F$ is left of $R_a$, $F$ is also left of $rev(Q_a)$ by transitivity. Hence $F \circ Q_a$ is a clockwise cycle in $G_0$, a contradiction. This case is illustrated in Figure 2.4.

Figure 2.4: If flows to and from $\Delta_a$ (dashed), $F_s$ and $F_t$ share a vertex $w$, we can construct from them a start($R_a$)-to-end($R_a$) flow path $F$ (bold). The shaded area is bounded by a clockwise cycle of arcs.

Now we have all the tools needed to prove the Unusability Theorem. We prove it in three parts. First we show that an unusable arc cannot belong to a leftmost residual path (Lemma 2.3.16). Next we show that if an arc $a$ satisfies the condition of the Unusability Theorem, there is an obstruction for $a$ in the residual graph (Lemma 2.3.17). Finally we show that obstructions persist under leftmost augmentations (Lemma 2.3.18). The Unusability Theorem follows.

Lemma 2.3.16 (Unusable Arc Consequence). A leftmost augmenting path contains no unusable arcs.

Proof. Let $A$ be the leftmost augmenting path, and assume for a contradiction that it goes through an unusable arc $a$.

The goal is to first construct a non-self-crossing cycle, $C$, that encloses $a$ and does not enclose $t$. $A[tail(a), \cdot]$ must therefore cross $C$. We will show that this results in a contradiction.
By the definition of $\Delta_a$, there is an $s$-to-$end(R_a)$ flow path. Let $F_s$ be any such path and let $P_1 = Q^2_s \circ R_a \circ rev(F_s)$. $P_1$ is a residual head(a)-to-$s$ path. Let $A_1$ be the maximal suffix of $A[s, tail(a)]$ that does not cross $P_1$. Let $C_1 = A_1 \circ P_1$: $C_1$ is a residual cycle and since there are no c.w. residual cycles, it is c.c.w. $C_1$ is also non-self-crossing by construction.

We next define another c.c.w. non-self-crossing cycle, $C_2$, whose boundary is given by subpaths of $Q^2_a$, $A$, and a flow path not enclosed by $\Delta_a$. If $P_1$ does not include $start(R_a)$, define $C_2 = C_1$. Note that in this case, $P_1$ is a subpath of $Q^2_a$. Otherwise, we consider the following construction. By the definition of $\Delta_a$, there is a $start(R_a)$-to-$t$ flow path. Let $F_t$ be any such path and let $F_t'$ be the maximal prefix of $F_t$ that is enclosed by $C_1$ (possibly the empty path). By Corollary 2.3.15, $F_t$ and $F_s$ do not share any vertices and by Corollary 2.3.14, no part of $F_t$ is enclosed by $\Delta_a$. We conclude that $end(F_t')$ is in $A_1$ and define $C_2 = F_t' \circ A_1[\text{end}(F_t') \cdot \circ P_1[\cdot, \text{start}(F_t')].$ Note that $P_1[\cdot, \text{start}(F_t')] = Q^2_s[tail(a), \cdot].$ By definition, $C_2$ is non-self-crossing.

Notice that in both cases, $R_a$ is not enclosed by $C_2$. Let $P_2$ be the maximal prefix of $R_a \circ Q^1_a$ that is not enclosed by $C_2$. Applying the Composition Lemma to $C_2$ and $rev(P_2), we get a non-self-crossing cycle, $C$ whose boundary is composed of subpaths of $F_t, A, rev(Q^1_a), rev(R_a),$ and possibly $rev(Q^2_a).$ Further, $C$ is c.c.w. and encloses $a$.

Let $A_3$ denote the maximal prefix of $A[head(a), \cdot]$ that does not cross $C$. The last two cases are illustrated in Figure 2.5.

Case 1 $end(A_3) \in Q^2_a \circ R_a$. Let $P_3 = Q^2_s \circ R_a$: $P_3$ is a boundary of $C$ and since $A_3$ is enclosed by $C, A_3$ is right of $P_3[start(A_3), end(A_3)],$ violating Part 1 of Lemma 2.3.7.

Case 2 $end(A_3) \in rev(F_t').$ Since $rev(F_t')$ is a subpath of a $t$-to-$s$ residual path, this case contradicts Part 3 of Lemma 2.3.7.

Case 3 $end(A_3) \in Q^1_a$. Let $A_4$ be the maximal suffix of $A_3$ that is internal to $\Delta_a$. The only boundary vertices of $\Delta_a$ that are not boundary vertices of $C$ are the vertices of $Q^2_a$ so $start(A_4)$ must be a vertex of $Q^2_a$. This case therefore contradicts Property 2.3.11.

\begin{lemma}[Unusable Arc Creation] If augmentation $A$ uses arc $a$ in the reverse direction, $a$ will be unusable after augmentation $A$.
\end{lemma}

\begin{proof}
Let $a$ be an arc and let $A$ be the leftmost residual path. Suppose $d$ is a dart in $A$ where $d = rev(a)$. Since $d$ is residual, $a$ must carry flow. Let $F$ be any $s$-to-head($a$) flow path. Let $x$ be the last vertex of $A[\cdot, tail(a)]$ that is in $F$. Let $L = rev(A[x, tail(a)])$ and let $M = F[x, tail(a)].$ See Figure 2.6.

Both $L$ and $M$ are simple and by the choice of $x$, $L$ does not cross $M$. $L$ is residual after augmentation and $a$ is the first dart of $L$. $M$ consists entirely of arcs. Since $rev(M)$ is residual before augmentation, $A[x, tail(a)]$ must make a c.c.w. cycle with it by Part 2 of Lemma 2.3.7. Therefore $M \circ L$ is a c.w. non-self-crossing cycle: it is an obstruction for $a$.
\end{proof}

\begin{lemma}[Unusable Arc Persistence] Once an arc becomes unusable, it remains unusable.
\end{lemma}
Figure 2.5: (a) An example of a possible augmentation that uses an unusable arc, $a$ as outlined in Lemma 2.3.16. In this particular example, $C = A[end(F'), tail(a)] ◦ rev(Q^1) ◦ rev(R) ◦ F'$. (b) The counterexample as described in Case 2 of Lemma 2.3.16. $A$ cannot escape $C$ via $F'$ or $R$. (c) The counterexample as described in Case 3 of Lemma 2.3.16. $A$ cannot escape $C$ via $Q^1$.

Figure 2.6: The creation of an obstruction (whose interior is shaded) from the flow path (dotted) through $a$ (grey) and the augmentation path through $rev(a)$ (solid).

Proof. Let $a$ be an unusable arc and let $A$ be the leftmost residual $s$-to-$t$ path. Saturating $A$ can only change the fact that $R_a$ is residual. Assume that $A$ and $R_a$ share a dart.

Let $b$ be the first dart of $R_a$ that is in $A$. Let $A_1$ be the maximal suffix of $A[\cdot, tail(b)]$ that is not enclosed by $\Delta_a$. Since $A$ cannot enter $R_a$ from the right by Part 2 of Lemma 2.3.7 and since the right of $R_a$ is enclosed by $\Delta_a$, $A_1$ is not a trivial path.

The following cases are illustrated in Figure 2.7.

Case 1 $A_1 \neq A[\cdot, tail(b)]$. Let $c$ be the last dart of $A_1$ that is enclosed by $\Delta_a$. Both $Q^2_a$ and $R_a$ belong to $t$-to-$s$ residual paths (Property 2.3.12 and by consequence of Lemma 2.3.9, resp.). Since $c$ is enclosed by $\Delta_a$ and so enters $\Delta_a$ from the right, $head(c)$ cannot belong to either $Q^2_a$ or $R_a$ by Part 3 of Lemma 2.3.7. So $head(c)$ is on $Q^1_a$. Let $P = Q^1_a[head(c), \cdot] ◦ Q^2_a ◦ R_a[\cdot, tail(b)]$. Since $A_1$ does not cross $P$, $A_1$ is either left of or right of $P$.

(a) $A_1$ is left of $P$. Let $F_t$ be any $start(R_a)$-to-$t$ flow path as guaranteed by Part 4 of Lemma 2.3.9. This flow path is a part of a $t$-to-$s$ residual path $P'$ guaranteed by
Property 2.3.12. Let $C$ be the cycle obtained by applying the Composition Lemma to $\Delta_a$ and $A_1$: this cycle encloses the last dart $d$ of $Q^2_a$. Since $d \in P'$ and $F'_1$ is in not part enclosed by $\Delta_a$ by Property 2.3.13, $P'$ enters $C$ via a part $A_1$. Since $C$ is c.w. by construction, $A_1$ must enter $P'$ from the right. This contradicts Part 3 of Lemma 2.3.7. 

(b) $A_1$ is right of $P$. Since $\text{rev}(A_1)$ is residual after augmentation, $\text{rev}(A_1) \circ P$ is an obstruction for $a$ after augmentation.

Case 2 $A_1 = A[\cdot, \text{tail}(b)]$. Let $F_s$ be any $s$-to-$\text{end}(R_a)$ flow path. Let $A_2$ be the maximal suffix of $A_1$ that does not cross $F_s$. Let $F'_s = F_s[\text{start}(A_2), \cdot]$. Since $\text{start}(A_2)$ is not enclosed by $\Delta_a$, $F'_s$ starts outside the interior of $\Delta_a$, and so by Property 2.3.13 no part of $F'_s$ is interior to $\Delta_a$. Let $C = F'_s \circ \text{rev}(R_a[\text{tail}(b), \cdot]) \circ \text{rev}(A_2)$. By the choice of $A_2$, $C$ is non-self-crossing.

By Part 2 of Lemma 2.3.7, $\text{rev}(C)$ is c.c.w. and so $C$ is c.w. Let $C_1$ be the composition of $C$ and $\Delta_a$. Since the interior of $C$ is disjoint from the interior of $\Delta_a$ and $C_1$ is c.w., $C_1$ is non-self-crossing. Let $Q^1 = F'_s \circ Q_a^1$ and $R = R_a[\cdot, \text{tail}(b)] \circ \text{rev}(A_2)$. Then $C_1 = Q^1 \circ Q_a^2 \circ R$ is an obstruction for $a$ after augmentation since $\text{rev}(A_2)$ is residual after augmentation.

This completes the proof of the Unusability Theorem.

2.3.3 Implementation of MaxFlow

For the sake of bounding the running time, we next give a implementation of $\text{max-flow}(G_0, c_0, s, t)$ as a kind of network-simplex algorithm (Table 2.4). Later in this section we show that the implementation indeed implements the abstract version. The algorithm maintains a spanning tree $T$ of the graph rooted at the sink $t$ and the corresponding dual spanning tree $T^*$ rooted at the infinite face $f_\infty$.

Right-first search [92] in Step 2 constructs a tree $T$ spanning every vertex $v$ that can reach $t$ in $G_0$, and the path $T[v]$ is the leftmost $v$-to-$t$ path in $G_0$. The primal tree $T$ is represented using a dynamic-tree data structure [1, 3, 42, 99, 103], enabling Steps 6, 7, 11 and 12 to run in $O(\log n)$ time.
(Implementation) \( \text{MAXFLOW}(G_0, c_0, s, t) \)

1. Initialize \( f = 1 \).
2. Initialize \( T \) to be the right-first search tree searching backwards from \( t \) (disregarding the orientation of the arcs).
3. Let \( G \) be the graph obtained from \( G_0 \) by deleting all vertices not in \( T \).
4. Initialize \( T^* \) to consist of the darts of \( G \) whose arcs are not in \( T \).
5. Repeat:
   6. If \( T[s] \) is residual, saturate \( T[s] \), modifying \( f \).
   7. Let \( d \) be the last non-residual dart in \( T[s] \).
   8. If \( \text{tail}_{G^*}(d) \) is a descendant in \( T^* \) of \( \text{head}_{G^*}(d) \) then return \( f \).
   9. Let \( e \) be the parent dart in \( T^* \) of \( \text{head}_{G^*}(d) \).
   10. Eject \( e \) from \( T^* \) and insert \( d \) into \( T^* \).
   11. Eject \( d \) from \( T \) and insert \( e \) into \( T \).
   12. Compute the reverse residual capacities of the darts in \( T[\text{tail}_G(e), \text{tail}_G(d)] \).

Table 2.4: A network-simplex type implementation of the \text{MAXFLOW} algorithm.

The dual tree \( T^* \) is represented by an Euler-tour tree data structure [54], so Steps 8, 9 and 10 can also be implemented in \( O(\log n) \) time. To clarify Step 12: the path \( T[\text{tail}_G(e), \text{tail}_G(d)] \) becomes directed away from \( t \) in Step 11. In order to efficiently compute residual capacities towards \( t \), we must consider the capacities of the \( \text{rev}(T[\text{tail}_G(e), \text{tail}_G(d)]) \).

An iteration of Step 5 is a pivot step and is illustrated in Figure 2.8. To show that \text{MAXFLOW}(G_0, c, s, t) takes \( O(m \log n) \) time, we show that there are at most 3m pivot steps (Theorem 2.3.26). It therefore follows that the algorithm runs in \( O(m \log n) \) time.

First we show that the algorithm does maintain spanning trees of \( G \) and \( G^* \).

**Invariant 2.3.19.** \( T \) is a spanning tree of \( G \) and \( T^* \) is a spanning tree of \( G^* \).

*Proof that the algorithm maintains the invariant.* Step 2 establishes that \( T \) is a spanning tree of \( G \) and Step 4 establishes that \( T^* \) is a spanning tree of \( G^* \), by the Interdigitating Spanning Trees Theorem. Steps 10 and 11 preserve the invariant. \( \Box \)

**Invariant 2.3.20.** If \( d \) is a dart in \( T^* \) such that \( \text{tail}_{G^*}(d) \) is the parent of \( \text{head}_{G^*}(d) \) (i.e., \( d \) is oriented away from \( f_\infty \), the root of \( T^* \) in \( G^* \)), then \( d \) is non-residual.

*Proof that the algorithm maintains the invariant.* First we show that the invariant holds initially. \( T^* \) is composed of arcs not in \( T \). Let \( a \) be any arc not in \( T \). By construction of \( T \), the path of arcs \( a \circ T[\text{head}_G(a)] \) is right of \( T[\text{tail}_G(a)] \). Let \( C = a \circ T[\text{head}_G(a), \text{tail}_G(a)] \); \( C \) is a simple c.c.w. cycle. The face to the left of \( a \) is enclosed by \( C \) and the face to the right is not. In \( G^* \), \( a \) is directed out of \( C \) in \( G^* \). Since \( a \) is the only arc on the boundary of \( C \) that is not in \( T \) and \( f_\infty \) is not enclosed by \( C \), \( \langle a, 1 \rangle \) is directed towards \( f_\infty \) in \( T^* \) and \( \langle a, -1 \rangle \) is oriented away from \( f_\infty \). Since the reverses of arcs have zero capacity, the invariant holds initially.
Figure 2.8: The edges of $T$ are solid, non-tree edges are dashed. $T^*$ is represented by light edges. In an iteration of \textsc{max-flow}, the $s$-to-$t$ path (bold) is saturated and $d$ is the root-most non-residual edge. The shaded face is the face whose parent in $T^*$ changes, $\text{head}_{T^*}(d)$. The parent dart $e$ is removed from $T^*$ and inserted into $T$, and $\text{rev}(d)$ is inserted into $T^*$ and becomes the new parent dart.

Next, note that, in each nonterminating pivot step, $\text{tail}_{T^*}(d)$ is not a descendnet in $T^*$ of $\text{head}_{T^*}(d)$. Dart $e$, the parent of $\text{head}_{T^*}(d)$, is removed from $T^*$. The component of $T^* - \{e\}$ that contains $f_\infty $contains $\text{tail}_{T^*}(d)$ and not $\text{head}_{T^*}(d)$: $d$ is oriented away from $f_\infty $. Since $d$ was saturated in Step 6, $d$ is non-residual. See Figure 2.8 for an illustration of this.

Let $c$ be a dart that remains in $T^*$ during a pivot. The residual capacity of $c$ does not change and the orientation of $c$ in $T^*$ does not change. The invariant holds. □

We say that a dart $d$ is a non-tree dart if $d \notin T$ and $\text{rev}(d) \notin T$.

\textbf{Lemma 2.3.21.} There is no clockwise simple cycle whose non-tree darts are all residual.

\textbf{Proof.} Suppose for a contradiction that $C$ was such a cycle. Let $S$ be the set of non-tree darts in $C$. By Invariant 2.3.19, for every dart $d \in S$, the tree $T^*$ contains either $d$ or $\text{rev}(d)$. Since every dart in $S$ is residual, Invariant 2.3.20 implies that $T^*$ contains every dart in $S$. Since $C$ is clockwise, $\text{head}_{T^*}(d)$ is enclosed by $C$ for every dart $d$ in $C$. Since $T$ is a tree, $C$ contains at least one non-tree dart $d \in S$. The directed path $T^*[\text{head}_{T^*}(d)]$ is completely enclosed by $C$, implying that $C$ also encloses $f_\infty = end(T^*[\text{head}_{T^*}(d)])$, a contradiction. □

We show in the next two corollaries that the network-simplex version of \textsc{max-flow} implements the abstract version. Corollary 2.3.22 shows that the leftmost residual $s$-to-$t$ path is augmented in each iteration and Corollary 2.3.23 shows that the algorithm is correct.

\textbf{Corollary 2.3.22.} For every vertex $v$, there is no residual path strictly left of $T[v]$.

\textbf{Proof.} Suppose for a contradiction that there is a residual path strictly left of $T[v]$. Then the leftmost residual $v$-to-$t$ path, $P$ must be strictly left of $T[v]$. Let $Q$ be a subpath of $P$ such that the endpoints of $Q$ are on $T[v]$ but $Q$ and $\text{rev}(Q)$ are both edge disjoint from $T[v]$. Since $P$ is leftmost, $Q$ is left of $T[end(Q), start(Q)]$. So, $Q \circ \text{rev}(T[end(Q), start(Q)])$ is a simple c.w. cycle whose non-tree darts are residual, contradicting Lemma 2.3.21. □
**Corollary 2.3.23.** The st-flow $f$ returned by the algorithm is maximum.

**Proof.** When the algorithm terminates in Step 7, $\text{tail}_{G^*}(d)$ is a descendent in $T^*$ of $\text{head}_{G^*}(d)$. Let $C$ be the simple cycle $d \circ T^*[\text{head}_{G^*}(d), \text{tail}_{G^*}(d)]$. In the primal $G$, the darts of $C$ form a directed cut $\Gamma_G^+(S)$. Every dart in $C$ except $d$ is a non-tree dart, so the $\text{head}_{G^*}(d)$-to-$t$ path in $T$ does not use any dart in $C$ or the reverse of any dart in $C$. Since $t$ is on the infinite face, $C$ does not enclose $t$ and so $S$ does not contain $t$. Likewise the $s$-to-$\text{tail}_{G^*}(d)$ path in $T$ does not use and dart in $C$ or the reverse of any dart in $C$. Since $d$ crosses $C$, $S$ contains $s$. Since every dart comprising the cut is non-residual, there is no residual $s$-to-$t$ path. By the Max-Flow Min-Cut Theorem, the flow is maximum.

\[\]}

---

**Figure 2.9:** An illustration of Corollary 2.3.23. Darts of the dual tree are dark, with darts of $T^*$ solid. In the dual, $s$ and $t$ are faces, shown shaded. Upon termination, $\text{tail}_{G^*}(d)$ is a descendent in $T^*$ of $\text{head}_{G^*}(d)$. Since $\text{rev}(d)$ is non-residual and the reverses of dual tree darts are non-residual, the cycle shown is a saturated cut separating $s$ from $t$. The darts of this cut (in the primal) are light.

---

We now show that there are at most $3m$ pivot steps in the max-flow algorithm. Let $d$ be a dart. We have the following facts with regards to the max-flow algorithm:

**Fact 1.** If $d$ is residual at time $i$ and non-residual at time $j$ ($i < j$), there was an augmentation including $d$ at some time between $i$ and $j$.

**Fact 2.** If $d$ is non-residual at time $i$ and residual at time $j$ ($i < j$), there was an augmentation including $\text{rev}(d)$ at some time between $i$ and $j$.

**Fact 3.** When it is inserted into $T$, $d$ is residual by Invariant 2.3.20.

**Fact 4.** When it is ejected from $T$, $d$ is non-residual.

**Claim 2.3.24.** A dart $\langle a, 1 \rangle$ where $a$ is an arc of $G_0$ is ejected at most once.

**Proof.** Let $a$ be an arc and let $d = \langle a, 1 \rangle$. Suppose for a contradiction that $a$ is ejected at time $i_1$ and at time $i_2$ ($i_1 < i_2$).

To be ejected at time $i_1$, $d$ must be non-residual by Fact 4. Fact 1 implies that there was an augmentation including $d$ at some time $k_0$ where $0 < k_0 < i_1$.

To be ejected at time $i_2$, $d$ must have been inserted at some time $j_1$ where $i_1 < j_1 < i_2$. At time $j_1$, $d$ is residual by Fact 3. By Fact 2, there was an augmentation including $\text{rev}(d)$ at some time $k_1$ where $i_1 < k_1 < j_1$. 

\[\]
Since there was an augmentation including \( d \) at time \( k_0 \) and there was an augmentation including \( \text{rev}(d) \) at time \( k_1 > k_0 \), \( d \) cannot be augmented after time \( k_1 \) by the Unusability Theorem.

Finally, to be ejected at time \( i_2 \), \( d \) must be non-residual by Fact 4. By Fact 2, there was an augmentation including \( d \) at some time \( k_2 \) where \( j_1 < k_2 < i_2 \). But \( d \) cannot be augmented after time \( k_1 \). This is a contradiction.

**Corollary 2.3.25.** A dart \( \langle a, -1 \rangle \) where \( a \) is an arc of \( G_0 \) is ejected at most twice.

**Proof.** Let \( a \) be an arc and let \( d = \langle a, -1 \rangle \).

Suppose \( d \) is ejected at times \( i_1 \) and \( i_2 \). Then \( d \) must be inserted at time \( i_1 < j_1 < i_2 \). By Fact 4, \( d \) is non-residual at time \( i_1 \) and by Fact 3, \( d \) is residual at time \( j_1 \). By Fact 2, \( \text{rev}(d) \) must be part of an augmentation at some time \( k_1 \) where \( i_1 < k_1 < j_1 \).

Likewise, by Fact 4, \( d \) is non-residual at time \( i_2 \) and by Fact 1 \( d \) must be augmented at time \( k_2 \) where \( j_1 < k_2 < i_2 \).

At time \( i_2 \), \( d \) is out of the tree and non-residual. Since \( \text{rev}(d) \) cannot be augmented after time \( k_2 \) by Claim 2.3.24, \( d \) can never become residual again and so cannot be inserted or ejected again.

As a consequence of the above, we have the following theorem:

**Theorem 2.3.26.** There are at most 3\( mn \) pivot steps in the MAX-FLOW algorithm.

### 2.4 Multiple minimum cuts

In 1961, Gomory and Hu showed that the minimum cut between all pairs of nodes in a general (i.e. not necessarily planar) undirected, weighted graph \( G \) with \( n \) vertices can be represented by a tree \( T \) on the same \( n \) vertices [48]. Further, they showed that \( T \) can be computed using \( n - 1 \) minimum-cut computations (or, equivalently, \( n - 1 \) maximum-flow computations). Such a tree is called a cut-equivalent tree and has the property that for every pair of vertices \( s \) and \( t \), the min \( st \)-cut in \( G \) is equal (in both value and bipartition of the vertices) to the min \( st \)-cut in \( T \). It has long been an open question as to whether one can improve on the Gomory-Hu algorithm by constructing a cut-equivalent tree in time faster than \( O(n F(n, m)) \) where \( F(n, m) \) is the time for a single max-flow (or min-cut) computation on a graph with \( n \) vertices and \( m \) edges.

Until recently, no algorithm has broken the \( O(n F(n, m)) \) bound. In [12], Bhalgat et. al. present an algorithm for the special case of unweighted graphs that runs in \( O(mn \log^2 n) \) time. This improves the best previously-known solution (using Goldberg and Rao’s max-flow algorithm for unit capacity graphs [46]) by \( \Omega(\sqrt{n} / \log^2 n) \). Their solution uses an efficient tree-packing algorithm as a subroutine, as opposed to a max-flow algorithm.

In undirected planar graphs, maximum flow can be computed in \( O(n \log n) \) time (by using the algorithm presented in this thesis restricted to undirected graphs, or by using [55] or [68] to implement [52, 90]). Shiloach [98] first studied the problem of computing a cut-equivalent tree in planar graphs. He demonstrated that a planarity-exploiting maximum-flow algorithm can be used as a subroutine of Gomory and Hu’s algorithm. (It has since been demonstrated that any min-cut producing
algorithm can be used to produce a cut-equivalent tree [49]). Combining these results produces an $O(n^2 \log n)$-time algorithm for constructing a cut-equivalent tree for a planar graph. The algorithm of Bhalgat et. al. for general graphs does not beat this bound.

In this work we consider the special case of finding the min cuts between all pairs of vertices on a common face $f$ of a weighted planar graph. Such cuts can be represented by a (partial) cut-equivalent tree (Figure 2.10). Gomory and Hu’s algorithm can be generalized to this subset version of the problem using $k - 1$ max-flow computations [24] where $k$ is the number of vertices bounding $f$. Using the $st$-planar max-flow algorithm of Hassin [51] (which can be implemented in $O(n)$-time using [55]) as a subroutine of the Gomory-Hu algorithm results in an $O(kn)$-time algorithm. Since $k$ can be as large as $n$, this algorithm may take $\Omega(n^2)$ time.

Figure 2.10: (a) A planar graph $G$ (capacities not shown) with $f$ the infinite face. One half of the bipartition of the min $st$-cut is given by the shaded region. The value of the cut is the sum of the capacities of the edges leaving this region. (b) The tree $T$ (capacities not shown) that is cut-equivalent for the boundary vertices of $G$. The min $st$-cut is highlighted and is given by the minimum-capacity edge on the path in $T$ from $s$ to $t$. This edge has capacity equal to the value of the min $st$-cut in $G$ and the corresponding bipartition of the vertices is the same. For the partial cut-equivalent tree, this property need only hold for the bold vertices. In this work, $T$ turns out to be a spanning tree of $G$.

Here we give an algorithm that constructs the cut-equivalent tree for the boundary vertices of an undirected, weighted planar graph $G$ that runs in $O(n)$ time, regardless of how many vertices there are on the boundary of the graph. The algorithm is inspired by the relationship between minimum cuts and shortest paths via the planar dual (as demonstrated by Hassin [51]). We first find a shortest-path tree $T^*$ of the planar dual rooted at $f$, interpreting capacities as lengths. We show that the dual tree comprised of edges not in $T^*$ forms a cut-equivalent tree $T$ for the vertices in $\partial G$. We then show that from $T$, which has $n$ vertices, we can construct another tree $T'$ that has $k$ vertices (as illustrated in Figure 2.12) and is also cut equivalent (which is the smallest tree representing these cuts).

A cut $X_1$ in graph $G_1$ with capacities $c_1$ is equivalent to a cut $X_2$ in graph $G_2$ with capacities $c_2$ on the same vertex set as $G_1$ if $c_1(X_1) = c_2(X_2)$ and the bipartition of the vertices given by $X_1$ is equal to the bipartition given by $X_2$. For undirected graphs, $\Gamma^+(S)$ for any $S \subseteq V$ is equivalent to $\Gamma^+(\bar{S})$ where $\bar{S} = V \setminus S$, in the same graph with the same capacities. (In particular, a minimum
A $K$-cut-equivalent tree for a graph $G$ is a tree $T$ on the same vertex set as $G$ such that the minimum $st$-cut in $T$ is equivalent to the minimum $st$-cut in $G$ for every pair of vertices $s, t \in K$. We will describe an alternative representation for a $K$-cut-equivalent tree in Section 2.4.4. Traditionally a cut-equivalent tree is for the set $K = V$.

### 2.4.1 Minimum cuts between boundary vertices of a planar graph

In the following we give a construction for a $K$-cut-equivalent tree where $K$ is the set of vertices on the boundary of an input face $f$ of a planar graph. We will drop the prefix 'K-' and will assume that $f$ is the infinite face of the embedding. So, $K$ is the set of vertices on the boundary of the planar graph. We call the vertices of $K$ terminals. The algorithm **BoundaryCutTree** is given in Table 2.4.1.

<table>
<thead>
<tr>
<th><strong>Table 2.5</strong></th>
<th><strong>BoundaryCutTree</strong>($G,c,f$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Let $T^<em>$ be a shortest-path tree, interpreting $c$ as lengths, in $G^</em>$ rooted at $f$. Let $\phi(g)$ be the corresponding $f$-to-$g$ distance for every vertex $g$ of $G^*$.</td>
<td></td>
</tr>
<tr>
<td>2. Let $T$ be the spanning tree of $G$ consisting of all the edges not in $T^*$.</td>
<td></td>
</tr>
<tr>
<td>3. For each dart $d$ of $T$, compute $h(d) = \phi(\text{head}<em>{G^*}(d)) + c(d) + \phi(\text{tail}</em>{G^*}(d))$.</td>
<td></td>
</tr>
<tr>
<td>4. Return $T$ with capacities $h$.</td>
<td></td>
</tr>
</tbody>
</table>

The algorithm **BoundaryCutTree** takes as input an undirected planar embedded graph $G$ with capacities $c$ and a face $f$ and computes a tree that is cut-equivalent for vertices adjacent to $f$. That is, the algorithm implicitly finds all the minimum cuts between vertices adjacent to $f$.

### 2.4.2 Running time of **BoundaryCutTree**

Since the capacities are non-negative, the shortest-path tree needed for Step 1 can be computed in $O(n)$ time using the algorithm of Henzinger et. al. [55]. Steps 2 and 3 can be computed in $O(n)$ time. Overall, the algorithm takes $O(n)$ time.

Note that since $c(\cdot)$ is symmetric (since $G$ is undirected), $h(\cdot)$ is also symmetric: $h(d) = \phi(\text{head}_{G^*}(d)) + c(d) + \phi(\text{tail}_{G^*}(d)) = \phi(\text{tail}_{G^*}(\text{rev}(d))) + c(\text{rev}(d)) + \phi(\text{head}_{G^*}(\text{rev}(d))) = h(\text{rev}(d))$. So, one need only compute $h(d)$ once per pair of oppositely oriented darts. Also, since $T$ will only be used to compute min cuts between boundary vertices of $G$, one need only compute $h(d)$ for darts that are on paths in $T$ between boundary vertices of $G$ (i.e. darts of the minimal subgraph of $T$ that spans the boundary vertices of $G$).

### 2.4.3 Correctness of **BoundaryCutTree**

We prove the correctness of **BoundaryCutTree** as follows. First we show that for two vertices $s$ and $t$ on the boundary of $f$, there is a min $st$-cut that shares exactly one dart with $T[s,t]$ where $T$ is the tree found in Step 2 (Lemma 2.4.1). Next we show that for any dart $d \in T[s,t]$, the min
st-cut constrained to intersect $T[s,t]$ at only $d$ is composed of the edges of the fundamental cycle of $d$ with respect to $T^*$ (Lemma 2.4.2). The correctness of the algorithm follows fairly directly (Theorem 2.4.3).

**Lemma 2.4.1.** Let $G$ be an undirected planar embedded graph with capacities $c$. Let $T^*$ be a $f$-rooted shortest-path tree in the dual $G^*$, interpreting the capacities as lengths. Let $T$ be the spanning tree of $G$ consisting of all the edges not in $T^*$. For every pair of vertices $s$ and $t$ such that $t$ is on the boundary of $f$, there exists a minimum-capacity st-cut $X$ such that $|X \cap T[s,t]| = 1$.

**Proof.** Let $X$ be a min st-cut $\Gamma^+(S)$ such that $s \in S$ and $S$ is minimal. Let $d$ be the first dart of $T[s,t]$ that is in $X$. Let $C$ be the fundamental cycle of $d$ with respect to $T^*$ (i.e. $C = d \circ T^*[head_{G^*}(d), tail_{G^*}(d)]$).

We show that $X$ is enclosed in $C$.

Suppose, for a contradiction, that there is a dart $e$ of $C$ strictly enclosed by $X$ in $G^*$ (recall that $X$ is a cycle in $G^*$). Either $e$ is a dart of $T^*[head_{G^*}(d), f]$ or $e$ is a dart of $T^*[f, tail_{G^*}(d)]$. If $e$ is a dart of $T[head_{G^*}(d), f]$, let $P$ be the maximal subpath of $T[head_{G^*}(d), f]$ such that $e \in P$ and $P$ is strictly enclosed by $X$. If $e$ is in $T[f, tail_{G^*}(d)]$, let $P$ be the maximal subpath of $T[f, tail_{G^*}(d)]$ such that $e \in P$ and $P$ is strictly enclosed by $X$. In both cases, the endpoints of $P$ are on $X$ and both $P$ and $rev(P)$ are shortest paths (since $T^*$ is a shortest-path tree in an undirected graph).

Let $a$ and $b$ be the start- and end-points of $P$, respectively. We create two cycles from $P$ and $X$: $X_1 = X[a, b] \circ P$ and $X_2 = X[a, b] \circ rev(P)$. Since $s$ is a face in $G^*$ that is enclosed by $X$, either $s$ is enclosed by $X_1$ or $X_2$. Suppose, without loss of generality, that $s$ is enclosed by $X_1$. Since $P$ is a shortest path, $c(P) \leq c(X[a,b])$. We have $c(X_1) = c(X[a,b]) + c(P) = c(X) - c(X[a,b]) + c(P) \leq c(X) - c(X[a,b]) + c(X[a,b]) \leq c(X)$. Since $X_1$ is a cycle in $G^*$, it is a cut in $G$ and so, $X_1$ is a min st-cut that enclosed fewer faces than $X$, contradicting the minimality of $X$.

Therefore $X$ is a cycle enclosed by $C$ in $G^*$. Since $d$ was chosen to be the first dart of $T[s,t]$ in $X$, a second dart of $T[s,t]$ cannot be in $X$ without crossing $C$ (which contains only one dart of $T^*$, $d$). The lemma follows.

**Lemma 2.4.2.** Let $G$ be an undirected planar graph with capacities $c$. Let $T^*$ be a $f$-rooted shortest-path tree in the dual $G^*$, interpreting the capacities as lengths. Let $T$ be the spanning tree of $G$ consisting of all the edges not in $T^*$. For any dart $d$ in the path $T[s,t]$, the fundamental cycle of $d$ with respect to $T^*$ is the minimum st-cut whose intersection with $T[s,t]$ is $\{d\}$.

**Proof.** Let $C = T^*[head_{G^*}(d), tail_{G^*}(d)] \circ d$: $C$ is the fundamental cycle with respect to $d$ and $T^*$. Let $S$ be the set of vertices in the component of $T \setminus \{d\}$ that contains $s$: $S$ does not contain $t$. By the Cycle-Cut Duality Theorem, $C = \Gamma^+(S)$ and so $C$ is an st-cut. Since both $s$ and $t$ are adjacent to $f$, $C$ must contain $f$ as a vertex in $G^*$ (that is, $C$ must cut through the face $f$). Thus we can express $C$ as $T^*[f, tail_{G^*}(d)] \circ d \circ T^*[head_{G^*}(d), f]$.

Let $X$ be the minimum st-cut such that $X \cap T[s,t] = \{d\}$. Since both $s$ and $t$ are adjacent to $f$ in $G^*$, $X$ must contain the dual vertex $f$. Since $X$ is a cycle in $G^*$, it follows that $X$ is a simple
cycle of the form $Q \circ d \circ P$ where $P$ is a $\text{head}_G^*(d)$-to-$f$ path and $Q$ is a $f$-to-$\text{tail}_G^*(d)$ path. We have shown that $C$ is a cycle of this form and minimizes the capacity, subject to the constraints.

The correctness of $\text{BoundaryCutTree}$ now follows from the following theorem:

**Theorem 2.4.3.** Let $G$ be an undirected planar graph with capacities $c$. Let $T$ and $h$ be the tree and capacity function returned by $\text{BoundaryCutTree}(G,c,f)$. For two vertices $s$ and $t$ on the boundary of $G$, there is a minimum $st$-cut in $T$ that is equivalent to the minimum $st$-cut in $G$.

**Proof.** For any dart $d$, let $X(d) = T^*[f, \text{tail}_G^*(d)] \cup \{d\} \cup T^*[\text{head}_G^*(d), f]$. By Lemma 2.4.2, $X(d)$ is the minimum $st$-cut whose intersection is $\{d\}$. By Lemma 2.4.1, there is a minimum $st$-cut in $G$ whose intersection with $T[s,t]$ is a single dart. So, there is a dart $d \in T[s,t]$ for which $X(d)$ is the min-$st$ cut in $G$. The dart that achieves this minimizes the capacity of $X(d)$ and is:

$$d_{\min} = \arg \min_{d \in T[s,t]} c(X(d))$$

$$= \arg \min_{d \in T[s,t]} c(T^*[\text{head}_G^*(d), f]) + c(d) + c(T^*[f, \text{tail}_G^*(d)])$$

$$= \arg \min_{d \in T[s,t]} \phi(\text{head}_G^*(d)) + c(d) + \phi(\text{tail}_G^*(d))$$

$$= \arg \min_{d \in T[s,t]} h(d).$$

And so $d_{\min}$ also minimizes $h(d)$ along $T[s,t]$ and is the min $st$-cut in $T$: the value of the minimum $st$-cut in $T$ is equal to the value of the minimum $st$-cut in $G$.

In fact, since $X(d_{\min})$ is the fundamental cycle in $G^*$ corresponding to $d_{\min}$ with respect to $T^*$, the bipartition of the vertices given by $T \setminus \{d_{\min}\}$ is equal to that given by the cut $X(d_{\min})$ in $G$. Therefore the min $st$-cut in $T$ gives the same bipartition of the nodes as the min $st$-cut in $G$. Such a cut is illustrated in Figure 2.11.

![Figure 2.11: A graph $G$ with spanning tree $T$ given by the set of solid edges. The grey cycle is the fundamental cycle in $G^*$ composed of dart $d$ in $G^*$ and the path in $T^*$ between the endpoints of $d$. Since $d$ is on the path in $T$ from $s$ to $t$, the grey cycle is an $st$-cut in $G$.](image)

**2.4.4 A more compact cut tree**

$\text{BoundaryCutTree}$ finds a tree that is cut-equivalent for pairs of vertices on the boundary of the input graph $G$. However, $T$ has $n$ vertices, while $G$ has only $k$ terminals. We would like to find a
tree \( \hat{T} \) with the following properties: \( \hat{T} \) has one node for every vertex on the boundary of \( G \); each node of \( \hat{T} \) is labeled with a subset of the vertices of \( G \), forming a partition of the vertices of \( G \), with the label of a node containing the corresponding terminal of \( G \); given two boundary vertices \( a \) and \( b \) of \( G \) the min \( ab \)-cut in \( G \) should be equivalent to the \( AB \)-cut in \( \hat{T} \) where \( A \) is the node of \( \hat{T} \) whose label contains \( a \) (likewise \( b \in B \), where the bipartition of the vertices is inherited from the labels of the partition of the nodes of \( \hat{T} \)). The result is a \( K \)-cut-equivalent tree and is described in [24].

We review the Gomory-Hu algorithm here. If the terminal set \( K \) has cardinality \( > 1 \), two terminals are chosen \( s, t \) and the min \( st \)-cut \( \Gamma^+(S) \) is computed. The algorithm proceeds recursively on the graph obtained by identifying the vertices in \( S \) with terminal set \( K \cap (V - S) \) and on the graph obtained by identifying the vertices \( V - S \) with terminal set \( K \cap S \). The cuts found are enough to construct a cut-equivalent tree for the vertices.

Given the tree \( T \) with capacities \( h \) output by \textsc{BoundaryCutTree}, we give an \( O(n) \) implementation of the Gomory-Hu algorithm to find a \( K \)-cut-equivalent tree where \( K \) is the set of terminals (boundary vertices). In the following, \( \pi \) is the cyclic ordering of the terminals according to the first time the terminal appears in an Euler tour of \( T \) starting from a terminal \( s \) (that is, for \( t \in K \), \( \pi(t) \) is the next terminal in this Euler-tour ordering of \( K \)). Initially, \( r = s \), and \( g \) is a null function. To restrict \( \pi \) to a subtree means to restrict \( \pi \) to the sub-ordering of the terminals of \( K \) that appear in that the subtree.

\[
\text{GomoryHuEdges}(T, h, \pi, s, r, g):
\]

1. For every vertex \( x \) on \( T[r, \pi(s)] \), compute \( g(x) = \arg\min_{d \in T[s,x]} h(d) \).
2. Let \( d = g(\pi(s)) \).
3. Let \( r \) be the last vertex of \( T[s, d] \) that has degree \( > 2 \) in \( T \) or is a terminal.
4. Let \( T_1 \) and \( T_2 \) be the components of \( T - d \) such that \( r \in V(T_1) \).
5. If \( T_1 \) has more than one terminal,
   \[ M_1 = \text{GomoryHuEdges}(T_1, h, \pi \text{ restricted to } T_1, s, r, g) \]
6. If \( T_2 \) has more than one terminal,
   \[ M_2 = \text{GomoryHuEdges}(T_2, h, \pi \text{ restricted to } T_2, \pi(s), \pi(s), \text{null}) \]
7. Return \( M_1 \cup M_2 \cup \{d\} \).

Table 2.6: The \text{GomoryHuEdges} algorithm finds a \( K \)-cut-equivalent tree when the input is a tree.

First we argue that \text{GomoryHuEdges} finds the cuts that a Gomory-Hu algorithm would. In step 1, the min \( s\pi(s) \)-cut is computed: \( d \). Deleting \( d \) creates two subproblems (Step 4). Step 3 guarantees that \( r \) will be a vertex of \( T[s, \pi(s)] \) for the subproblem in Step 5. Therefore, upon calling \text{GomoryHuEdges}, the function \( g \) satisfies that for every vertex \( x \) on \( T[s, r] \), \( g(x) = \arg\min_{d \in T[s,x]} h(d) \). It follows that \text{GomoryHuEdges} correctly implements the Gomory-Hu algorithm.

The function \( g \) stores partial min cuts and allows us to achieve an \( O(n) \) running time. Step 1 is implemented so that evaluating \( g(x) \) takes one comparison for each \( x \): between \( h(e) \) for the last dart of \( T[s, x] \) and \( g(\text{tail}_T(d)) \). It is not hard to see that each dart \( e \) appears only once in such a comparison. Since there are \( O(n) \) edges in a planar graph, there are \( O(n) \) such comparisons in \text{GomoryHuEdges}. 

Table 2.6: The \text{GomoryHuEdges} algorithm finds a \( K \)-cut-equivalent tree when the input is a tree.
Let $M$ be the set of darts output by \textsc{GomoryHuEdges}. Contract to a vertex $v$ every maximal subtree $H$ of $T$ that does not contain a dart of $M$ and label $v$ with vertex set of $H$. The resulting tree $\hat{T}$ is the $K$-cut-equivalent tree we desire. It is illustrated in Figure 2.12.

Figure 2.12: The tree $T$ given by solid edges is that returned by \textsc{BoundaryCutTree} on the graph $G$ with boundary given by dotted lines. The edges of $M$ found by \textsc{GomoryHuEdges} are highlighted in grey. The nodes of a compact cut tree $T''$ for the boundary vertices of $G$ are given by the large grey circles and are labeled with the vertices of $G$ they contain. The edges of $T''$ are exactly the edges found by \textsc{GomoryHuEdges}.

### 2.5 Open Problems

The \textsc{BoundaryCutTree} algorithm represents the first step towards an algorithm for the all-pairs min-cut problem in undirected, weighted planar graphs (that is, building the cut-equivalent tree for all the vertices of a planar graph) that would beat the $O(nF(n,m))$ bound given by the Gomory-Hu algorithm. This bound, in planar graphs, is $O(n^2 \log n)$. It isn’t immediately obvious how the algorithm seen here could result in, say, a sub-quadratic bound for the all-pairs problem. However the algorithm for representing all the boundary-to-boundary distances of a planar graph presented by Klein in 2005 [68] was later used for efficiently computing distances between an arbitrary set of vertex pairs [20] by using balanced separators. The algorithm given here could prove similarly useful.

Turn now to the maximum-flow problem. While the algorithm given in this thesis seems to close the book on a long line of research, it is certainly not the most general of maximum-flow algorithms for planar graphs. Our algorithm is only guaranteed for a single source and a single sink. In general graphs, this is no limitation as one can turn a multiple-source, multiple-sink flow problem into an $st$-flow problem: connect a new vertex $s$ to each of the sources with infinite-capacity edges and repeat for the sinks. However, as Miller and Naor point out [79], planarity is not preserved by this reduction. They give a planarity-exploiting algorithm for a similar problem in which the demands on the sources and sinks are known a priori.

Consider the maximum-flow problem with multiple sources and a single sink. While this problem is not completely general, an efficient solution would solve the image segmentation problem as defined by Cox, Rao, and Zhong [25] more efficiently [87]. Although the Unusability Theorem for \textsc{MaxFlow}
depends on the existence of a single source (i.e. Property 2.3.12), the algorithm does not. Certainly, the preprocessing step does not depend on the sources or sinks: clockwise cycles can be saturated. For each source, the primal spanning tree maintains a leftmost source-to-sink path. The natural extension of MAX-FLOW saturates one of these leftmost residual paths in each iteration. There is freedom in the choice of sources: in order to maintain a leftmost flow, it does not matter which source is chosen, only that the augmentation is a leftmost augmentation from that source. Unfortunately, we have counterexamples to the Unusability Theorem for various choices of source (including leftmost, rightmost, and any fixed order) for each iteration.

In 2005, Klein presented an algorithm that computes the shortest path tree rooted at each vertex on a common face [68]. Klein’s algorithm and MAXFLOW use similar ideas. In Klein’s algorithm, a preprocessing step computes a leftmost shortest-path tree rooted at one such vertex \( u \). (Klein instead used rightmost.) This is analogous to our finding a leftmost or clockwise-saturating circulation. The tree is re-rooted at the next vertex \( v \) along the boundary of the face. A pivoting step changes the tree to reduce the distance labels while maintaining that the distance to every vertex in the graph is the shortest given by any path to the left of the tree path to that vertex. This is analogous to our maintaining a leftmost flow. When the current tree becomes a shortest-path tree, the tree is re-rooted at the next vertex along the boundary of the face. Klein shows that the algorithm uses \( O(n) \) pivot steps and implements each pivot step using primal and dual spanning trees represented by a dynamic-tree data structure. A similar concept has also been used to design algorithms for finding edge- and vertex-disjoint paths in planar graphs [92].

In these algorithms, the planar graph is searched in a consistent manner. In MAXFLOW, the augmenting path moves from left to right. This is particularly obvious when both the source and sink are on the infinite face. In the multiple-source shortest-path algorithm, described above, the tree moves from left to right. In both cases, the current solution is optimal when considering solutions to the left of the current solution. This is analogous to the plane-sweep technique of computational geometry. In two dimensions, a line is swept through the plane of data from left to right maintaining optimal solution to the problem considering only the data to the left of the line. Unifying the analysis of MAXFLOW and the multiple-source shortest-path algorithm could lead to a general plane-sweep technique for planar graphs. The MAXFLOW and multiple-source shortest-path algorithms have a network simplex interpretation. The primal tree that is maintained throughout the algorithm corresponds to a basic solution; the pivot step corresponds to a pivot step in the simplex algorithm. Problems that have a natural network simplex interpretation are obvious starting points for such a generalization.
Chapter 3

Connectivity Problems

The Steiner-tree problem is a classic problem in combinatorial optimization that started with a geometry problem. Fermat is said to have asked Torricelli what point in the Euclidean plane minimizes the sum of the distances to three given points. Torricelli solved the problem exactly [106]. Later, Steiner gave a significantly more elegant construction [101], and the problem was attributed to him. More generally, metric Steiner-tree problem is to find a tree spanning a given set of points in the metric space where additional (Steiner) points may be introduced in order to minimize the total length of the tree. In graphs, given a subset of the vertices called terminals of the graph, the Steiner-tree problem is to find a minimum weight subgraph that connects that subset.

In some cases, one might require more than a single path connecting each pair of terminals. Most generally, the goal is to find a subgraph $H$ that satisfies a set of connectivity requirements, $r(x)$ for each vertex $x$. For a pair of vertices $x$ and $y$, $H$ satisfies the connectivity requirement between $x$ and $y$ if there are $\min\{r(x), r(y)\}$ disjoint $x$-to-$y$ paths in $H$. Disjoint may have one of two interpretations: the paths can either be edge disjoint or vertex disjoint (not considering, of course, the endpoints of the paths). If paths are vertex disjoint, then they are also edge disjoint. Connectivity problems for $r > 1$ are often called survivable network problems as the network is resistant to disconnection if (in the edge-connected case) an edge or (in the vertex-connected case) a vertex is removed. The problem arises in telecommunication network design where an important requirement is the network’s resilience to link failures [91].

In this work we consider $\{0, 1, 2\}$ requirements in undirected, planar graphs: $r(x) \in \{0, 1, 2\}$ for every terminal $x$. The Steiner-tree problem is a special case of this problem. We consider a standard relaxation of the problem where the solution $H$ is a multi-subgraph of the input graph $G$. That is, an edge $e$ of $G$ may be counted multiple times in $H$. In this case, the weight of edge $e$ is also counted multiple times in the weight of $H$. We summarize the problems here:

**Steiner-tree problem** Given a set of terminals $Q$, find a minimum-weight tree $T$ of $G$ that spans $Q$ ($r(x) = 1$ for every vertex $x \in Q$).

**2-edge-connectivity (2-EC) problem** Given a set of requirements on the vertices, $r(x) \in \{0, 1, 2\}$,
find a minimum-weight multi-subgraph $H$ of $G$ that contains at least $\min\{r(x), r(y)\}$ edge-disjoint paths between every pair $x,y$ of terminals.

We call the set of vertices with non-zero requirements the set of terminals. The 2-EC problem contains, as a special case, the Steiner-tree problem. It also contains the special case of finding a multi-subgraph that achieves 2-edge-connectivity between a set of terminal vertices (i.e. $r(x) \in \{0, 2\}$). Note that it is sufficient to allow at most 2 copies of any edge for the 2-EC problem, so $H$ is a bi-subgraph of $G$. In the following, we will denote by $\text{OPT}_1(G, Q)$ and $\text{OPT}_2(G, Q)$ the weight of the optimal solution to the Steiner-tree and 2-EC problem, respectively. When we omit the subscript, the statement is true for both problems.

### 3.1 Algorithms for connectivity problems

Most versions of connectivity problems are NP-complete. The weighted network Steiner-tree problem (in general graphs) was one of Karp’s original 21 NP-hard problems [63]. So, unless P=NP, there is no polynomial-time algorithm for computing an optimal solution. In fact, even when the input is restricted to be planar, the Steiner-tree problem is NP-complete (by a reduction from the NP-completeness of the rectilinear Steiner-tree problem [44]). The 2-EC problem was shown to be NP-hard in complete graphs with arbitrary weights by Eswaran and Tarjan [36]. It is NP-hard in planar graphs (by a reduction from the Hamiltonian cycle problem).

The goal, then, of algorithms researchers is to find approximation algorithms. For the Steiner-tree problem, the first was a 2-approximation due to Takahashi and Matsuyama [102] and Kou, Markowsky and Berman [73]. Their running time was improved by Wu, Widmayer, and Wong [112], Widmayer [110], and Mehlhorn [78]. The approximation ratio has been improved by Zelikovsky, Berman, Ramaiyer, Romel, Steger, Karpinsky, Hougardy, and Promèl [8, 57, 64, 84, 114, 115], leading to a 1.55-approximation by Robins and Zelikovsky [93]. For the problem of finding a subgraph containing 2 edge-disjoint paths between every pair of terminals, there is a 2-approximation algorithm due to Frederickson and Járá [43] (the running time was improved by Khuller and Thurimella [66]). This is the best known approximation ratio for weighted graphs. For unweighted graphs, Kuller and Thurimella gave a 1.5-approximation that was later improved to a 5/4-approximation [62]. For the non-spanning case, Ravi [89] showed that Frederickson and Járás’s algorithm could be generalized to give a 3-approximation. Klein and Ravi [70] gave a 2-approximation for a more general problem that specifies which pairs of nodes must be connected. This result was greatly generalized by Williamson et. al. [111], Goemans et. al. [45] and Jain [59]. All of these algorithms are for general, undirected graphs and do not require duplicate edges.

### 3.1.1 Polynomial-time approximation schemes

One may consider the *polynomial-time approximation scheme* or PTAS the best one can do for an NP-hard problem. A PTAS is a family of algorithms such that for any given $\epsilon > 0$, the corresponding
Algorithm runs in polynomial time and, for any input, returns a solution whose value is within a $1 + \epsilon$ factor of the optimal solution.

Unfortunately, it has been shown that the network Steiner-tree problem is max-SNP complete [11, 105]: it is NP-hard to find a Steiner tree of weight at most $96/95$ times optimal if $RP \neq NP$ [23]. Similarly, the 2-EC problem is max-SNP complete [27]: it is NP-hard to find a 2-edge-connected spanning subgraph of a degree-3 graph to within $1573/1572$ of optimal [26].

However, we show that this intractability can be overcome by restricting the input to be planar by giving a PTAS for the Steiner tree [19, 15] and 2-EC problems. Further, we give an efficient PTAS (or EPTAS) for these problems: the degree of the polynomial in the running time does not depend on $\epsilon$. Our running times are of the form $O(2^{poly(1/\epsilon)} n + n \log n) = O(n \log n)$.

Previously no PTAS was known for the Steiner-tree problem in planar graphs. However, a PTAS has been given for the related problem in low-dimensional Euclidean space. The first algorithms used a recursive quad-tree decomposition of the space [4, 80] such that there is a near-optimal solution that crosses each region of the quad-tree a constant number of times. This is analogous to the Structure Theorem we give (Section 3.4). The algorithm due to Arora [4] is near-linear in $n$ with a polylog factor whose degree depends on $\epsilon$. Rao and Smith [88] gave an $O(n \log n)$ approximation scheme using the concept of a spanner, a concept we also use to derive our algorithms. In this Euclidean case, $n$ denotes the number of terminals.

No PTAS has been given for the subset 2-EC problem we consider here (i.e. the set of terminals $Q$ is a strict subset of the vertices of the input graph). Berger et. al. [6] gave a PTAS for the spanning case ($Q = V$) in unweighted, planar graphs when duplicate edges are allowed. Berger and Grigni [7] extended this result to handle weighted, planar graphs (still for the spanning case), allowing and not allowing edge-duplication. The PTAS is not an EPTAS. A PTAS is also known for the corresponding geometric problem in complete geometric graphs of low dimension [27].

### 3.1.2 Polynomial-time solvable cases

**Theorem 3.1.1.** [35] Let $G$ be a weighted planar embedded graph and let $Q$ be a set of $k$ terminals that all lie on the boundary of a single face. Then there is an algorithm to find an optimal Steiner tree of $G$ spanning $Q$ in time $O(nk^3 + (n \log n)k^2)$.

This algorithm has been generalized by Bern [9] and by Bern and Bienstock [10] to handle some additional special cases, e.g. where the terminals lie on a constant number of faces. Provan [86, 85] used the same approach to give exact and approximate algorithms for some geometric special cases. The algorithm of [35] uses as a subroutine an algorithm for computing single-source shortest paths. Using instead as a subroutine the linear-time planarity-exploiting algorithm of [55], one can improve the running time to $O(nk^3)$. We will use this algorithm to solve both the Steiner-tree and 2-EC problems.

We give an equivalent algorithm for the 2-EC problem (Section 3.8):
**Theorem 3.1.2.** Let $G$ be a weighted planar embedded graph and let $Q$ be a set of $k$ terminals that all lie on the boundary of a single face. Then there is an algorithm to find an optimal multi-subgraph of $G$ satisfying requirements $r(x) \in \{0, 1, 2\}$ for every vertex $x \in Q$. The running time is $O(k^3n)$. For the case when $r(x) \in \{0, 2\}$, the running time is linear.

As with many NP-hard problems, the Steiner-tree and 2-EC problems can be solved in polynomial time (in fact in linear time) in graphs of bounded treewidth given a tree-decomposition of bounded width $w$. An optimal Steiner tree can be computed in $O(w^n)$-time [71]. An optimal solution to the 2-EC problem can be found in $O(2^{O(w^2)}n)$-time [7] (this algorithm solves the spanning case, but can be easily modified to solve the subset version).

### 3.1.3 New approximation schemes for planar connectivity problems

We give two $O(n \log n)$-time approximation schemes for each Steiner tree and 2-EC. Both rely on a graph decomposition we call a *brick decomposition*. The brick decomposition is a grid-like subgraph that spans the input terminals with weight at most $O(\text{OPT})$ where $\text{OPT}$ is the weight of the optimal solution (Section 3.2). We then show that for both the Steiner-tree and 2-EC problem, the optimal solution crosses each face of the brick decomposition a small number of times. This property is given by the main Structure Theorem, Theorem 3.4.4 in Section 3.4. The Structure Theorem is the main tool for giving a PTAS.

The first approximation scheme computes, for each face of the brick decomposition, optimal solutions within each face where the terminals are assumed to be on the boundary of the face. We show that the union of the brick solutions together with the brick decomposition edges forms a *light spanner*: a subgraph whose weight is $O(\text{OPT})$ that contains a near-optimal solution to the original problem. The details of the framework are given in Section 3.5. This use of a spanner is similar to the algorithm for Steiner tree in the Euclidean plane due to Rao and Smith [88]. This scheme has doubly-exponential dependence on $1/\epsilon$.

The second approximation scheme has singly-exponential dependence on $1/\epsilon$. Here a second level of decomposition is found that breaks the brick decomposition into *parcels*, each of which has (something close to) bounded carving width [97]. Based on a carving decomposition of each parcel, a near-optimal solution is found using portals to limit the interaction between faces of the brick decomposition. The details are given in Section 3.6. The use of portals and the brick decomposition is similar to Arora’s approximation schemes for problems in the Euclidean plane [4].

In both algorithms we will actually construct a $(1 + c\epsilon)$-approximate solution for a constant $c$ fixed by the analysis (and not dependent on $\epsilon$). In order to achieve a $(1 + \epsilon)$-approximation, one should use $\epsilon/c$ as the precision. We will (not unreasonably) assume $\epsilon$ is in the range $(0, 1)$.

### 3.2 Brick decomposition

In this section, we give first the construction for the graph decomposition that we call a *brick decomposition*. The brick decomposition is defined by a subgraph of the input graph called a *mortar*
graph, each face of which defines a brick. In Section 3.4.2, we will show that there is a near-optimal solution that crosses the boundary of each brick a small number of times.

### 3.2.1 Construction of the mortar graph

The mortar graph, denoted $MG$, is a connected, grid-like subgraph of an input graph $G$ (Figure 3.1).

![Figure 3.1: (a) An input graph $G$ with bold edges forming the mortar graph $MG$ in (b).](image)

The construction of the mortar graph depends on the terminal set $Q$ and the precision $\epsilon$. The algorithm BrickDecomposition for finding $MG$ is given in Table 3.1. Steps 1 through 5 of the algorithm are identical to the first three steps in Klein’s construction [69] of a subset spanner.

We require the following definition for the algorithm:

**Definition 3.2.1 ($\epsilon$-short).** A path $P$ in a graph $G$ is $\epsilon$-short if for every pair of vertices $x$ and $y$ on $P$, the distance from $x$ to $y$ along $P$ is at most $(1 + \epsilon)$ times the distance from $x$ to $y$ in $G$:

$$\text{dist}_P(x, y) \leq (1 + \epsilon)\text{dist}_G(x, y).$$

![Figure 3.2: (a) The first strip is created by a path (dashed) connecting $x$ to $y$. The distance between every pair of vertices, $x'$ and $y'$, between $x$ and $y$ on the boundary is well approximated by the boundary distance. We recurse on the shaded face. (b) A graph is divided into strips (by the dashed lines). One strip is shaded and enlarged in (c). Columns (vertical lines) are taken from the set of shortest paths from the lower, south boundary $S$ (dashed) to the upper, north boundary $N$ (solid).](image)

Note that in Step 3, such vertices always exist because for $x = y$, $H[x, y]$ is equal to $H$ and is not an $\epsilon$-short path. This step is illustrated in Figures 3.2(a) and (b). The path $N$ (a shortest path) is called the north boundary of the strip. The path $H[x, y]$ (whose interior is an $\epsilon$-short path) is called the south boundary of the strip (denoted $S$). We embed a strip so that $S$ is below $N$ and we take $s_0$ to be the left endpoint common to $S$ and $N$. By convention column $C_0$ is defined to be the (empty)
BrickDecomposition(G,Q,\epsilon):

1. Find a 2-approximate Steiner-tree T spanning Q in G.
2. Let G' = TREE CUT(G,T). Interpret the new face as the infinite face f_\infty with counterclockwise boundary H.
3. Find vertices x, y on H such that H[x,y] is a minimal subpath of H that is not \epsilon-short. Let N be a shortest path from x to y in G': the subgraph enclosed by H[x,y] \cup N is called a strip and we denote H[x,y] by S. Recursively decompose the subgraph of G' enclosed by N \cup H[y,x] into strips (if this subgraph is nontrivial).
4. For each strip embedded with N above S and vertices ordered from left to right along S:
5. For i = 1, 2, ..., find the first vertex s_i on S (in left-to-right order) such that the distance from s_{i-1} to s_i along S is greater than \epsilon times the distance from s_i to N in the strip: dist_S(s_{i-1}, s_i) > \epsilon dist_{strip}(s_i, N). Column C_i is defined to be the shortest path in the strip from s_i to N.
6. Let
\[ \kappa = \kappa(\epsilon) = 4\epsilon^{-2}(1 + \epsilon^{-1}), \] (3.1)
and let C_i = C_i \cup C_{i+\kappa} \cup C_{i+2\kappa} \cup ... for i \in \{0, 1, ..., \kappa - 1\}. Let \iota^* be the index that minimizes \ell(C_i). We designate the columns in C_{\iota^*} as supercolumns.
7. Return the edges belonging to T, the strip boundaries and supercolumns.

Table 3.1: The algorithm for constructing the mortar graph, a subgraph of G that depends on an input set of terminals Q and a given precision, \epsilon > 0.

shortest path from s_0 to N. We also include as a column the trivial path starting and ending at the rightmost vertex common to S and N. Columns are illustrated in Figure 3.2(c).

The mortar graph is composed of the edges of the 2-approximate Steiner tree T (Step 1), the strip boundaries (Step 3), and the supercolumns (Step 6). Each face of the mortar graph is bounded by two supercolumns and a northern and southern strip boundary. These faces will define the bricks (Section 3.3).

3.2.2 Running time of BrickDecomposition

Step 1 can be done in O(n \log n) time [78, 110, 112]. In planar graphs, Step 1 can be computed in linear time [104]. Klein [69] shows that the strip decomposition of an n-vertex planar graph can be found in O(n \log n) time using [68] and describes how to reduce the problem of finding the columns in a strip to a single shortest-path computation in an augmented strip. It is therefore possible to find all the columns in O(n) time. The entire construction takes O(n \log n) time.

3.2.3 Properties of the mortar graph

We upper-bound the total weight of the mortar graph, including various components of the mortar graph, in terms of the length of the optimal solution. The following was given by Klein in [69] as \ell(H) \leq 4 OPT_1(G, Q). Since a solution to the 2-EC problem is also a solution to the Steiner-tree
Lemma 3.2.2. $\ell(H) \leq 4\text{OPT}(G, Q)$.

Lemma 3.2.3 (Inequality (10), Klein [69]). The total length of all the boundary edges of all the strips is at most $(\epsilon^{-1} + 1)$ times the length of $H$.

Lemma 3.2.4 (Lemma 5.2, Klein [69]). The sum of the lengths of the columns in a strip is at most $\epsilon^{-1}\ell(S)$ where $S$ is the southern boundary of the strip.

Lemma 3.2.5. The sum of lengths of the supercolumns in a strips is at most $1/\kappa$ times the sum of the lengths of the columns in the strip.

The following lemma show that the sum of the weights of the supercolumns is an $\epsilon$ fraction of OPT. We will assume that our optimal solution includes all the edges of the supercolumns. This will allow us to concentrate on maintaining connectivity between the northern and southern boundaries of bricks.

Lemma 3.2.6. The sum over the strips of the lengths of all the supercolumns is at most $\epsilon\text{OPT}(G, Q)$.

Proof. Combine Lemmas 3.2.2, 3.2.3, 3.2.4, 3.2.5 and Equation (3.1).

The next two lemmas are needed for the approximation schemes we give in Sections 3.5 and 3.6.

Lemma 3.2.7 (Terminal Property). The mortar graph spans $Q$.

Proof. This is clear from Step 1 of BRICKDECOMPOSITION. In fact every terminal is a vertex of a strip boundary.

Lemma 3.2.8 (Mortar-Graph Length Property). The length of MG is at most $5\epsilon^{-1}\text{OPT}(G, Q)$.

Proof. From Lemma 3.2.2 and Lemma 3.2.3, we have that the lengths of the strip boundaries is at most $4(\epsilon^{-1} + 1)\text{OPT}(G, Q)$. From Lemma 3.2.6, the lengths of the supercolumns is at most $\epsilon\text{OPT}(r_i)$. Adding those quantities yields

$$\ell(MG) \leq 4(1 + \epsilon + \epsilon^2/4)\epsilon^{-1}\text{OPT}(G, Q),$$

hence the Lemma.

3.3 Structural properties of bricks

The mortar graph defines a set of subgraphs of the input graph called bricks. Each brick corresponds to a face of the mortar graph (Figure 3.3).

The decomposition into bricks will prove very useful. In this section we define the set of bricks given by the mortar graph (Lemma 3.3.1) and then state a structural property about the bricks.
Figure 3.3: Given an input graph (a), the faces of the mortar graph (b) define the set of bricks (c).

(Theorem 3.3.3) concerning forests embedding in bricks. We prove this theorem in stages, first proving three lemmas about trees with leaves on ε-short paths (Section 3.3.2) and finally proving Theorem 3.3.3 in Section 3.3.3. This theorem will apply directly to the Steiner-tree problem. However, for the 2-EC problem, we need to extend the theorem to handle 2-connectivity. We do this in Section 3.3.4.

3.3.1 Bricks

Consider the mortar graph edges in the cut open graph (G’ of Step 2). Each face f of the mortar graph that strictly encloses at least one edge of G’ defines a graph called a brick. In G, the brick consists of the edges of G that are enclosed by the boundary ∂f of f. This boundary is a cycle of edges, possibly with repetition if some edges occur twice in the boundary. (An example of such a situation is shown on Figure 3.4. Note that in G’ each ∂f is a simple cycle by construction, but in G, this might not be the case.) In G’, we duplicate edges as follows:

Cut the original graph G_{in} along ∂f, duplicating the edges you cut along (and replicating the vertices) as in the operation TREECUT, and define the brick to be the subgraph of G_{in} embedded inside that cycle, including the boundary edges according to their multiplicity in ∂f. That is, if an edge occurs twice in the boundary of the face, then there are two copies of that edge in the corresponding brick.

It is easy to see that computing the set of bricks takes O(n) time.

The boundary ∂B of a brick B is the simple cycle of boundary edges. The corresponding face of MG is called the mortar boundary of B. Each edge of the mortar graph occurs at most twice in the disjoint union of the boundaries of the bricks. Since we defined bricks corresponding only to non-empty faces, every brick contains at least one edge not belonging to MG. Figure 3.3(c) is an example of the set of bricks corresponding to the mortar graph of Figure 3.3(b). The construction of a brick is illustrated in Figures 3.4(a) and (b).

Each brick B is bounded by subpaths of the northern and southern boundaries of a strip (N_B and S_B, respectively) and two supercolumns E_B and W_B (east and west). The construction of MG implies the following lemma, which summarizes the properties of a brick.

**Lemma 3.3.1.** The boundary of a brick B, in counterclockwise order, is the concatenation of four paths W_B ∪ S_B ∪ E_B ∪ N_B such that:
Figure 3.4: Construction of a brick: (a) The boundary of a face $f$ of $MG$ is a cycle of edges (thick edges), possibly with repetition (i.e. an edge can occur twice in the boundary). The light edges are those in the interior of $f$ in $G_{in}$. (b) We obtain the corresponding brick by cutting open the graph along the boundary of $f$. The resulting brick $B$ has boundary $\partial B$.

(B1) The set of edges $B \setminus \partial B$ is nonempty.

(B2) Every terminal of $Q$ that is in $B$ is on $N_B$ or on $S_B$.

(B3) $N_B$ and $S_B$ are $\epsilon$-short.

(B4) There exists a number $k \leq \kappa$ and vertices (called column bases) $s_0, s_1, s_2, \ldots, s_k$ ordered from left to right along $S_B$ such that $\dist_{S_B}(x, s_i) < \epsilon \dist_B(x, N_B)$ where $x$ is a vertex of $S_B(s_i, s_{i+1})$. That is, for any vertex $x \in S(s_i, s_{i+1})$, the distance from $x$ to $s_i$ along $S_B$ is less than $\epsilon$ times the distance from $x$ to a $N_B$ in $B$.

$\kappa$ is defined by Equation (3.1) in Table 3.1.

We now state a theorem that shows the existence of a near-optimal Steiner tree that joins the boundary of each brick a small number of times.

**Definition 3.3.2** (Joining vertex). Let $H$ be a subgraph of a graph $G$ and let $P$ be a path in $G$. A joining vertex of $H$ with $P$ is a vertex of $P$ that is the endpoint of an edge of $H \setminus P$.

The intersection of a tree with a brick may not be connected, and so the theorem applies to forests inside bricks. This theorem is a key ingredient to the proof of correctness of the PTAS.

**Theorem 3.3.3** (Structural Property of Bricks). Let $B$ be a plane graph with boundary $N \cup E \cup S \cup W$ satisfying the brick properties of Lemma 3.3.1. Let $F$ be a subgraph of $B$. There is a forest $\tilde{F}$ of $B$ with the following properties:

(F1) If two vertices of $N \cup S$ are connected in $F$, then they are connected in $\tilde{F}$.

(F2) The number of joining vertices of $\tilde{F}$ with both $N$ and $S$ is at most $\alpha(\epsilon)$.

(F3) $\ell(\tilde{F}) \leq (1 + c\epsilon)\ell(F)$.

In the above, $\alpha(\epsilon) = o(\epsilon^{-5.5})$ and $c$ is a fixed constant.

Recall that the sum of the weights of the supercolumns is at most $\epsilon\OPT$. This allows us to include them in a near-optimal solution: the intersection of this near-optimal solution with a brick
is a forest with leaves only on the northern and southern boundaries of the brick. The connectivity between these vertices is guaranteed by the Structural Property of Bricks.

We sketch the proof of this theorem. Each brick $B$ has at most $\kappa$ column bases along the southern boundary. For each column base $s_i$, we find a south-to-north path $P_i$ in $F$. $P_i$ starts at the last vertex on $S_B(s_{i-1}, s_i)$ that is in $F$ and ends at a vertex of $N_B$. By the property of column bases, $S_B(s_{i-1}, start(P_i))$ is an $\epsilon$ fraction of the length of $P_i$. We will add these subpaths of $S_B$ to $F$. This will turn $F$ into a set of $O(\kappa)$ rooted trees, each of which has leaves on either $N_B$ or $S_B$, but not both (Section 3.3.3). We will replace each tree $T$ with another, simpler tree $T'$ that spans the leaves of $T$ (in order to maintain the connectivity of $F$). We will additionally require $T'$ to span the root of $T$. In some cases, we will need $T'$ to span an additional node of $T$ (a second ‘root’, so to speak) in order to maintain the connectivity in $F$. We will show various ways of simplifying trees in Section 3.3.2.

We will give a counterpart to the Structural Property of Bricks Theorem for the 2-EC problem in Section 3.3.4.

The Structural Property of Bricks Theorem is used to prove our main Structure Theorem (Section 3.4) which essentially states:

There is a near-optimal solution to the problem that crosses the boundary of each face of the mortar graph $O(1)$ times.

The Structure Theorem applies to both the Steiner-tree and 2-EC problems. We will solve a subproblem for each face of the mortar graph. The small number of crossings allows us to limit the interaction between these subproblems, forcing our solution to use a small number of identified portal vertices (Section 3.4.1) on the boundary of each face of the mortar graph.

### 3.3.2 Simplifying trees with leaves on $\epsilon$-short paths

In this self-contained section, we establish a few combinatorial lemmas that simplify trees whose leaves lie on an $\epsilon$-short path. These lemmas will be used in Section 3.3.3.

**Lemma 3.3.4.** Let $T$ be a tree all of whose leaves lie on an $\epsilon$-short path $P$. There is a subpath of $P$ spanning the vertices of $T \cap P$ whose total length is at most $(1 + \epsilon)\ell(T)$.

**Proof.** Let $P'$ be the shortest subpath of $P$ that spans all the vertices of $T \cap P$. There is a path $Q$ in $T$ between the endpoints of $P'$. Since $P$ is $\epsilon$-short, $\ell(P') < (1 + \epsilon)\ell(Q) \leq (1 + \epsilon)\ell(T)$. \qed

**Lemma 3.3.5.** Let $T$ be an $r$-rooted tree in graph $G$ with leaves all on an $\epsilon$-short path $P$. There is a binary $r$-rooted tree that has length at most $(1 + \epsilon)\ell(T)$ and spans all the vertices of $T \cap P$.

**Proof.** Let $u$ be a root-most vertex of $T$ with at least three children and let $T_u$ be the subtree of $T$ rooted at $u$. Let $\overline{T_u}$ be the tree consisting of the minimum subpath $P'$ of $P$ that spans all the leaves of $T_u$ and the shortest $u$-to-$P$ path $Q'$ (Figure 3.5).
Figure 3.5: The replacement for a tree $T_u$ (shown as a shaded triangle) rooted at $u$ consists of the minimum subpath $P'$ of $P$ that spans the leaves of $T_u$ and the shortest $u$-to-$P'$ path, $Q'$. These paths are shown in bold.

Since $u$ has at least three children in $T$, it must be that $T_u$ contains three disjoint paths from $u$ to $P'$, including paths $Q_1$ and $Q_2$ to the endpoints of $P'$ and a third path $Q_3$ to some other vertex of $P'$ (Figure 3.6). Note that $\ell(Q_1) + \ell(Q_2)$ is at least the distance in $G$ between the endpoints of $P'$ which in turn is at least $\ell(P')/(1 + \epsilon)$ because $P' \subseteq P$ is an $\epsilon$-short path. Also, $\ell(Q_3) \geq \text{dist}(u, P')$. Combining, we get the following bound on the length of $T_u'$:

$$\ell(\hat{T}_u) = \ell(P') + \text{dist}_G(u, P') < (1 + \epsilon)(\ell(Q_1) + \ell(Q_2)) + \ell(Q_3) < (1 + \epsilon)(\ell(T_u)).$$

Figure 3.6: If $u$ has at least 3 children, then there are edge-disjoint $u$-to-$P$ paths, $Q_1$, $Q_2$, and $Q_3$.

Repeating this process until every vertex has at most two children produces a binary tree $T'$ rooted at $r$ and spanning the vertices of $T \cap P$. Since the vertices $\{u\}$ are chosen root-most, the trees $\{T_u\}$ are disjoint. Summing over all replacements, we infer that

$$\ell(\hat{T}) \leq (1 + \epsilon)\ell(T).$$

**Lemma 3.3.6.** Let $T$ be a tree in graph $G$ rooted at a vertex $r$ with leaves on an $\epsilon$-short path $P$. There is another tree $\hat{T}$ rooted at $r$ spanning the vertices of $T \cap P$ whose total length is at most $(1 + 4 \cdot \epsilon)\ell(T)$ such that $\hat{T}$ has at most $11 \cdot \epsilon^{-1.45}$ joining vertices with $P$. 

\[\square\]
Proof. Let $T'$ be the tree derived from $T$ in Lemma 3.3.5. Starting from $T'$, we construct a tree $T''$ from $T'$ that satisfies the properties of the lemma. We partition the edges of $T'$ into super-edges, defined by maximal paths in $T'$ whose internal vertices all have degree 2 in $T'$. The level of a super-edge is equal to the number of super-edges traversed when going from the root of $T'$ to the beginning of the super-edge. The endpoints of a super-edge are called super-vertices and the level of a super-vertex is the equal to the number of super-edges traversed when going from the root of $T'$ to the super-vertex.

Select a level $k$ (to be determined shortly). Let $U_k$ be the set of all super-vertices at level $k$. For each $u \in U_k$, replace the subtree $T'_u$ of $T'$ rooted at $u$ with another tree $T''_u$ rooted at $u$ that is the union of the minimum subpath $P'$ of $P$ spanning the vertices of $T'_u \cap P$ and the shortest $u$-to-$P'$ path (just as we did for Lemma 3.3.5, Figure 3.5). After all such replacements, we get a new tree, $T''$.

To analyze this construction, we show that there is a level $k \leq \lceil \log_\Phi (\sqrt{5}/\epsilon + 1) \rceil$ such that $\ell(T'') \leq (1 + \epsilon) \ell(T')$, where $\Phi$ is the golden ratio. Since $T'$ is binary, the number of super-vertices at level $k$ is at most

\[
2^k \leq 2^{1 + \log_\Phi (\sqrt{5}/\epsilon + 1)} \\
\leq 2^{(\sqrt{5}(\epsilon^{-1} + 1))^{1/\log_2 \Phi}} \\
< 11 \cdot \epsilon^{-1.45}
\]

assuming $\epsilon < 1$. Hence the number of joining vertices of $T''$ with $P$ is at most $11 \cdot \epsilon^{-1.45}$.

For a super-vertex $u$, there is a unique path $Q^1_u$ in $T'_u$ between the endpoints of $P'$. For a level $i$ of $T'$, let $E_i$ be the union over all super-vertices in level $i$ of the super-edges of $Q^1_u$. That is, $E_i = \bigcup_{u \in U_i} Q^1_u$. Note that the super-edges in $E_i$ are in level $\geq i$. For $i > 0$, let $L_i$ be the set of super-edges in level $i$ that are not in the set $E_{i-1}$. Let $L_0$ be one of the two super-edges in level 0. Note that $L_i \cap E_i$ is non-empty, and that $L_i \cap E_{i-1}$ is empty. For $j < i$, every super-edge in $E_j$ that has level $i$ is also in $E_{i-1}$. Since $L_i$ consists of super-edges having level $i$, $L_i \cap E_j$ is empty for $j < i$. For an illustration of $L_i$ and $E_i$, see Figure 3.7. Define $S_i = \bigcup_{j=i}^\infty L_j$.

![Figure 3.7: The bold edges are in $E_1$. The dotted edges are in $L_2$.](image)

Let $k'$ be the minimum index for which $\ell(L_{k'}) \leq \ell(S_{k'+2})$ (if there is no such level, let $k' = \infty$). Let

\[
k = \min \left( k', \left\lceil \log_\Phi (\sqrt{5}(1/\epsilon - 1)) \right\rceil \right),
\]
where $\Phi$ is the golden ratio. Clearly $k \leq \lceil \log_2(\sqrt{5}(1/\epsilon + 1)) \rceil$ and so the number of joining vertices of $T''$ with $P$ is at most $11 \cdot \epsilon^{-1.45}$ as argued above. The rest of the proof is devoted to showing $\ell(T'') \leq (1 + \epsilon)\ell(T')$.

Figure 3.8: The bold path is $Q_1$ and the dashed path is $Q_2$. Both paths use edge $e$, which is an edge of $L_i$ where $u$ is a vertex of level $i$. The tree rooted at $u$ will be replaced by the dotted tree.

Let $u$ be a super-vertex in level $k$. Note that for each vertex in level $i$, exactly one child super-edge is in $L_i$. Let $e$ be the super-edge in $L_k$ whose parent is $u$. Let $Q_1$ be the path in $T'$ between the endpoints of $P'$ and let $Q_2$ be the path from $u$ to $P$ that traverses $e$ and subsequently uses only edges of $E_{k+1} \setminus E_k$ (Figure 3.8). There are two cases:

**Case 1**, $k = k'$: Then $\ell(L_k) \leq \ell(S_{k+2})$. Note that $Q_1 \subseteq E_k$ and $Q_2 \subseteq E_{k+1} \cup \{e\}$. For $i > k + 1$, therefore, neither $Q_1$ nor $Q_2$ shares any edges with $L_i$. Hence neither $Q_1$ nor $Q_2$ shares any edges with $S_{k+2}$. We bound the length of $T''_u$:

$$\ell(T''_u) = \ell(P') + dist_G(u, P')$$

$$\leq (1 + \epsilon)[\ell(Q_1) + \ell(Q_2)]$$

$$< (1 + \epsilon)[\ell(T''_u) + \ell(e) - \ell(S_{k+2} \cap T''_u)]$$

$$< (1 + \epsilon)[\ell(T''_u) + \ell(L_k \cap T''_u) - \ell(S_{k+2} \cap T''_u)]$$

Summing over all $u \in U_k$, we bound the length of $T''$:

$$\ell(T'') < (1 + \epsilon)[\ell(T') + \ell(L_k) - \ell(S_{k+2})] < (1 + \epsilon)\ell(T').$$

**Case 2**, $k \neq k'$: Then $\ell(L_i) > \ell(S_{i+2})$ for every $i \leq k$. Note that $S_i$ is the disjoint union of $L_i$ and $S_{i+1}$, so

$$\ell(S_i) = \ell(L_i) + \ell(S_{i+1})$$

$$> \ell(S_{i+2}) + \ell(S_{i+1})$$

for every $i \leq k$. It follows that $\ell(S_k) > \text{Fib}_k \ell(S_k)$, where $\text{Fib}_k$ is the $k^{th}$ Fibonacci number. By our choice of $k$,

$$\text{Fib}_k > \frac{\Phi^k - 1}{\sqrt{5}} = \frac{\sqrt{5}/\epsilon}{\sqrt{5}} \geq \epsilon^{-1},$$

which implies that $\ell(S_k) < \epsilon \ell(S_1) \leq \epsilon \ell(T')$. We bound the length of $T''_u$:

$$\ell(T''_u) = \ell(P') + dist_G(u, P') \leq (1 + \epsilon)\ell(Q_1) + \ell(Q_2) < (1 + \epsilon)\ell(Q_1 \cup Q_2) + \ell(e)$$
Summing over all \( u \in \mathcal{U}_k \), we bound the length of \( T'' \):
\[
\ell ( T'') < (1 + \epsilon) \ell ( T') + \ell ( L_k ) < (1 + \epsilon) \ell ( T') + \ell ( S_k ) < (1 + 2\epsilon) \ell ( T') .
\]

Using \( T = T'' \), we find that \( \ell ( T ) < (1 + 2\epsilon) \ell ( T') < (1 + 2\epsilon)(1 + \epsilon) \ell ( T ) < (1 + 4\epsilon) \ell ( T ) \). \( \square \)

**Lemma 3.3.7.** Let \( T \) be a tree in planar graph \( G \) with leaves on an \( \epsilon \)-short path \( P \) that is a subpath of \( \partial G \). Let \( p \) and \( q \) be two vertices of \( T \). There is another tree \( \hat{T} \) that contains \( p \) and \( q \) and the vertices of \( T \cap P \) whose total length is at most \((1 + c_1 \epsilon) \ell ( T )\) such that \( \hat{T} \) has at most \( c_2 \cdot \epsilon^{-2.5} \) joining vertices with \( P \), where \( c_1 \) and \( c_2 \) are constants.

**Proof.** Let \( Q \) be the unique \( p \)-to-\( q \) path in \( T \). Removing the edges of \( Q \) from \( T \) breaks \( T \) into a forest with \( k \) trees rooted at vertices of \( Q \) and leaves on \( P \): \( \mathcal{T} = \{T_1, T_2, \ldots, T_k\} \). The root of \( T_i \) is \( r_i \) and is the unique vertex common to \( T_i \) and \( Q \). The trees are numbered according to their root along \( Q \) with \( r_1 = p \) and \( r_k = q \) (without loss of generality). See Figure 3.9 for an illustration. Since \( P \subseteq \partial G \), the leaves of \( T \) are in order along \( P \) and is consistent with the ordering of the trees \( T \). We include as (trivial) trees in this sequence all joining vertices of \( Q \) with \( P \).

![Figure 3.9: The edges of a tree with roots \( p \) and \( q \) is partitioned into trees \( T_1, T_2, \ldots, T_k \) and a \( p \)-to-\( q \) path \( Q \).](image)

We define a transformation:

Let \( T_a \) and \( T_b \) be such that \( a \leq b \). Let \( x \) and \( y \) be the endpoints of the minimum subpath of \( P \) spanning the leaves of \( T_a, T_{a+1}, \ldots, T_b \). Since \( P \subseteq \partial G \), we can assume that \( x \) is a vertex of \( T_a \) and \( y \) is a vertex of \( T_b \). Remove from \( T \) the trees \( T_a, T_{a+1}, \ldots, T_b \) and the \( r_a \)-to-\( r_b \) subpath of \( Q \). Add \( T_a[r_a, x], P[x, y], \) and \( T_b[r_b, y] \) creating tree \( T' \) as illustrated in Figure 3.10.

The path \( Q[p, r_a] \circ T_a[r_a, x] \circ P[x, y] \circ T_b[y, r_b] \circ Q[r_b, q] \) is a \( p \)-to-\( q \) path in \( T' \), so \( T' \) spans \( p \) and \( q \). Further, this transformation guarantees that \( T' \) spans the vertices of \( T \cap P \): a tree \( T_i \) that is removed from \( T \) has leaves on the \( x \)-to-\( y \) subpath of \( P \), which is included in \( T' \). So, \( T' \) spans the vertices required by the Lemma.

The increase in length due to this transformation is given by
\[
\Delta = \ell ( T_a[r_a, x] ) + \ell ( T_b[y, r_b] ) + \ell ( P[x, y] ) - \ell ( Q[r_a, r_b] ) - \sum_{i=a}^{b} \ell ( T_i ) .
\]
Now we find values of $a$ and $b$ for the transformation that reduces the number of joining vertices of the tree with $P$ while not increasing the length of the tree by much. Let $P'$ be the shortest subpath of $P$ that spans the vertices of $T \cap P$. Say a subtree $T_i$ is light if $\ell(T_i) < \epsilon \ell(P')$ and heavy otherwise. Let $I = \{ i : T_i \text{ is light} \}$.

Let $$ w = \begin{cases} (\min I) - 1 + k - (\max I) & \text{if } I \neq \emptyset \\ k & \text{otherwise} \end{cases} $$

\textbf{Case I, } $w \geq \epsilon^{-1} + 2$: In this case, there are at least $\epsilon^{-1} + 2$ heavy trees in $T$. We apply the transformation described above with $a = 1$ and $b = k$. The increase in length is given by:

$$ \Delta = \ell(T[p, x]) + \ell(T[y, q]) + \ell(P[x, y]) - \ell(Q) - \sum_{i=1}^{k} \ell(T_i) $$

$$ \leq \ell(P') - \sum_{i=2}^{k-1} \ell(T_i) $$

$$ < \ell(P') - \epsilon^{-1} \epsilon \ell(P'), $$

since there are at least $\epsilon^{-1} + 2$ heavy trees

$$ = 0. $$

The resulting tree $T'$ is a single path from $p$ to $q$ containing exactly two joining vertices with $P$: $x$ and $y$ (since $T' = \{x, y\}$). Since $\ell(T') < \ell(T)$, $T'$ achieves the properties of the Lemma.

\textbf{Case II, } $w < \epsilon^{-1} + 2$: In this case, we apply the transformation described with $a = \min I$ and $b = \max I$. The increase in length is given by:

$$ \Delta = \ell(T_a[r_a, x]) + \ell(T_b[y, r_b]) + \ell(P[x, y]) - \ell(Q) - \sum_{i=a}^{b} \ell(T_i) $$

$$ \leq 2a \ell(P') + \ell(P[x, y]) - \ell(T_a[r_a, x] \cup Q[r_a, r_b] \cup T_b[y, r_b]) $$

$$ \geq \ell(P'[x,y])/(1+\epsilon), \text{since } P \text{ is } \epsilon\text{-short} $$

$$ = 2a \ell(P') + \ell(P[x, y]) + \ell(T_a[r_a, x] \cup Q[r_a, r_b] \cup T_b[y, r_b]) - \ell(P'[x,y]) $$

$$ \leq 2\epsilon(1+\epsilon) \ell(T) + \epsilon \ell(T), $$

For the last inequality, observe that there is a path in $T$ between the endpoints of $P'$; since $P'$ is $\epsilon$-short, $(1+\epsilon)\ell(P')$ is at most the length of this path in $T$ (which in turn is at most the length of $T$ itself).
The new tree \( T' \) consists of the \( p \)-to-\( q \) path \( Q' \) and, attached to \( Q' \), the trees

\[
T_1, \ldots, T_{a-1}, T_{b+1}, \ldots, T_k.
\]

The joining vertices of \( T' \) with \( P \) include \( p, q \), and the joining vertices of all the trees \( T_i \). Though there are fewer than \( \epsilon^{-1} + 2 \) such trees remaining, each of the trees might itself have many joining vertices with \( P \).

Let \( T'_i \) be the tree obtained from \( T_i \) by applying Lemma 3.3.6 with the \( \epsilon \)-short path \( P \). Obtain a new tree \( T'' \) from \( T' \) by replacing each tree \( T_i \) with \( T'_i \). By Lemma 3.3.6, there are at most \( 11\epsilon^{-1.45} \) joining vertices with \( P \) per tree \( T'_i \). Since \( w < \epsilon^{-1} + 2 \), the new tree \( T'' \) has at most \( 11\epsilon^{-1.45}(\epsilon^{-1} + 2) + 2 \) joining vertices with \( P \) (the extra 2 counts the joining vertices \( x \) and \( y \)), achieving the last property of the lemma.

By Lemma 3.3.6, \( \ell(T'_i) < (1+\epsilon)\ell(T_i) \), and so \( \ell(T'') < (1+\epsilon)\ell(T') \) which, using the bound on the length of \( T' \), is at most \((1+\epsilon)(1+ce)\ell(T)\), satisfying \( T'' \), is at most \((1+\epsilon)(1+2\epsilon(2+\epsilon))\ell(T)\), satisfying the length bound for the lemma.

\[ \square \]

### 3.3.3 Simplifying forests inside bricks (Proof of Theorem 3.3.3)

We are now ready to prove Theorem 3.3.3. We are given a subgraph \( F \) embedded in a brick \( B \) that has boundary \( S \cup E \cup N \cup W \) where \( E \) and \( W \) are supercolumns. Let \( F_0 \) be a minimal subgraph of \( F \) such that if two vertices of \( N \cup S \) are connected in \( F \), then they are connected in \( F_0 \): \( F_0 \) is a forest whose leaves are all on \( N \) and \( S \). We partition \( F_0 \) into two forests \( F_1 \) and \( F_2 \) such that each component of \( F_1 \) has leaves only on \( N \) or \( S \) (but not both) and every vertex of \( F_2 \cap S \) belongs to a path \( P \) from \( S \) to \( N_B \) that shares only one vertex with \( S \) (i.e. the first vertex of \( P \)).

We find \( F_1 \) and \( F_2 \) as follows. Let \( x \) be a vertex of \( F_0 \cap (N \cup S) \). Cut \( F_0 \) at this vertex: suppose \( x \) has degree \( d \) in \( F_0 \); partition \( F_0 \) into \( d \) sets, in each of which \( x \) has degree one. Repeating for every such vertex partitions \( F_0 \) into a set of trees \( T \). Consider a tree \( T \) of \( T \) that spans vertices of both \( N \) and \( S \). Let \( y \) be a vertex of \( T \cap S \). If every path from \( y \) to \( N \) in \( T \) uses a vertex of \( S_B \) (other than \( y \)) then this vertex has degree \( > 1 \) in \( F_0 \) and would have undergone the cut operation described above.

The forest \( F_2 \) is the union of trees in \( T \) that have leaves on both \( N \) and \( S \) and \( F_2 = F_0 - F_1 \). These forests have the required properties. From forest \( F_i \) (for \( i = 1, 2 \)), we will build a forest \( \hat{F}_i \) satisfying the three properties of Theorem 3.3.3.

Before specifying \( \hat{F}_1 \) and \( \hat{F}_2 \), let us see how this implies Theorem 3.3.3. Define \( \hat{F} \) as the union of \( \hat{F}_1 \) and \( \hat{F}_2 \). Suppose two vertices \( z_0 \) and \( z_k \) of \( N \cup S \) are connected in \( F \). By construction, they are connected in \( F_0 \). By definition of \( F_1 \) and \( F_2 \), there are vertices \( z_1, \ldots, z_{k-1} \) of \( S \cup N \) such that for \( i = 1, \ldots, k \), vertices \( z_{i-1} \) and \( z_i \) are connected in either \( F_1 \) or \( F_2 \). By (F1), \( z_{i-1} \) and \( z_i \) are connected in either \( \hat{F}_1 \) or \( \hat{F}_2 \), and so they are connected in \( \hat{F} \). It follows that \( z_1 \) and \( z_k \) are connected in \( \hat{F} \).
We have that \( \hat{F} \) has at most \( \alpha(e) = 2\alpha(e) = o(\epsilon^{-5.5}) \) joining vertices with \( N \cup S \). Moreover,

\[
\ell(\hat{F}) \leq \ell(\hat{F}_1) + \ell(\hat{F}_2) \\
\leq (1 + \alpha(\epsilon))\ell(F_1) + (1 + \alpha(\epsilon))\ell(F_2) \\
\leq (1 + \alpha(\epsilon))(\ell(F_0)) \\
= (1 + \alpha(\epsilon))(\ell(F)).
\]

So \( \hat{F} \) satisfies all of the requirements of Theorem 3.3.3.

We now give the construction of \( \hat{F}_1 \). Let \( T \) be a connected component of \( F_1 \); by construction, \( T \) has leaves on either \( S \) or \( N \), but not both. Without loss of generality, assume that \( T \)'s leaves are on \( N \). Let \( \hat{T} \) be the tree guaranteed by Lemma 3.3.4. Replace each connected component of \( F_1 \) with such a tree produces a forest \( \hat{F}_1 \) with the desired properties. In fact \( \hat{F}_1 \) has no joining vertices with \( N \) or \( S \) and \( \ell(\hat{F}_1) \leq (1 + \epsilon)\ell(\hat{F}_1) \).

In the rest of this subsection, we give the construction of \( \hat{F}_2 \). Let \( s_0, \ldots, s_t \) be the vertices of \( S \) guaranteed by Lemma 3.3.1 (where \( s_0 \) is the vertex common to \( S \) and \( W \) and \( s_t \) is the vertex common to \( S \) and \( E \)).

We define a path \( P_i \) for each \( i \geq 0 \):

For \( i = 0 \), let \( x_0 \) be the last vertex of \( S \) that is in \( F_2 \) (where \( s_0 \) is the first vertex of \( S \)). Let \( k_0 \) be the minimum index such that \( x_0 \) is a vertex of \( S[s_0, s_{k_0}] \). By definition of \( F_2 \), there exists a path in \( F_2 \) from \( x_0 \) to \( N \) that does not use any vertices of \( S \) (other than \( x_0 \)). Let \( P_0 \) be the rightmost such path. That is, \( P_0 \) is the path in \( F_2 \) from \( x_0 \) to the last vertex common to \( F_2 \) and \( N \). This vertex is guaranteed to be connected to \( x_0 \) in \( F_2 \) by the definition of \( F_2 \).

For \( i \geq 1 \), let \( x_i \) be the last vertex of \( S[s_0, s_{k_i - 1}] \) for which there is a path in \( F_2 \) from \( x_i \) to \( N \) that does not use any vertices of \( S \) nor of \( P_{i-1} \). Let \( P_i \) be the rightmost such path.

Let \( y_i \) be the last vertex of \( P_i \) (\( y_i \) is a vertex of \( N \)). Let \( k_i \) be the minimum index such that \( x_i \) is a vertex of \( S[s_0, s_{k_i}] \). Let \( t' \leq t \) be the last index for which \( x_{t'} \) and \( P_{t'} \) exist. Note that \( \ell(S[s_{k_i}, x_i]) \leq \ell(P_i) \) by Property (B4) of Lemma 3.3.1. These paths are illustrated in Figure 3.11.

For \( 0 \leq i < t' \), let \( H_i \) be subgraph of \( F_2 \) enclosed by \( P_i, N, P_{i+1}, S \) not including any edges of \( P_{i+1} \). Let \( H_{t'} \) be the subgraph of \( F_2 \) enclosed by \( P_{t'}, N, E, \) and \( S \). Note that the graphs \( H_i \) form a partition of the edges of \( F_2 \). For the rest of our construction, we work in each \( H_i \) independently.

By definition of \( x_{i+1} \), any vertex of \( H_i \cap S[s_0, s_{k_i - 1}] \) is connected to \( P_i \) in \( F_2 \), and so in \( H_i \). By definition of \( P_{i+1} \), every vertex of \( H_i \cap N[y_{i+1}, y_i] \) that is connected to \( P_{i+1} \) is also connected to \( P_i \) or to \( S[s_{k_i - 1}, x_i] \). It follows that \( H_i \cup S[s_{k_i}, x_i] \) is a connected graph. We construct a graph \( \hat{H}_i \) from \( H_i \cup S[s_{k_i}, x_i] \). There are two cases.

\( P_{i+1} \cap H_i = \emptyset \): Define \( T_i \) to be a subtree of \( H_i \cup S[s_{k_i - 1}, x_i] \) that contains all of the edges of \( P_i \) and of \( S[s_{k_i - 1}, x_i] \) and spans all vertices of \( H_i \cap N \) and of \( H_i \cap S \). \( T_i \) is a tree rooted at \( s_{k_i - 1} \) with
Figure 3.11: We define a set of vertex-disjoint paths that break $F_2$ into manageable regions. There are at most $t \leq \kappa$ such paths (and the same number of regions).

leaves only on $N$. Define $r_i = s_k, -1$. Let $H'_i$ be the tree corresponding to $T_i$ that is guaranteed by Lemma 3.3.6. Let $\hat{H}_i = H'_i \cup S[s_k, -1, x_i]$.

$\hat{H}_i$ spans all the vertices of $H_i \cap N$ by Lemma 3.3.6. $\hat{H}_i$ spans all the vertices of $H_i \cap S$ since $H_i \cap S \subset S[s_k, -1, x_i]$ by construction and $S[s_k, -1, x_i] \subseteq \hat{H}_i$.

$H'_i$ (and so $\hat{H}_i$) has at most $c_1 \epsilon^{-1.45}$ joining vertices with $N$ by Lemma 3.3.6. Since $H'_i$ is a binary tree, $\hat{H}_i$ has at most $c_1 \epsilon^{-1.45}$ joining vertices with $S$.

We analyze the length of $\hat{H}_i$:

$$
\ell(\hat{H}_i) \leq \ell(H'_i) + \ell(S[s_k, -1, s_i])
$$

$$
\leq (1 + 4\epsilon)\ell(T_i) + \ell(S[s_k, -1, s_i])
$$

$$
\leq (1 + 4\epsilon)(\ell(H_i) + \ell(S[s_k, -1, x_i]) + \ell(S[s_k, -1, s_i]))
$$

$$
\leq (1 + 4\epsilon)(\ell(H_i)) + (2 + 4\epsilon)\ell(P_i), \text{ as observed above}
$$

$$
\leq (1 + c_2\epsilon)\ell(H_i), \text{ for a fixed constant } c_2.
$$

$q_{i+1} \in P_{i+1} \cap H_i$: Let $Q_i$ be a minimal path from $q_{i+1}$ to $S[s_k, -1, x_i] \circ P_i$ and let $r_i$ be the vertex common to $Q_i$ and $S[s_k, -1, x_i] \circ P_i$. Using an argument similar to the above, $H_i \cup S[s_k, -1, x_i] - Q_i$ is connected. By definition of $P_{i+1}$, $Q_i$ contains no vertex of $N$, except perhaps $r_i$. Let $T_i$ be a subtree of $H_i \cup S[s_k, -1, x_i] - Q_i$ that contains all the edges of $S[s_k, x_i] \cup P_i$ and spans all vertices of $H_i \cap N$, $H_i \cap S$, and $\{q_i, r_i\}$.

Let $T^S_i$ be the subgraph of $T_i$ that is enclosed by the cycle formed by $Q_i$, $S$, $P_i$ and $P_{i+1}$. Let $r_i$ be the root of $T^S_i$. $T^S_i$ has leaves only on $S$. Let $T^N_i = T_i - T^S_i$. $T^N_i$ is a tree because $T^N_i \cap T^S_i = \{r_i\}$. The leaves of $T^N_i$ (except perhaps $r_i$) are all on $N$. Without loss of generality, assume that $T^S_i$ does not contain $q_i$, the vertex possibly inherited from a similar construction in $H_{i-1}$.

Let $T^S_i$ be the tree guaranteed by Lemma 3.3.6 as applied to tree $T^S_i$, root $r_i$, and $\epsilon$-short path $S$: $\hat{T}^S_i$ spans $V(T^S_i \cap S) \cup \{r_i\}$. Let $\hat{T}^N_i$ be the tree guaranteed by Lemma 3.3.7 as applied to tree $T^N_i$, vertices $r_i$ and $q_i$, and $\epsilon$-short path $N$. Lemma 3.3.7 guarantees that $\hat{T}^N_i$ spans $V(T^N_i \cap N) \cup \{r_i, q_i\}$. Let $\hat{T}_i = \hat{T}^S_i \cup \hat{T}^N_i \cup S[s_k, x_i]$. 
If two vertices of $N \cup S \cup \{q_i, r_i\}$ are connected in $T_i$, then they are connected in $\hat{H}_i$. Let $\hat{H}_i = T_i \cup Q_i$. If two vertices of $N \cup S \cup \{q_i, q_i, r_i\}$ are connected in $H_i$, then they are connected in $\hat{H}_i$.

The number of joining vertices of $\hat{H}_i$ with $N$ and $S$ is equal to the number of joining vertices of $\hat{T}_i^N$ with $N$, which is at most $c_3 \epsilon^{-2.5}$ by Lemma 3.3.7, plus the number of joining vertices of $\hat{T}_i^S$ with $S$, which is at most $c_4 \epsilon^{-1.45}$.

We analyze the length of $\hat{H}_i$:

$$\ell(\hat{H}_i) \leq \ell(T_i) + \ell(T_i^N) + \ell(S[q_{k_i-1}, s_i]) + \ell(Q_i)$$

$$\leq (1 + c_6 \epsilon)\ell(T_i^S) + (1 + c_7 \epsilon)\ell(T_i^N) + \ell(P_i) + \ell(Q_i)$$

by Lemmas 3.3.6, 3.3.7 and 3.3.1

$$\leq (1 + c_8 \epsilon)\ell(T_i \cup Q_i) + \ell(P_i)$$

$$\leq (1 + c_8 \epsilon)(\ell(H_i) + \ell(S[q_{k_i-1}, s_i])) + \ell(P_i)$$

$$\leq (1 + c_8 \epsilon)(\ell(H_i) + \ell(P_i)) + \ell(P_i)$$

by Lemma 3.3.1

$$\leq (1 + c_9 \epsilon)\ell(H_i).$$

We define $\hat{F}_2$ to be the union of $\hat{H}_i$ over all $i$. It only remains to show that $\hat{F}_2$ satisfies properties (F1), (F2), and (F3) of Theorem 3.3.3:

(F1) Suppose vertices $z_a$ and $z_b$ are vertices of $N \cup S$ that are connected in $F_2$. Let $R$ be the path from $z_a$ to $z_b$ in $F_2$. Let $H_a, H_{a+1}, \ldots, H_b$ be the subgraphs of $F_2$ that contain edges of $R$. Since $R$ is a path in $F_2$, $H_i$ is connected to $H_{i+1}$ for $a \leq i < b$, and so $q_i$ and $q_{i+1}$ are connected in $H_i$. Likewise since $z_a$ is a vertex of $S \cup N$, $z_a$ is connected to $q_j$ for some $a \leq j < b$ (and likewise for $z_b$). It follows that $z_a$ and $z_b$ are connected in $\hat{F}_2$.

(F2) $\hat{F}_2$ is the union of $t' \leq t \leq \kappa$ subgraphs $\hat{H}_i$. We have seen that $\hat{H}_i$ has at most $c_4 \epsilon^{-2.5}$ joining vertices with $N \cup S$. $\hat{F}_2$ has at most $c_4 \kappa \epsilon^{-2.5} = c_10 \epsilon^{-5.5}$ joining vertices with $N \cup S$, using Equation (3.1).
(F3) We bound the length of $\hat{F}_2$:

\[
\ell(\hat{F}_2) = \sum_{i=0}^{t'} \ell(\hat{H}_i) \\
\leq \sum_{i=0}^{t'} (1 + c' \epsilon) \ell(H_i) \\
= (1 + c' \epsilon) \ell(F_2)
\]

since the $H_i$'s are disjoint.

Theorem 3.3.3 is proved.

\[\blacksquare\]

3.3.4 Simplifying subgraphs of 2-EC multi-subgraphs inside bricks

Theorem 3.3.3 applies directly to the Steiner-tree problem: the intersection of a tree with a brick is a forest and since out terminals are vertices of $MG$, it is enough to maintain connectivity between vertices on the boundary of a brick. However, for the 2-EC problem, the intersection of a solution with a brick has a more complicated structure. At the end of this section we will state and prove a counterpart to Theorem 3.3.3 that maintains up to 2 connectivity between vertices on the north and south boundary of a brick.

For a graph $H$ and vertices $x$ and $y$, define

\[
c_H(x, y) = \min\{2, \text{maximum number of edge-disjoint } x\text{-to-}y \text{ paths in } H\}.
\]

For two (multi-)subgraphs $H$ and $H'$ of a common graph $G$, $H'$ achieves the connectivity of $H$ if $c_{H'}(x, y) \geq c_H(x, y)$ for all pairs of vertices. We will restrict our attention to connectivity between boundary vertices of a common graph $G$. That is, if $H$ and $H'$ are multi-subgraphs of $G$, then $H'$ achieves the boundary 2-connectivity of $H$ if $H'$ achieves the 2-connectivity of $H$ for pairs of vertices on the boundary of $G$.

**Lemma 3.3.8.** Let $H$ be a (multi-)subgraph of $G$ and let $C$ be a cycle of $H$. Let $H'$ be the subgraph of $H$ obtained by removing the edges of $H$ that are strictly enclosed by $C$. $H'$ achieves the boundary 2-connectivity of $H$.

**Proof.** Without loss of generality, take $C$ to be a clockwise cycle. Consider two boundary vertices $x$ and $y$. We show that there are $c_H(x, y)$ edge-disjoint $x$-to-$y$ paths in $H$ that do not use edges strictly enclosed by $C$. There are 2 non-trivial cases:

$c_H(x, y) = 1$: Let $P$ be an $x$-to-$y$ path in $H$. If $P$ intersects $C$, let $x_P$ be the first vertex of $P$ that is in $C$ and let $y_P$ be the last vertex of $P$ that is in $C$. Let $P' = P[x, x_P] \circ C[x_P, y_P] \circ P[y_P, y]$. If $P$ does not intersect $C$, let $P' = P$. $P'$ is an $x$-to-$y$ path in $H$ that has no dart strictly enclosed by $C$.

$c_H(x, y) = 2$: Let $P$ and $Q$ be edge-disjoint $x$-to-$y$ paths in $H$. If $Q$ does not intersect $C$, then $P'$ and $Q$ are edge-disjoint paths, neither of which has a dart strictly enclosed by $C$ (where $P'$ is as
defined above). Suppose that both $P$ and $Q$ intersect $C$. Define $x_Q$ and $y_Q$ as for $P$. Suppose these vertices are ordered $x_P$, $x_Q$, $y_Q$, $y_P$ around $C$. Then $P[x, x_P] \circ C[x_P, y_Q] \circ Q[y_Q, y]$ and $Q[x, x_Q] \circ \text{rev}(C[y_P, x_Q]) \circ P[y_P, y]$ are edge disjoint $x$-to-$y$ paths that do not use any darts enclosed by $C$. This case is illustrated in Figure 3.13; other cases follow similarly.

We have shown that we can achieve the boundary 2-connectivity of $H$ without using any edges enclosed by a cycle of $H$. The lemma follows.

![Figure 3.13: An illustration of the proof of Lemma 3.3.8: there are edge disjoint $x$-to-$y$ paths that do not use edges enclosed by $C$.](image)

We say that a planar graph $G$ has the empty-cycle property if for every cycle $C$ of $G$, $C$ does not strictly enclose any edges. The following is easy to achieve from the above lemma.

**Corollary 3.3.9.** Let $H$ be a subgraph of $G$. There is a subgraph $H'$ of $H$ that achieves the boundary 2-connectivity of $H$ and has the empty-cycle property.

**Lemma 3.3.10.** Let $H$ be a subgraph of $G$ and let $a, b, c, d$ be a set of vertices in order along $\partial G$. If $c_H(a, c) = 2$ and $c_H(b, d) = 2$ then $c_H(x, y) = 2$ for $x, y \in \{a, b, c, d\}$.

**Proof.** Let $P_1$ and $P_2$ be non-crossing edge-disjoint $a$-to-$c$ paths and let $Q_1$ and $Q_2$ be edge-disjoint $b$-to-$d$ paths. Let $C = P_2 \circ \text{rev}(\partial G[a, c])$. Without loss of generality, assume that $P_2$ is left of $P_1$. Then $b$ is a vertex of $C$ and $C$ does not enclose $P_1$ (as pictured in Figure 3.14). Let $Q_1'$ be the longest prefix of $Q_1$ that is enclosed by $C$. Similarly define $Q_2'$.

Without loss of generality assume that $\text{end}(Q_1)$ occurs before $\text{end}(Q_2')$ on $P_2$. Then $Q_2' \circ P_2[\text{end}(Q_2'), c]$ and $Q_1' \circ \text{rev}(P_2[\text{end}(Q_1), a]) \circ P_1$ are edge-disjoint $b$-to-$c$ paths and so $c_H(b, c) = 2$. Similarly $c_H(c, d) = 2$. The lemma follows from the transitivity of 2-edge connectivity.

**Lemma 3.3.11.** Let $P$ and $Q$ be leftmost non-self-crossing $x_P$-to-$y_P$ and $x_Q$-to-$y_Q$ paths, respectively, where $x_P$, $y_P$, $x_Q$, and $y_Q$ are vertices in order on $\partial G$. Then $x_P$ does not cross $y_P$.

**Proof.** For a contradiction, assume that $Q$ crosses $P$. Let $C$ be the cycle $P \circ \text{rev}(\partial G[y_P, x_P])$ where $\partial G$ is counterclockwise. $C$ not contain both $x_Q$ and $y_Q$. If $Q$ crosses $P$, there must be a subpath of $Q$ enclosed by $C$. Let $x$ be the first vertex of $Q$ in $P$ and let $y$ be the last. There are two cases:

$x \in P[x_P, y] \setminus Q[x_Q, x] \circ \text{rev}(P[x_P, x]) \circ \partial G[x_P, x_Q]$ is a cycle that strictly encloses $y$ and does not enclose $y_Q$. Since $y$ is the last vertex of $Q$ on $P$, $Q$ must cross itself, a contradiction.
\( x \in P[y, y_P] \): Since \( Q \) crosses \( P \), at least part of \( Q[x, y] \) is enclosed by \( C \). Therefore \( Q[x, y] \circ P[y, x] \) is not clockwise; so \( Q[x, y] \) is not left of \( \text{rev}(P[y, x]) \), contradicting the leftmost of \( Q \).

\( \square \)

We call a set of vertices \( S \) a 2-requirement clique if \( c_H(x, y) = 2 \) for all \( x, y \in S \).

**Lemma 3.3.12.** Let \( H \) be a subgraph of \( G \). Let \( S \) be a boundary 2-requirement clique of \( H \). Then there is a non-self-crossing cycle \( C \) in \( H \) such that \( S \subseteq V(C) \) and the order that \( C \) visits the vertices in \( S \) is the same as their order along \( \partial G \).

**Proof.** Assume that the vertices of \( S \) are in the order \( s_1, s_2, \ldots, s_k \) along \( \partial G \).

Let \( P_i \) be the leftmost non-self-crossing \( s_i \)-to-\( s_{i+1} \) path in \( H \) taking the indices modulo \( k \). Let \( C = P_1 \circ P_2 \circ \cdots \circ P_{k-1} \). Certainly \( C \) visits each of the vertices \( s_1, s_2, \ldots \) in order. By Lemma 3.3.11, \( P_i \) does not cross \( P_j \). Therefore, \( C \) is non-self-crossing, proving the lemma. \( \square \)

We will use the algorithm DecomposeConnectivity in proving Lemma 3.3.16 and in a dynamic program in Section 3.7. We will show in Lemma 3.3.16 that given 1- or 2-edge-connectivity requirements \( c \) between a set of boundary vertices \( S \), DecomposeConnectivity\((S, c)\) finds a set of subsets \( \mathcal{X} \) of \( S \) such that independently satisfying 1-connectivity for each of the sets in \( \mathcal{X} \) will satisfy the requirements \( c \).

**Lemma 3.3.13.** DecomposeConnectivity takes time \( O(|S|^2) \).

**Proof.** Since each 2-requirement clique has size at least 2, and these vertices will not be in a 2-requirement clique of a subproblem, the recursion tree has at most \( |S|/2 \) levels. Each vertex of the 2-requirement clique, \( S' \), will appear in two subproblems, and each vertex will be duplicated at most once. The total number of vertices considered at each level of recursion is at most \( 2|S| \). The overall running time is \( O(|S|^2) \). \( \square \)

**Lemma 3.3.14.** Let \( \mathcal{X} \) be the output of DecomposeConnectivity. Then \( \mathcal{X} \) is a non-crossing multi-partition of \( S \).
DecomposeConnectivity(S,c):

1. Let \( S' = \{s_1, s_2, \ldots, s_k\} \) be a maximal 2-requirement clique in \( S \) corresponding to \( c \). Let \( s_{k+1} = s_1 \).
2. If \( |S'| = 0 \), return \( \{S\} \).
3. Let \( X = \emptyset \).
4. Let \( c'(s_i, s_{i+1}) = c'(s_{i+1}, s_i) = 1 \) for \( i = 1, \ldots, k \).
5. For \( i = 1, \ldots, k \),
   6. Let \( S_i \) be the vertices of \( S \) ordered between \( s_i \) and \( s_{i+1} \).
   7. \( X = X \cup \text{DecomposeConnectivity}(S_i, c') \).
8. Return \( X \).

Table 3.2: DecomposeConnectivity takes as input a cyclically ordered set vertices \( S \) and a symmetric function \( c : S \times S \rightarrow \{1, 2\} \). The algorithm returns a set of subsets of \( S \).

Proof. The recursive calls of DecomposeConnectivity are on non-crossing subsets of \( S \). Since every element of \( S \) is returned at least once by Step 2, \( X \) is a multi-partition of \( S \).

We will need the following lemma to prove the Structure Theorem for 2-edge connectivity.

Lemma 3.3.15. Let \( x \) and \( y \) be vertices that are consecutive in \( S \) in a maximal 2-requirement clique. Then there is a set \( X \) returned by DecomposeConnectivity such that \( x, y \in X \) and \( X \) is a subset of the elements in \( S \) between \( x \) and \( y \).

Proof. Without loss of generality, assume that \( x = s_1 \) and \( y = s_2 \) in Step 1 of DecomposeConnectivity. Then, \( x \) and \( y \) will be consecutive in the set \( S_1 \) in the recursive call. Since they will not appear in another 2-requirement clique, by the maximality of the cliques, \( x \) and \( y \) will remain consecutive until they are output in a set in Step 2. Further \( S_1 \) contains only vertices of \( S \) that are in order between \( x \) and \( y \).

We are now ready to prove the main lemma:

Lemma 3.3.16. Let \( H \) be a subgraph of \( G \) that satisfies the Empty-Cycle Property. Let \( X = \{X_1, X_2, \ldots\} \) be the result of DecomposeConnectivity(\( \partial G \cap H, c_H \)). Then \( H \) is the disjoint union of trees \( T_i \) such that

(T1) \( T_i \) spans \( X_i \);

(T2) \( \cup_i \hat{T}_i \) achieves the boundary 2-connectivity of \( H \) where \( \hat{T}_i \) is any subgraph that spans \( X_i \).

(T3) \( X \) is a non-crossing sub-partition of \( V(\partial G) \).

Proof. Without loss of generality, assume that \( H \) is connected. We prove the theorem inductively via the recursion of DecomposeConnectivity. Let \( S \) be the vertices of \( H \cap \partial G \), ordered along \( \partial G \).
Let $S' = \{s_1, s_2, \ldots, s_k\}$ be the 2-requirement clique found in Step 1 of DECOMPOSECONNECTIVITY. Let $C$ be the corresponding non-self-crossing cycle guaranteed by Lemma 3.3.12. Assume that $C$ and $\partial G$ are clockwise.

Let $H_i$ be the subgraph of $H$ that is enclosed by $C[s_i, s_{i+1}] \circ \text{rev}(\partial G[s_i, s_{i+1}])$ for some $i \in \{1, \ldots, k\}$. Note that $C[s_i, s_{i+1}] \subseteq H_i$. We have several observations:

Fact 1 $H$ is the disjoint union $\cup_i H_i$.

Fact 2 No path in $H_i$ crosses any path in $H_j$. (Since $C$ is non-self-crossing, $H_i$ is edge disjoint from $H_j$ for $i \neq j$.)

Fact 3 Let $x \in \partial G(s_i, s_{i+1})$ and $y \in \partial G(s_{i+1}, s_i)$. Then $c_H(x, y) \leq 1$. (This follows from the maximality of $S$ and Lemma 3.3.10: if $c_H(x, y) \geq 2$, then $x$ and $y$ would be in $S$.)

Fact 4 Let $x \in \partial G(s_i, s_{i+1})$ and $y \in \partial G(s_{i+1}, s_i)$. If $c_H(x, y) = 1$, then $c_H(x, s_i) = 1$. (An $x$-to-$y$ path would have to intersect the $s_i$-to-$s_{i+1}$ path.)

Fact 5 Let $x \in \partial G(s_i, s_{i+1})$ and $y \in \partial G(s_i, s_{i+1})$. If $c_H(x, y) \geq 2$, then there are $x$-to-$y$ edge-disjoint paths $P$ and $Q$ in $H_i$. (Otherwise, $c_H(x, s_i) \geq 2$, contradicting the maximality of $S$.)

Consider the modified requirements found in Step 4. Let $c_i$ be the requirements in $c'$, restricted to vertices in $\partial G[s_i, s_{i+1}] \cap S$. It follows from Facts 4 and 5 that $H_i$ satisfies the requirements (i.e. $c_{H_i}(x, y) \geq c_i(x, y) \forall x, y \in V(\partial G)$). In particular, $H_i$ spans the vertices $S_i$. Inductively, DECOMPOSECONNECTIVITY terminates when the requirements between vertices of the input set $S$ are all unitary: therefore $H$ is the union of trees, each of which spans a set $X \in \mathcal{X}$ (giving Property 1). Property 3 follows from Lemma 3.3.14. Now we prove Property 2 inductively by showing that Property 2 holds for one level of recursion.

Let $\widehat{H}_i$ be any subgraph of $G$ that satisfies the new requirements, $c_i$. Let $\widehat{H}$ be the disjoint union $\cup_i \widehat{H}_i$. We show that $\widehat{H}$ achieves the boundary 2-connectivity of $H$, thus proving (T3). We have three cases:

1. Consider the connectivity between $s_i$ and $s_j$. The requirements guarantee there is an $s_i$-to-$s_{i+1}$ path $P_i$ in $\widehat{H}_i$. Since $\widehat{H}$ is a disjoint union of the $\widehat{H}_i$’s, $P_i \circ P_{i+1} \circ \cdots \circ P_{j-1}$ and $P_j \circ P_{j+1} \circ \cdots \circ P_{i-1}$ are edge-disjoint $s_i$ to $s_j$ paths.

2. Let $x$ and $y$ be vertices of $\partial G[s_i, s_{i+1}]$ (but with $\{x, y\} \neq \{s_i, s_{i+1}\}$). The connectivity requirements between $x$ and $y$ are not modified in Step 4 so $c_i(x, y) = c(x, y)$ and $\widehat{H}_i$ satisfies these requirements.

3. Let $x$ be a vertex of $\partial G(s_i, s_{i+1})$ and let $y$ be a vertex of $\partial G(s_j, s_{j+1})$ with $i \neq j$. By Fact 3 $c_H(x, y) \leq 1$. By Fact 4, if $c_H(x, y) = 1$, then $c_H(x, s_i) = 1$ and $c_H(y, s_j) = 1$. By the requirements, there is an $x$-to-$s_i$ path in $\widehat{H}_i$ and a $s_j$-to-$y$ path in $\widehat{H}_j$. We have already argued that $s_i$ and $s_j$ are connected in $\widehat{H}$. 
We illustrate the 2-requirement cliques found by \textsc{DecomposeConnectivity} and the set of trees comprising $H'$ in Figure 3.15.

![Figure 3.15: (a) A subgraph (thick edges) $H$ of $G$ (whose boundary is rectangular) satisfying the Empty-Cycle Property can be decomposed into a set of trees (b) whose leaves are on the boundary of $G$. The leaves of the trees correspond to the sets $X_i$ found by \textsc{DecomposeConnectivity}. The dark edges in (a) correspond to the 2-requirement cliques found in \textsc{DecomposeConnectivity}.](image)

We are now ready to state and prove a theorem that is to 2-connectivity what Theorem 3.3.3 (Structural Properties of Bricks) is to Steiner trees.

**Theorem 3.3.17.** Let $B$ be a plane graph with boundary $N \cup E \cup S \cup W$ and satisfying the brick properties of Lemma 3.3.1. Let $H$ be a subgraph of $B$. There is another subgraph $\hat{H}$ with the following properties:

(H1) $\hat{H}$ achieves the 2-connectivity of $H$ for vertices of $N \cup S$.

(H2) The number of joining vertices of $\hat{H}$ with both $N$ and $S$ is at most $2\alpha(\epsilon)$.

(H3) $\ell(\hat{H}) \leq (1 + c\epsilon)\ell(H)$.

In the above, $\alpha(\epsilon)$ is the function given by Theorem 3.3.3 and $c$ is a fixed constant.

**Proof.** Let $H'$ be a minimal subgraph of $H$ such that $H'$ achieves the 2-connectivity of $H$ for vertices on $N \cup S$. By Corollary 3.3.9, $H'$ has the empty-cycle property.

By Lemma 3.3.16, $H'$ is the union of a set of disjoint trees $T = \{T_1, T_2, \ldots\}$ where $T_i$ spans a set of vertices $X_i \in V(N \cup S)$. Partition $T$ into three sets:

$T_1 = \{T_i \in T \text{ such that } X_i \subseteq V(N) \text{ or } X_i \subseteq V(S)\}$

$T_2 = \{T_i \in T \text{ such that } X_i \text{ has vertices in both } V(N) \text{ and } V(S)\}$

$T_3 = \{T_i \in T \text{ such that } X_i \text{ is adjacent to both } N \text{ and } S\}$

We further partition $T_2$. Let $T_i$ and $T_j$ be two trees in $T_2$. Since $X$ is a non-crossing multi-subpartition of boundary vertices of the brick, if the vertices $X_i \cap V(S)$ appear before $X_j \cap V(S)$ along $S$ then the vertices in $X_i \cap V(N)$ appear before $X_j \cap V(N)$ along $N$. It follows that there is an ordering of the trees in $T_2$ from left to right in the brick. Let $T_A$ be the set of trees of $T_2$ that are even numbered in this ordering and let $T_B = T_2 \setminus T_A$. That is, the trees in $T_2$ alternate between $T_A$ and $T_B$.\]
Consider a tree \( T_i \in \cal T \) such that (w.l.o.g.) \( X_i \subseteq V(N) \). Let \( \hat{T} \) be the minimal subpath of \( N \) that spans \( X_i \). Let \( \hat{F}_1 \) be the disjoint union of \( \{ \hat{T} : T \in \cal T \} \). Let \( \hat{F}_A \) be the forest guaranteed by Theorem 3.3.3 for the graph obtained by taking the union of the trees in \( \cal T_A \). Similarly define \( \hat{F}_B \). Let \( \hat{H} \) be the disjoint union of \( \hat{F}_1, \hat{F}_2, \hat{F}_A, \hat{F}_B \). We show that \( \hat{H} \) achieves the required properties.

It is clear from the construction that \( \hat{F}_1 \) does not have any joining vertices with \( N \) or \( S \). By Theorem 3.3.3, each of \( \hat{F}_A \) and \( \hat{F}_B \) has at most \( \alpha(\epsilon) \) joining vertices with \( N \cup S \). Therefore \( \hat{H} \) has at most \( 2\alpha(\epsilon) \) joining vertices with \( N \cup S \), proving Property H2.

Since \( N \) and \( S \) are \( \epsilon \)-short paths, \( \ell(\hat{F}_1) \leq (1+\epsilon)\ell(\hat{F}_1) \). By Theorem 3.3.3, \( \ell(\hat{F}_A) \leq (1+c\epsilon)\ell(\hat{F}_A) \) and \( \ell(\hat{F}_B) \leq (1+c\epsilon)\ell(\hat{F}_B) \). It follows that \( \ell(\hat{H}) \leq (1+c\epsilon)\ell(\hat{H}) \), proving Property 3.

First we show that if two vertices of \( N \cup S \) are connected in \( H' \), then they are connected in \( \hat{H} \).

Consider a set \( X_i \in \mathcal{X} \). There is a tree \( T_i \in \mathcal{T} \) that spans \( X_i \). By construction there is a tree (perhaps a subtree) in \( \hat{H} \).

If \( T_i \) is in \( \mathcal{T}_1 \), by construction \( \hat{T}_i \) spans Lemma 3.3.4 guarantees that the vertices \( X_i \) will be connected in \( \hat{F}_1 \). Let \( \hat{T}_i \) be a subgraph of \( \hat{H} \) that spans \( X_i \). By Property 2 of Lemma 3.3.16, \( \cup_i \hat{T}_i \) achieves the 2-connectivity of \( H' \). Note that we have not shown that \( \hat{T}_i \) is edge-disjoint from \( \hat{T}_j \), and so \( \hat{H} \) is not necessarily the disjoint union of these trees. However, this is sufficient to show that if two vertices of \( N \cup S \) are connected in \( H' \), then they are connected in \( \hat{H} \).

We now show that if two vertices of \( N \cup S \) are 2-edge connected in \( H' \), then they are 2-edge connected in \( \hat{H} \). This will complete the proof.

Let \( a \) and \( b \) be vertices of \( N \cup S \) that are 2-edge connected in \( H' \). Let \( Y \subseteq V(N \cup S) \) be the maximal 2-requirement clique such that \( a, b \in Y \). Let \( y_1, y_2, \ldots, y_k \) be the order of the vertices of \( Y \) along the boundary of the brick. Let \( X_i \) be the set such that \( y_i, y_{i+1} \in X_i \) and \( X_i \subseteq V(\partial B[y_i, y_{i+1}]) \) (as guaranteed by Lemma 3.3.15).

We have three cases:

\[ Y \subseteq V(N) \]: Without loss of generality, assume that \( y_1 \) is the first vertex and \( y_k \) is the last vertex of \( Y \) along \( N \). Then \( X_1, \ldots, X_{k-1} \) are subsets of \( N \). \( X_k \) may contain vertices of \( S \). Let \( \hat{T}_i \) be a tree in \( \hat{F}_1 \) that spans \( X_i \) (for \( i = 1, \ldots, k-1 \)). Since \( \hat{F}_1 \) is the disjoint union of these trees, there is a path \( P \) in \( \hat{F}_1 \) that visits each vertex \( y_1, \ldots, y_k \) in order.

If \( X_k \) spans a vertex of \( S \) then \( X_k \in F_A \) (without loss of generality). The vertices \( X_k \) are spanned by \( \hat{F}_A \) and so there is a \( y_k \)-to-\( y_1 \) path \( Q \) in \( \hat{H} \) that is edge disjoint from \( P \). \( P \circ Q \) is a cycle such that \( Y \subseteq V(P \cup Q) \). The vertices in \( Y \) are 2-edge connected in \( \hat{H} \).

\[ Y \subseteq V(S) \]: This case follows as the above case.

\[ Y \cap V(N) \neq \emptyset \text{ and } Y \cap V(S) \neq \emptyset \]: Without loss of generality, assume that \( y_1 \) and \( y_l \) are the first and last vertices of \( Y \) along \( N \). Then \( y_k \) and \( y_{l+1} \) are the first and last vertices of \( Y \) along \( S \). By the argument used in the above case, there is a path \( P \) in \( \hat{H} \) that visits the vertices \( y_1, \ldots, y_l \) in order. Likewise, there is a path \( Q \) in \( \hat{H} \) that visits the vertices \( y_{l+1}, \ldots, y_k \) in order. We now argue that there are edge-disjoint \( y_l \)-to-\( y_{l+1} \) and \( y_k \)-to-\( y_1 \) paths in \( \hat{H} \) by showing that \( T_i \)
(the tree corresponding to $X_l$) is in $F_A$ and $T_k$ (the tree corresponding to $X_k$) is in $F_B$. (This case is illustrated in Figure 3.16.)

At some point during DecomposeConnectivity, $Y$ is the clique that is considered in Step 1. Let $C$ be the clique-cycle corresponding to $Y$ (Lemma 3.3.12). Suppose there is another clique cycle $C'$ that contains a vertex of $N(y_1, y_l)$ and a vertex of $S(y_k, y_{l+1})$. Then $C \cup C'$ is connected and is a clique cycle that visits a vertex set larger than $Y$, violating the maximality of $Y$. So $Y$ is the only clique considered by DecomposeConnectivity that contains vertices of both $N(y_1, y_l)$ and $S(y_k, y_{l+1})$. The subproblems corresponding to $Y$ that include vertices of $N(y_1, y_l)$ only include vertices of $N(y_1, y_l)$. It follows that there is no set $X$ in $X'$ that contains a vertex between $y_1$ and $y_l$ along $N$ and a vertex between $y_k$ and $y_{l+1}$ along $S$ (other than $X_l$ and $X_k$). Therefore, there is no tree $T$ in $T_3$ that is ordered between $T_l$ and $T_k$. If $T_l \in T_A$ (without loss of generality), then $T_k \in T_B$.

![Figure 3.16: If there are two edge-disjoint paths between $x$ and $y$, then there is a cycle $C$ (bold) through $x$ and $y$. Exactly two subpaths $P$ and $Q$ of $C$ (solid) will appear in $T_2$. Since $C$ encloses no edges, $P$ and $Q$ are alternating in the order used to define $T_A$ and $T_B$. There will be edge disjoint paths between the endpoints of $P$ and $Q$ in $\hat{H}$.

3.4 The Structure Theorem

In this section, use the brick decomposition as a scaffolding to build a graph called the portal connected graph. In the portal connected graph, we can

- compute an optimal solution via the spanner method (Section 3.5) or a near-optimal solution via a parcel-decomposition (Section 3.6), and
- derive a near-optimal solution in the original graph using the Structure Theorem.

An illustration of the construction of the portal connected graph is given in Figure 3.17.

3.4.1 Portals

In order to define the portal connected graph we first need to designate as portals some vertices of $\partial B$ for each brick $B$. 
Figure 3.17: (a) A mortar graph $MG$. (b) The corresponding set of bricks. For each brick, $B$, we select a subset of the boundary vertices and designate them as portals. (c) Each brick is embedded in the corresponding face of $MG$. The portals are connected to the corresponding vertices in $MG$ via portal edges shown in grey.

Let

$$\theta = \theta(\epsilon) = 10\alpha(\epsilon)\epsilon^{-2},$$

(3.2)

where $\alpha(\epsilon) = o(\epsilon^{-5.5})$ as given by Theorem 3.3.3. We the algorithm given in Table 3.3.

<table>
<thead>
<tr>
<th>PortalSelection($B$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Let $v_0 \in V(\partial B)$ be the endpoint of an edge strictly enclosed by $\partial B$.</td>
</tr>
<tr>
<td>2. Designate $v_0$ as a portal vertex.</td>
</tr>
<tr>
<td>3. Set $i = 0$.</td>
</tr>
<tr>
<td>4. Repeat:</td>
</tr>
<tr>
<td>5. Let $v_i$ be the first vertex of $\partial B$ such that $\ell(\partial B[v_{i-1}, v_i]) &gt; \ell(\partial B)/\theta$.</td>
</tr>
<tr>
<td>6. If $v_0 \in V(\partial B(v_{i-1}, v_i))$, stop.</td>
</tr>
<tr>
<td>7. Otherwise, designate $v_i$ as a portal vertex and set $i = i + 1$.</td>
</tr>
</tbody>
</table>

Table 3.3: A greedy algorithm to select portals.

The following lemma follows trivially:

**Lemma 3.4.1** (Coverage Property). For any vertex $x$ on $\partial B$, there is a portal $y$ such that the $x$-to-$y$ subpath of $\partial B$ has length at most $\ell(\partial B)/\theta$.

**Lemma 3.4.2** (Cardinality Property). There are at most $\theta$ portals on $\partial B$.

*Proof.* Suppose there are $p$ iterations. Each iteration selects a subpath of length more than $\ell(\partial B)/\theta$, so we have $\ell(\partial B) \geq \sum_{i=1}^{p} \ell(\partial B[v_{i-1}, v_{i-1}]) > p \ell(\partial B)/\theta$, and so it follows that $p < \theta$.

### 3.4.2 Portal-connected graph

In preparation for stating the Structure Theorem, we define an operation called *brick insertion*. For any subgraph $G$ of $MG$, we derive a planar embedded graph $B^+(G)$ as follows. For each face $f$ of $G$ corresponding to a brick $B$, embed a copy of $B$ inside the face $f$, and, for each portal vertex $v$ of $B$, connect $v$ in the brick to the corresponding vertex in $f$, using a zero-length artificial edge (or, in
Figure 3.18: A brick (a) is inserted by embedding it in the corresponding face of $MG$ (b) and introducing a portal edge connecting each portal to the corresponding vertex of $MG$ (c).

the case of 2-EC, using 2 zero-length artificial edges). We refer to the artificial edges as portal edges. This step is illustrated for a single brick in Figure 3.18 and for a collection of bricks in Figure 3.17(c).

We refer to $B^+(MG)$ as the portal-connected graph. Intuitively, this graph is almost the same as the input graph, except that artificial cost-zero separations have been added so that paths that connect vertices strictly enclosed by faces of the mortar graph to outside vertices are forced to go through the portals.

If a vertex of $MG$ is a vertex of $Q$ (the set of input terminals), we do not consider its copy on the brick to be a terminal vertex. Thus a brick has no terminals.

The following simple lemma follows directly from the fact that each portal edge in $B^+(MG)$ connects a vertex of a brick to the corresponding vertex of $MG$.

**Lemma 3.4.3.** If $A$ is a connected multi-subgraph of $B^+(MG)$ for which there are $k$ edge-disjoint paths connecting terminals $x$ and $y$ in $Q$, then $A - \{\text{portal edges}\}$ is a connected multi-subgraph of $G$ in which there are at least $k$ edge-disjoint paths connecting $x$ and $y$.

The following theorem is central to the proof of correctness of the spanner construction and the approximation scheme. Indeed, taken together, Lemma 3.4.3 and Theorem 3.4.4 provide a reduction from a connectivity problem on $G$ to a connectivity problem on $B^+(MG)$.

**Theorem 3.4.4 (Structure Theorem).**

$$\text{OPT}(B^+(MG), Q) \leq (1 + c\epsilon)\text{OPT}(G, Q)$$

where $c$ is an absolute constant and OPT refers to either the Steiner-tree or the 2-EC problem.

**Proof.** We start with the optimal solution $H$ to the Steiner-tree or 2-EC problem and transform it, in stages, into a solution $\tilde{H}$ to the same problem in $B^+(MG)$, while almost preserving its length: $t(\tilde{H}) < (1 + c'\epsilon)t(H)$ for a fixed constant $c'$. We prove this theorem for the more general 2-EC problem.

First we add 2 copies of the east and west boundaries of each brick. Let $H_1$ be the union of $H$ with 2 copies of the east and west boundaries ($E_B$ and $W_B$) for every brick $B$ in $G$. Using Lemma 3.2.6, we have

$$\ell(H_1) \leq \text{OPT}(G, Q) + 2\epsilon\text{OPT}(G, Q).$$

(3.3)

Then we reduce the number of joining vertices on the north and south boundaries of each brick. Let $H'_1$ be the subgraph of $H_1$ that is strictly embedded in a brick of $G$. Replace $H'_1$ with the
subgraph $H_2'$ that is guaranteed by Theorem 3.3.17. From the third part of the Theorem, we deduce that $\ell(H_2') \leq (1 + c\epsilon)\ell(H_1')$. Repeating this process for every brick of $G$ produces the subgraph $H_2'$. Since the bricks are disjoint, we have

$$\ell(H_2) \leq (1 + c\epsilon)\ell(H_1).$$  \hspace{1cm} (3.4)

Now we map the edges of $H_2$ to a subgraph of $B^+(MG)$. Every edge of $G$ has at least one corresponding edge in $B^+(MG)$. For every edge $e$ of $H_2$, we select one corresponding edge in $B^+(MG)$ as follows: if $e$ is an edge of $MG$ select the corresponding mortar edge of $B^+(MG)$, otherwise select the unique edge corresponding to $e$ in $B^+(MG)$ (according to the multiplicity of $e$ in the multi-subgraph $H_2$). This process constructs a subgraph $H_3$ of $B^+(MG)$ such that

$$\ell(H_3) = \ell(H_2).$$ \hspace{1cm} (3.5)

Since $H_3$ is not connected, we connect it via portal and mortar edges. Let $V_B$ be the set of joining vertices of $H_3$ with $N_B \cup S_B$ for a brick $B$ of $B^+(MG)$. For any vertex $v$ on the interior boundary $\partial B$ of a brick, let $p_v$ be the portal on $\partial B$ that is closest to $v$, let $P_v$ be the shortest $v$-to-$p_v$ path along $\partial B$ and let $P'_v$ be the corresponding path of mortar edges. Let $e$ be the portal edge corresponding to $p_v$. Add 2 copies of each $P_v$, $P'_v$, and $e$ to $H_3$. Repeat this process for every $v \in V_B$ and for every brick $B$, building a graph $\hat{H}$. This completes the definition of $\hat{H}$.

We now need to analyze the length of $\hat{H}$:

$$\ell(\hat{H}) \leq \ell(H_3) + \sum_{B \in B} \sum_{v \in V_B} 2(\ell(P_v) + \ell(e) + \ell(P'_v)),$$ \hspace{1cm} (3.6)

and we have:

$$\sum_{B \in B} \sum_{v \in V_B} 2(\ell(P_v) + \ell(e) + \ell(P'_v)) = 4 \sum_{B \in B} \sum_{v \in V_B} \ell(P_v), \text{ since } \ell(\text{portal edges}) = 0$$

$$\leq 4 \sum_{B \in B} \sum_{v \in V_B} \ell(\partial B)/\theta(\epsilon), \text{ by Lemma 3.4.1}$$

$$\leq 4 \sum_{B \in B} \frac{\alpha(\epsilon)}{\theta(\epsilon)} \ell(\partial B), \text{ by Theorem 3.3.3, Part 2}$$

$$\leq 4 \frac{\alpha(\epsilon)}{\theta(\epsilon)} \nu^{-1}OPT(G,Q), \text{ using Lemma 3.2.8}$$

$$\leq 2cOPT(G,Q), \text{ using Equation (3.2)}.$$ 

Combining this with equations (3.6), (3.5), (3.4) and (3.3), we obtain

$$\ell(\hat{H}) \leq (1 + c\epsilon)(OPT(G,Q) + 2cOPT(G,Q)) + 2cOPT(G,Q) < (1 + (4 + 3c)\epsilon)OPT(G,Q)$$

for the fixed constant $c$ given by Theorem 3.3.17.

It remains to show that $\hat{H}$ is a solution to the connectivity problem in $B^+(MG)$. First we show that $H_3$ is a solution to the corresponding problem in $G$. We give the argument for the 2-EC problem. The argument for the Steiner-tree problem follows similarly.
Clearly, since \( H \) is a subgraph of \( H_1 \) and \( H \) is a solution in \( G \), \( H_1 \) is a solution in \( H \). Now we argue that \( H_2 \) is a solution to the problem in \( G \). Let \( P \) and \( Q \) be edge-disjoint paths in \( H \) that connect terminals \( s \) and \( t \). We partition \( P \) into a sequence of subpaths as follows: \( P_i \) is a subpath of the partition if it is a maximal subpath strictly enclosed by a brick, or if it is a maximal subpath on the boundary of a single brick. Each subpath \( P_i \) is an \( x_i \)-to-\( y_i \) path. For a vertex \( x \) in \( MG \), if \( x \) is a vertex internal to an east boundary \( E_B \) of a brick \( B \), then define \( \hat{x} \) to be the vertex common to \( E_B \) and \( N_B \) (likewise for a vertex internal to a west boundary). If \( x \) is a vertex of a north or south boundary, then define \( \hat{x} = x \). We similarly partition \( Q \) into a sequence of subpaths. Note that for each path \( P_i \) and \( Q_i \), the first step of the construction guarantees that there are corresponding \( \hat{x}_i \)-to-\( \hat{y}_i \) paths in \( H_1 \). Since we added two copies of each east and west boundary, the paths \( P_i \) and \( Q_j \) are edge disjoint for every \( i \) and \( j \). Since \( \hat{x}_i \) and \( \hat{y}_i \) are vertices on \( S_B \cup N_B \) for a brick \( B \), Theorem 3.3.17 guarantees that there are the same number of edge-disjoint \( \hat{x}_i \)-to-\( \hat{y}_i \) paths in \( H_2 \). It follows that there are edge disjoint \( \hat{s} \)-to-\( \hat{t} \) path in \( H_2 \). By Lemma 3.3.1 since \( s \) and \( t \) are terminals of \( Q \), they are on north or south brick boundaries, and so \( \hat{s} = s \) and \( \hat{t} = t \): there are \( s \)-to-\( t \) paths \( \hat{P} \) and \( \hat{Q} \) in \( H_2 \).

The definition of \( H_3 \) breaks \( \hat{P} \) and \( \hat{Q} \) into disjoint paths. Consider one such path, \( \hat{P}_i \), that is not a subpath of \( MG \). By construction, the endpoints of \( \hat{P}_i \) are joining vertices. To go from \( H_3 \) to \( \hat{H} \), these endpoints are connected to the corresponding vertices on \( MG \) via portal edges. It follows that there are \( s \)-to-\( t \) paths \( \hat{P} \) and \( \hat{Q} \) in \( \hat{H} \). Since 2 copies of the portal edges and the subpaths of \( MG \) that are added in constructing \( \hat{H} \), we guarantee that \( \hat{P} \) and \( \hat{Q} \) are edge-disjoint.

\[ \square \]

### 3.5 Approximation via spanner construction

Here we review the framework for designing polynomial-time approximation schemes in planar graphs that Klein proposed in 2005 [68] that has previously been used to give a PTAS for the subset-tour problem in planar graphs [69]. The framework takes a graph \( G \) with edge weights and has four steps: Table 3.4.

Notice that the final solution is a \((1 + \epsilon)\)-approximate solution: The solution in \( G'' \) has value at most \( OPT(G') \), which in turn is at most \((1 + \frac{\epsilon}{2})OPT(G') \). In Step 4, we include at most \( c \) copies of the edges in \( S \), which adds at most \( cw(S) \leq \frac{2}{1/c}w(G') \leq \frac{2}{c}OPT(G) \). The weight of the final solution is bounded by \((1 + \epsilon)OPT(G) \). For the travelling salesman problem \( c = 2 \) [68] since a tour must be maintained. For the Steiner-tree problem \( c = 1 \) since connectivity need only be maintained. For the 2-EC problem \( c = 2 \).

The edges \( S \) found in the thinning step is a minimum-weight set of levels modulo \( 2cf(\epsilon) \) in a breadth-first search tree of \( G' \). This technique was first used by Baker [5]. We will use this technique again in Section 3.6.1. The thinning step is easy to carry out in linear time. The lifting step is also easy to carry out in linear time.

The graph \( G'' \) has radius \( g(\epsilon) \). We can find a tree decomposition of \( G'' \) with width \( 3g(\epsilon) \) in linear time [5, 33]. Since \( G'' \) has low treewidth, the dynamic programming step can be done in linear time.
PTAS Framework for planar graphs (Klein [67])

1. **Spanner Step:**
   Find a subgraph $G'$ of $G$ with two properties:
   
   (S1) $w(G') \leq f(\varepsilon)OPT(G)$, and
   (S2) $OPT(G') \leq (1 + \frac{\varepsilon}{2})OPT(G)$

   where $OPT(G)$ is the value of the optimal solution in $G$.

2. **Thinning Step:**
   Use a Baker-esque technique to select a set $S$ of edges of $G'$ such that for some constant $c$ and
   
   (a) $w(S) \leq \frac{\varepsilon}{2cf(\varepsilon)}w(G')$, and
   (b) $G''$, the graph obtained from $G'$ by contracting the set of edges $S$, has radius $g(\varepsilon) = poly(f(\varepsilon))$.

3. **Dynamic Programming Step:**
   Solve the problem optimally in the low-treewidth graph $G''$.

4. **Lifting Step:**
   Use at most $c$ copies of each edge of $S$ to transform the solution obtained in Step 3 for $G''$ into a solution for $G'$ and so for $G$.

Table 3.4: A framework for designing polynomial-time approximation schemes in planar graphs using a spanner.

The framework allows us to concentrate on finding a light spanner: a subgraph that satisfies the *shortness property* (as given by Property (S1) of the Spanner Step) and the *spanning property* (as given by Property (S2) of the Spanner Step). In Section 3.5.1 we give an $O(n \log n)$-time construction for a subgraph that we show has the shortness and spanning properties in Section 3.5.2.

### 3.5.1 Spanner Construction

The input to the spanner construction is a set of terminals $Q$ of the planar graph $G$ and our given precision $\epsilon$. To construct the spanner for 1/2-connectivity problems, we first compute the brick decomposition. Then for each brick, we exhaustively compute optimal Steiner trees that span a subset of the portals for the brick.

Since the number of terminals of each Steiner-tree problem solved here is at most $\theta$ by the Cardinality Property, Theorem 3.1.1 implies that the running time of Step 5 is $O(2^\theta \theta^3 n')$ where $n'$ is the number of vertices in the brick $B$.

Combining the running time of Step 5 with the running times of Steps 1 through 3, gives a total running time for constructing the spanner of $O(2^\theta \theta^3 n + n \log n)$ which is $O(2^{poly(1/\epsilon)} n + n \log n)$ since $\theta$ depends polynomially on $1/\epsilon$. 
Table 3.5: The input to the spanner algorithm for either the Steiner-tree or 2-EC problem is a planar graph $G$, the set of terminals $Q$ and the precision $\epsilon$.

### 3.5.2 Correctness

To complete the analysis, we prove that the output set of edges satisfy the shortness and spanning properties.

**Lemma 3.5.1** (Shortness property). The total length of the output edges is at most

$$(1 + 2^{1+\theta})5\epsilon^{-1}\text{OPT}(r, G).$$

**Proof.** For each brick $B$, the length of each Steiner tree is bounded by $\ell(\partial B)$. By construction of the bricks, each edge occurs at most twice as the edge of a brick boundary, so the sum of brick boundary lengths is at most $2\ell(MG)$. For each brick $B$, the Cardinality Property for $B$ implies that Step 5 finds at most $2^{\theta}$ Steiner trees, each of length at most $\ell(\partial B)$, so the total length of all Steiner trees found, summing over bricks $B$, is at most $2^{1+\theta} \cdot \ell(MG)$. The output also includes each edge of the mortar graph, so the total length of the output is at most $(1 + 2^{1+\theta}) \cdot \ell(MG)$. Appealing to the Lemma 3.2.8, which bounds the length of the mortar graph, completes the proof.

Since $\theta$ is polynomial in $1/\epsilon$, the treewidth of the graph found in the framework is $g(\epsilon) = 2^{\text{poly}(1/\epsilon)}$. Since the dynamic programming step is exponential in this treewidth, the dependence of the running time of the PTAS on poly$(1/\epsilon)$ is doubly exponential.

**Lemma 3.5.2** (Spanning property). The output contains a subgraph that satisfies the connectivity requirements $r$ between vertices in $Q$ and has length at most $(1 + \epsilon)\text{OPT}(G, Q)$ where $c$ is a constant.

**Proof.** By the Structure Theorem (Theorem 3.4.4), there exists a solution $H$ in $B^+(MG)$ that has length at most $(1 + \epsilon)\text{OPT}(G, Q)$. By Lemma 3.3.16, the subgraph $H_B$ of $H$ that is enclosed by a brick $B$ is the union of trees. Replacing one such tree $T$ with another $T'$ that maintains the boundary connectivity results in a subgraph that achieves the boundary 2-connectivity of $H$ (by (T3) of Lemma 3.3.16). As a consequence of the Structure Theorem, the leaves of $T$ are portals. Let $T''$ be the optimal Steiner tree in $B$ that spans the leaves of $T$.

Let $H'$ be the subgraph resulting from all these replacements: we have $\ell(H') \leq \ell(H) \leq (1 + \epsilon)\text{OPT}(G, Q)$. By Lemma 3.4.3, the edges of $H'$, not including the portal edges, form a solution to the problem in $G$. 

---

**ConnectivitySpanner$(G, Q, \epsilon)$**

1. Let $MG = \text{BrickDecomposition}$(G, Q, $\epsilon$).
2. Compute the set of bricks $B$ corresponding to $MG$.
3. Designate portals (Section 3.4.1) for each brick.
4. For each brick $B$ and each subset of vertices $X$ of portals:
   5. Compute the optimal Steiner tree spanning $X$ in $B$.
6. Return the union of all the edge sets found in Step 5, together with the edges of $MG$. 

---


3.6 Approximation via the brick decomposition

In the last section we used the brick decomposition as a scaffolding for finding a spanner. Using this spanner and the PTAS framework (Table 3.4) yields a running time whose dependence on \( poly(1/\epsilon) \) is doubly exponential [15]. Here we will merge the thinning and dynamic programming steps of the framework to design an algorithm that is only singly exponential in \( poly(1/\epsilon) \) [19].

We perform thinning on the mortar graph thereby grouping the bricks into subgraphs called parcels (Section 3.6.1). We use the bound on the number of portals per brick to design a dynamic program (Section 3.7) to solve the connectivity problem in a parcel. In order to guarantee connectivity between terminals in different parcels, we introduce some new terminals on each parcel (Section 3.6.2).

In this section we will assume without loss of generality that the degree of every vertex is 3 in our input graph.

3.6.1 Parcels

Here we describe the decomposition of \( MG \) into subgraphs called parcels. The construction uses a positive integer parameter \( \eta \), depending polynomially on \( \epsilon \), whose value

\[
\eta = \eta(\epsilon) = \lceil 20\epsilon^{-2} \rceil
\]

is used at the end of Section 3.7.5.

A linear-time algorithm for this step is given in Table 3.6. The basic idea is to use breadth-first search in \( MG^* \), the dual of \( MG \), together with the shifting technique of [5]. Each parcel is a planar embedded subgraph of \( MG \). Each edge is in at most two parcels. A boundary edge is one that belongs to two parcels. We denote by \( S \) the set of edges that are on parcel boundaries.

The parcel decomposition has the following two properties:

**Radius Property:** The planar dual of each parcel has a spanning tree of depth at most \( \eta + 1 \) (Lemma 3.6.2).

**Parcel-Boundary Length Property:** The sum of the lengths of the boundaries of the parcels is at most \( \ell(MG)/\eta \) (Lemma 3.6.4).

The radius property makes it possible to compute an optimal solution within a parcel in polynomial time. Our plan is to compute an optimal solution within each parcel and glue these together using the parcel boundaries to obtain a near-optimal solution for the original graph. To ensure that the parcel solutions form a connected subgraph of the original graph, we introduce new terminals on the boundaries of the parcels. The details of this are given in Section 3.6.2.

It should be clear from the algorithm that the running time is \( O(n) \). We call the set returned by \textsc{ParcelDecomposition} the set of parcels of \( MG \). We will now concentrate on showing that the parcels are planar graphs with the radius and parcel-boundary length properties. In order to do so, we need some additional notation. For an interval such as \([i, j]\) the subgraph of \( MG^* \) induced
PARCELDECOMPOSITION(\(MG, \eta\))

1. Let \(v_{\text{root}}\) be a vertex of \(MG^*\) such that there is a terminal (such that \(r(v_{\text{root}}) = 2\) for the 2-EC problem) on the boundary of the face \(v_{\text{root}}\) in \(MG\).
2. Do breadth-first search in \(MG^*\) starting from \(v_{\text{root}}\) to find the set of edges whose endpoints have breadth-first search distance \(i\) and \(i + 1\), \(E_i\).
3. For \(k = 0, 1, \ldots, \eta - 1\), let \(E_k = E_k \cup E_{k+\eta} \cup E_{k+2\eta} \cup \cdots\). Let \(k^*\) be the index that minimizes \(\ell(E_k)\).
4. Let \(\mathcal{Y}\) denote the set of connected components of \(MG^* - E_{k^*}\). For each \(Y \in \mathcal{Y}\), let \(H_Y\) denote the subgraph of \(MG\) consisting of the boundaries of faces in \(V(Y)\).
5. Return \(\mathcal{H} = \{H_Y : Y \in \mathcal{Y}\}\).

Table 3.6: A thinning technique on the input mortar graph finds a set of subgraphs of the mortar graph called parcels.

by vertices whose breadth-first search distance (or level) from \(v_{\text{root}}\) is in \([i, j]\) is denoted \(MG^*[i, j]\). For a positive integer \(i\), let \(K_i\) be the set of connected components of \(MG^*[i, \infty]\). Let \(K\) be one such connected component. The edges of \(\Gamma(V(K))\) form a bond in \(MG^*\) and by cycle-cut duality (Theorem 1.2.2) form a simple cycle in \(MG\). We denote this cycle by \(C_K\).

Lemma 3.6.1. Two vertices \(u\) and \(v\) of \(MG^*\) whose levels are in \([i, j]\) are connected in \(MG^*[i, j]\) iff they are connected in \(MG^*[i, \infty]\).

Proof. Let \(P\) be a \(u\)-to-\(v\) path in \(MG^*[i, \infty]\), chosen to minimize the number of vertices of \(P\) that are at levels greater than \(j\). Suppose for a contradiction that this number is positive, and let \(P'\) be a maximal subpath of \(P\) consisting of vertices at levels greater than \(j\). Let \(K\) be the connected component of \(MG^*[j + 1, \infty]\) containing \(P'\). The edges of \(C_K\) form a simple cycle \(e_1 e_2 \cdots e_g\) in \(MG\). Let \(f_j\) be the face whose corresponding vertex in \(MG^*\) is at level \(i\) and whose boundary includes \(e_j\). Note that the vertices of \(P\) just before and just after \(P'\) are among \(f_1, \ldots, f_g\).

For \(j = 1, \ldots, g - 1\), since the common endpoint of \(e_j\) and \(e_{j+1}\) in \(MG\) has degree at most three, either \(f_j\) and \(f_{j+1}\) are the same or they share an edge on their boundaries. This shows that \(f_1, \ldots, f_g\) (which are vertices of \(MG^*\)) are connected, which contradicts the choice of \(P\).

Lemma 3.6.2 (Radius Property). The planar dual of each parcel has a spanning tree of depth at most \(\eta + 1\).

Proof. Let \(P_v\) denote the path in \(MG^*\) from vertex \(v\) up the breadth-first search tree to the root \(v_{\text{root}}\).

Consider a connected component \(Y\) of \(MG^* \setminus E_{k^*}\) and the corresponding parcel \(H_Y\). Write \(H = MG \setminus A\) where \(A\) is a set of edges. The graph \(H\) consists of the boundaries of faces of \(MG\) that correspond to vertices of \(Y\). Thus \(A\) includes all edges of \(MG^*[0, i - 1]\) and all edges of \(MG^*[i + \eta + 1, \infty]\). We have \(H^* = MG^*/A\) which denotes the contraction of the edges in \(A\). For every vertex \(v\), the path \(P_v/A\) is in \(H^*\). Moreover, the level of \(v\) is at most \(i + \eta\), so \(P_v\) has at most \(\eta\) edges not belonging to \(MG^*[0, i - 1]\).
Since $H$ consists of the boundaries of faces that are vertices of $Y$, any vertex of $H^*$ that is not a vertex of $Y$ is adjacent to a vertex of $Y$. This shows that the radius of $H^*$ is at most $\eta + 1$.

Recall that an edge of $MG$ is a \textit{boundary edge} if it belongs to two parcels. We denote the set of all such edges by $\partial H$.

\textbf{Lemma 3.6.3.} \\
$\partial H = E_{k^*} = \bigcup\{C_K : K \text{ is a connected component of } MG^*[i + 1, \infty), i \geq 1, i \equiv k^* \mod \eta]\}$

\textit{Proof.} Let $e$ be an edge of $MG^*$. Consider the following three cases

$e \in \partial H$: There are two connected components $Y_1$ and $Y_2$ of $MG^* - E_{k^*}$ such that, in $MG^*$, one endpoint of $e$ is in $Y_1$ and the other is in $Y_2$. If $e$ were not in $E_{k^*}$ then $Y_1$ and $Y_2$ would be connected in $MG^* - E_{k^*}$, a contradiction. Thus $e \in E_{k^*}$.

$e \in E_{k^*}$: For some integer $i$ such that $i \equiv k^* \mod \eta$, $e \in E_i$. Let $K$ be the connected component of $MG^*[i + 1, \infty)$ containing the endpoint in $MG^*$ of $e$ that has level $i + 1$. Then $e \in C_K \subseteq \Gamma(V(K))$.

$e \in C_K \subseteq \Gamma(V(K)): (K$ is a connected component of $MG^*[i + 1, \infty)$ and $i \equiv k^* \mod \eta)$.) Let $Y_1$ be the connected component of $MG^* - E_{k^*}$ that contains the endpoint of $e$ in $V(K)$, and let $Y_2$ be the component that contains the endpoint not in $V(K)$. By definition of $K$, the first endpoint has level $i + 1$ and the second endpoint has level at most $i$ (in fact exactly $i$). Thus $e \in E_i$. By breadth-first search, every path between a vertex at level $i + 1$ and a vertex at level at most $i$ contains some edge of $E_i$, so $Y_1$ and $Y_2$ must be \textit{distinct} connected components of $MG^* - E_{k^*}$. Hence $e$ belongs to both parcel $H_{Y_1}$ and $H_{Y_2}$.

$\blacksquare$

\textbf{Lemma 3.6.4} (Boundary Length Property). $\ell(\partial H) \leq \ell(MG)/\eta$.

\textit{Proof.} It follows from the choice of $k^*$ that $\ell(E_{k^*}) \leq \ell(MG)/\eta$. The proof follows from Lemma 3.6.3.

$\blacksquare$

We have shown that the parcels have the required properties. We will need the following lemma for the introduction of new terminals in the next section.

\textbf{Lemma 3.6.5.} \textit{The following three sets are equal:}

1. $\{\text{connected components of the subgraph } \partial H\}$,

2. $\{\text{connected components of the boundary edges of } H : H \in \mathcal{H}\}$, and

3. $\{C_K : K \text{ a connected component of } MG^*[i + 1, \infty), i \geq 1, i \equiv k^* \mod \eta]\}$

\textit{Proof.} Lemma 3.6.3 shows that the set of edges given by the union of Set 1, the union of Set 2, and the union of Set 3 are all equal. We show the following:
Set 3 equals Set 1: Clearly, each cycle $C_K$ is connected and they are edge-disjoint. If two distinct cycles $C_K$ and $C_{K'}$ shared a vertex $u$, then $u$ would have degree at least 4 in $MG$, contradicting our assumption that our input graph has degree 3.

Set 2 equals Set 3: Let $i$ be positive integer such that $i \equiv k^* \pmod{\eta}$, and let $K$ be a connected component of $MG^*[i + 1, \infty)$. Let $Y$ be the intersection of $K$ with $MG^*[i + 1, i + \eta]$. By Lemma 3.6.1, $Y$ is connected and so is a connected component of $MG^* - \mathcal{E}_k$. It follows that the edges of $C_K$ are boundary edges of $H_Y$. By Part (A), $C_K$ is a connected component of the boundary edges of $H_Y$. This proves $C_K$ is a member of Set 2.

Let $u$ and $v$ be level-$i$ endpoints in $MG^*$ of edges of $C_K$. By Lemma 3.6.1, $u$ and $v$ are connected in $MG^*[i - \eta + 1, i]$. This shows that all the level-$i$ endpoints of edges of $C_K$ belong to a common connected component $Y'$ of $MG^*[i - \eta + 1, i]$. Thus the edges of $C_K$ are boundary edges of $H_{Y'}$. As before, Part (A) implies that $C_K$ is a connected component of the boundary edges of $H_{Y'}$.

We have shown that $C_K$ is a connected component of boundary edges of each of two parcels. This proves that no proper subset of $C_K$ is a member of Set 2.

3.6.2 New requirements

The next step is to select some new terminals and requirements to ensure that the solution we find in each parcel will combine to form a solution to the problem in the original graph. The parcel-boundary length property ensures that connecting to these new terminals does not increase the length of the optimal parcel solutions by much.

We select the new terminals according to the following terminal-selection rule:

For each parcel $H$ and for each connected component $C$ of the boundary of $H$, if $\mathcal{B}^+(MG) - V(C)$ disconnects some terminals $x$ and $y$, then designate a vertex $z$ of $C$ as a new terminal. For the 2-EC problem, the requirement $r(z)$ is 2 if $\mathcal{B}^+(MG) - V(C)$ separates terminals with 2-requirements and 1 otherwise.

Notice that the new terminals are vertices of the mortar graph, not of the bricks. We will show that the new terminals satisfy the following properties:

Spannable Property: Let $H$ be a subgraph of $\mathcal{B}^+(MG)$ that is a solution to the problem for the original terminals. Let $P$ be a parcel. Then $H \cup \partial H$ contains a solution to the problem for the original and new terminals in $\mathcal{B}^+(P)$ (Lemma 3.6.6).

Connecting Property: For each parcel $P$, let $H_P$ be a subgraph of $\mathcal{B}^+(P)$ that solves the problem for the original and new terminals in $P$. Then $\bigcup P H_P \cup \partial H$ contains a connected subgraph of $\mathcal{B}^+(MG)$ that solves the problem for the original terminals (Lemma 3.6.7).
In Table 3.7, we give a linear-time algorithm for selecting the new terminals and requirements and we prove that they satisfy the connecting property. Together with the parcel-boundary length property (along with the definition of $\eta$ and the mortar-graph length property), this reduces the connectivity problem for the original terminals in $B^+(MG)$ to the connectivity problem for the parcels and the induced subsets of old and new terminals.

We can restate the terminal-selection rule as:

For each connected component $X$ of $\partial H$ (the set of edges found in step 3 of ParcelDecomposition), if $B^+(MG) - V(X)$ disconnects some terminals $x$ and $y$, then designate a vertex $z$ of $X$ as a new terminal.

Let $t$ be a terminal on the boundary of $v_{\text{root}}$ (as guaranteed by step 1 of ParcelDecomposition). If terminals $x$ and $y$ are disconnected by a component $X$ of $S$, then $t$ cannot be connected to both $x$ and $y$ upon removal of $X$. Using this observation, we can modify the condition, yielding another restatement of the terminal-selection rule:

For each connected component $X$ of $S$, if some original terminal is not connected to $t$ in $B^+(MG) - V(X)$, then designate a vertex of $X$ as a new terminal. For the 2-EC problem, the requirement $r(z)$ is 2 if $B^+(MG) - V(C)$ separates terminals with 2-requirements and 1 otherwise.

The algorithm NEWREQUIREMENTS implements this last restatement in linear time. An illustration of new terminal selection is shown in Figure 3.19.

<table>
<thead>
<tr>
<th>NEWREQUIREMENTS($\partial H$, $MG$, $v_{\text{root}}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Compute the connected components of $S$.</td>
</tr>
<tr>
<td>2. In $MG$, contract each connected component into a single vertex.</td>
</tr>
<tr>
<td>3. Compute the biconnected components and the block/cut-vertex tree. Root it at the biconnected component containing the terminal face $v_{\text{root}}$.</td>
</tr>
<tr>
<td>4. For each connected component $X$ of $S$, let $k$ be the maximum requirement $r(s)$ of vertices $s$ in the subtree rooted at the cut-vertex corresponding to $X$, designate a vertex $v$ of $X$ as new terminal and assign $r(v) = k$.</td>
</tr>
</tbody>
</table>

Table 3.7: We compute the new terminals and their requirements in the mortar graph on the boundaries of parcels, $\partial H$.

**Lemma 3.6.6.** The new terminals satisfy the spannable property.

**Proof.** Let $H$ be a solution to the connectivity problem and let $P$ be a parcel.

Start from the edges $H \cap B^+(P)$, and for each connected component $C$ of $\partial P$ that has a new terminal on it, add the edges of $C$. The resulting set of edges is still connected because, by definition of new terminals, $H$ must intersect $C$. For the 2-EC problem, consider an original terminal $x$ in $P$ and a new terminal $z$ on $C$. By the terminal selection rule, $C$ separates $x$ from a terminal $y$ not in
Figure 3.19: On the left, the nested cycles that form the boundaries of the parcels are pictured. On the right, we see that parcels drawn according to their level in the breadth-first search. Given original terminals $x$, $y$ and $z$ (left and right), new terminals $a$ and $b$ must be introduced to connect these through the parcel boundaries (right). Note that when the parcels are separated (as shown on the right), we get two copies of each terminal that is on the boundary of a parcel.

$x$. There are edge disjoint paths from $x$ to $C$ in $H$, and so there are edge disjoint paths from $x$ to $z$ in $H \cup C$. The lemma follows from the transitivity of 2-edge connectivity.

Consider the face $v_{\text{root}}$ (the root of the breadth-first search tree) as the infinite face of the embedding of $MG$. We can now prove the Connecting Property of the new terminals.

**Lemma 3.6.7.** For each parcel $P$ that contains a terminal, let $H_P$ be a solution in $B^+(P)$ connecting the original and new terminals belonging to $P$. Then $\bigcup P H_P$ is a solution in $B^+(MG)$.

**Proof.** We prove by induction on $i$ that, for every positive integer $i$ such that $i \equiv k^* \pmod \eta$, the union over the parcels $P$ corresponding to connected components of $MG^*[0,i] - \mathcal{E}_k^*$ of the solutions $H_P$ is a solution in $B^+(MG)$.

Suppose $i = 1$. Then the vertices of $MG^*[i - \eta + 1,i]$ are connected via the first few levels of the breadth-first search tree of $MG^*$, so $|K_i| = 1$. There is only one component to the solution.

Suppose $i > 1$. Consider a parcel $P$ that has a terminal, where $P \subseteq K \in K_i$. We claim that one of the vertices of $C_K$ is a new terminal. $C_K$ separates the interior of $C_K$ from the root. $C_K$ corresponds to a cut-vertex $x$ of the block-cut-vertex tree. $P$ corresponds to part of a subtree rooted at $x$. Since $P$ has a terminal, that subtree does have a terminal, so $C_K$ has a new terminal.

Let $v$ be this terminal. Then $v$ belongs to the solution $H_P$. However, $v$ is a vertex of some parcel $P'$ where $P' \subseteq K' \in K_{i-\eta}$, and hence to $H_{P'}$, so adding $H_P$ preserves the required connectivity.

### 3.7 Dynamic program

In this section, we give a dynamic problem to find an optimal solution to the Steiner-tree and 2-EC problems in the graph $B^+(P)$ where $P$ is a parcel which in turn is a subgraph of the mortar graph. Recall that $B^+(P)$ is obtained by embedding bricks in those faces of $P$ for which bricks are defined and connecting each brick to the corresponding face of $P$ via at most $\theta$ portal edges. Once we have solved the problem optimally in each parcel, we can union the solutions over all parcels to find a
solution \( H \) in \( \mathcal{B}^+(MG) \). Our final solution is the union of the mortar and brick edges of \( H \). The correctness (Section 3.7.5) of the PTAS follows from Lemmas 3.6.6 and 3.6.7.

We first define a binary recursion tree \( \hat{T} \) that is a (non-spanning) subgraph of \( \mathcal{B}^+(P) \) (Section 3.7.1). For each vertex \( v \) of \( \hat{T} \) we bound the number of edges in \( \mathcal{B}^+(P) \) between the subtree of \( \hat{T} \) rooted at \( v \) and the rest of the graph (Lemma 3.7.2). The leaves of \( \hat{T} \) correspond to bricks. We show how to solve this base case separately for the Steiner-tree (Section 3.7.3) and 2-EC (Section 3.7.4) problems. For non-leaf vertices of \( \hat{T} \), the procedure for filling the dynamic programming table is common to both problems (Section 3.7.2).

We show that the dynamic program runs in \( O(2^{poly(1/\epsilon)}m) \) time, where \( m \) is the number of edges in the parcel. The overall running time of the PTAS (including the construction of the mortar graph and parcels, and selection of portals) is \( O(n \log n + 2^{poly(1/\epsilon)}n) \). Note that the PTAS is singly exponential in \( poly(1/\epsilon) \).

### 3.7.1 Defining the recursion tree

The recursion tree is common to both the Steiner-tree and 2-EC problems.

In order to define the recursion tree, we need an operation called brick contraction. The operation is applicable to a graph \( \mathcal{B}^+(H) \) obtained from a graph \( H \) by applying the brick-insertion operation from Section 3.4.1. Brick contraction is denoted \( \mathcal{B}^+(\mathcal{B}^+(H)) \). Starting with \( \mathcal{B}^+(H) \), contract each brick to a single vertex, called a brick vertex, as illustrated by Figure 3.20. The mortar edges of \( \mathcal{B}^+(H) \) are unaffected by this operation. Note that the degree of each brick vertex is the number of portals for the corresponding brick, which is at most \( \theta \) by Lemma 3.4.1. The graph \( \mathcal{B}^+(\mathcal{B}^+(H)) \) differs from \( H \) in that it has a single vertex connected to \( H \) via portal edges rather than a brick.

![Figure 3.20](attachment:image.png)

Figure 3.20: Given a portal-connected graph (a), we perform the brick contraction operation by contracting each brick to a single vertex, creating (b).

Consider a parcel \( P \). By Lemma 3.6.2, the dual graph \( P^* \) has a breadth-first search tree \( T^* \) of depth at most \( \eta + 1 \). Let \( T \) be the set of edges of \( P \) not in \( T^* \). \( T \) is a spanning tree of \( P \) by the Interdigitating Trees Theorem. For each face of \( P \) that is the mortar boundary of a brick \( B \), let \( e_B \) be the portal edge corresponding to the first portal vertex selected for \( B \) (see Table 3.3). Let \( \hat{T} := T \cup \{ e_B : B \text{ is a brick} \} \). Observe that \( \hat{T} \) is a spanning tree of \( \mathcal{B}^+(\mathcal{B}^+(H)) \) and that the brick vertices are leaves of \( \hat{T} \).

Recall that we have assumed for Section 3.6 that the degree of our input graph is 3.
Lemma 3.7.1. The degree of $\hat{T}$ in $B^+(B^+(H))$ is at most 3.

Proof. Note that each vertex of $\hat{T}$ is either a vertex of $P$ or the result of a brick contraction.

If a vertex $v$ of $\hat{T}$ is obtained by contracting a brick $B$, then $e_B$ is the only edge in $\hat{T}$ adjacent to $v$: $v$ has degree 1.

If $v$ is a vertex of $P$, then certainly $v$ has degree at most 3 in our input graph. So $v$ has degree at most 3 in both $P$ and $T$, which are subgraphs of our input graph. Since the first portal for a brick $B$ has degree 3 as guaranteed by Step 1 of PORTALSELECTION (Table 3.3), one adjacent edge is enclosed by $\partial B$ and the other two adjacent edges are in $\partial B$. Therefore a vertex can only be the first portal for one brick. If $v$ is the first portal chosen for brick $B$, then $v$ has degree at most 2 in $P$, and the addition of $e_B$ gives degree at most 3.

Root $\hat{T}$ at a non-brick-vertex of degree at most two. This tree will guide our dynamic program. For each vertex $v$ of $\hat{T}$, let $\hat{T}(v)$ denote the subtree of $\hat{T}$ rooted at $v$. We will use the following lemma to bound the interaction between subproblems. This property is similar to carving width [97].

Lemma 3.7.2. In the graph $B^+(B^+(H))$, there are at most $2\theta \eta + 1$ edges between $\hat{T}(v)$ and the rest of the graph.

Proof. Let $\hat{T}^*$ be the set of edges of $B^+(B^+(H))$ not in $\hat{T}$. Comparing $\hat{T}^*$ to the spanning tree $T^*$ of $P^*$, we see that each vertex of $T^*$ obtained by contracting a brick $B$ corresponds to a path in $\hat{T}^*$ consisting of all but one of the portal edges associated with $B$. Such a path has at most $\theta - 1$ edges. Since the depth of $T^*$ is at most $\eta$ (Lemma 3.6.2), the depth of $\hat{T}^*$ is at most $\theta \eta$.

Now we use an argument from [67]. Let $v$ be any vertex of $\hat{T}$ other than the root, and let $e_v$ be the edge connecting $v$ to its parent. Then $e_v$ is not in the tree $\hat{T}^*$. The path in $\hat{T}^*$ between the endpoints of $e_v$ has at most $2\theta \eta$ edges. Combining this path with $e_v$ yields a simple cycle in the dual graph $(B^+(B^+(H)))^*$ having at most $2\theta \eta + 1$ edges. The edges of this cycle are exactly the edges in the cut $\Gamma(V(\hat{T}(v)), B^+(B^+(H)))$ in the primal graph (Theorem 1.2.2).

3.7.2 The dynamic programming table

Here we relate the tree $\hat{T}$, which spans the brick-contracted parcel, to the brick-copied parcel. For each vertex $v$ of $\hat{T}$, define

$$f(v) = \begin{cases} B & \text{if } v \text{ is the result of contracting a brick } B \\ v & \text{otherwise} \end{cases}$$

and define $W(v)$ to be the subgraph of $B^+(P)$ induced by $\bigcup \{ f(w) : w \in \hat{T}(v) \}$. It follows from Lemma 3.7.2 that the cut $\Gamma(V(W(v)))$ in $B^+(P)$, which is equal to the cut $\Gamma(V(\hat{T}(v)))$ in $B^+(B^+(P))$, has at most $2\theta \eta + 1$ edges.

For a set of edges $L$ forming a cut, we will define a set of configurations $K_L$. Each configuration is a relationship on the edges in $L$. For the 2-EC problem we will assume the edges in $L$ are...
duplicated to simplify the presentation of the dynamic program. We say the configuration \( K_L \in K_L \) is connecting if
\[
|K_L \cap L| \geq \begin{cases} 
2 & \text{if } L \text{ separates terminals } x \text{ and } y \text{ such that } r(x) = r(y) = 2 \\
1 & \text{if } L \text{ separates terminals.}
\end{cases}
\]
(For the Steiner-tree problem, only the second condition applies.) We say configurations \( K_A \in K_A \) and \( K_B \in K_B \) are compatible if for every edge \( e \in A \cap B \) either \( e \in K_A \cap K_B \) or \( e \notin K_A \cup K_B \).

We will define what is means for a subgraph of \( B^+(P) \) to meet a configuration separately for Steiner-tree and 2-EC. The entry \( \text{Tab}_e[K] \) is the weight of the subgraph induced by the vertices \( W(v) \) of a minimum-weight subgraph that meets configuration \( K \). Note that the weight of no edge in \( \Gamma(V(W(v))) \) is included in \( \text{Tab}_e \). We present the common procedure \( \text{Tab}_e \) for filling the dynamic programming table for a non-leaf node \( v \) of the recursion for the Steiner-tree and 2-EC problems separately.

<table>
<thead>
<tr>
<th>( \text{Tab}_{u_0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Initialize each entry of ( \text{Tab}_{u_0} ) to ( \infty ).</td>
</tr>
<tr>
<td>2. Let ( u_1, \ldots, u_s ) be the children of ( u_0 ).</td>
</tr>
<tr>
<td>3. For every set of connecting, mutually compatible configurations ( K, K_1, \ldots, K_s ),</td>
</tr>
<tr>
<td>4. For every connecting configuration ( K_0 ) that is consistent with ( K, K_1, \ldots, K_s ),</td>
</tr>
<tr>
<td>5. ( L = w(K \cap (\cup_{i=1}^s K_i)) + w(\cap_{i=1}^s K_i) + \sum_{i=1}^s \text{Tab}_{u_i}[K_i] ).</td>
</tr>
<tr>
<td>6. ( \text{Tab}<em>{u_0}[K_0] := \min{\text{Tab}</em>{u_0}[K_0], L} ).</td>
</tr>
</tbody>
</table>

Table 3.8: This procedure fills the table \( \text{Tab}_{u_0} \) for vertices \( u_0 \) that are not leaves of the recursion tree, \( \tilde{T} \). In the above we use the following shorthand: \( K \) for \( K_{\Gamma(\{u_0\})} \), and \( K_u \), for \( K_{\Gamma(V(W(u)))} \). The cuts are with respect to the graph \( B^+(P) \).

### 3.7.3 Steiner tree

We now give the definitions of configuration, consistent, and meet that are specific to the Steiner-tree problem. We will then give a procedure for solving the base cases of the dynamic program and bound the running time of the dynamic program for the Steiner-tree problem.

Let \( U \) be a set of vertices and let \( E = \Gamma(U) \). A configuration \( K_E \) corresponding to \( E \) is a non-crossing sub-partition of \( E \). The edges in \( E \) form a cycle \( e_1 \circ e_2 \circ \cdots \) in the dual (by cycle-cut duality). The cycle induces a cyclic ordering of the edges of \( E \). A partition is non-crossing if no two sets of the partition cross each other. Set \( S_1 \) crosses set \( S_2 \) if \( e_i, e_j \in S_1 \) and \( e_k, e_l \in S_2 \) and these edges are in the order \( e_i \preceq e_k \preceq e_j \preceq e_l \) in \( E \). (See Figure 3.21.)

We say that two configurations \( K_E \) and \( K_L \) are consistent if for an edge \( e \in E \cap L \), \( e \) is either in both \( K_E \) and \( K_L \) or in neither.

Let \( H \) be a subgraph induced by \( U \cap V(E) \). We say that \( H \) meets a configuration \( K_E \) if for every set \( S \in K_E \), \( S \) is a subset of a connected component of \( H \).
Solving the base case

Suppose \( v \) is a leaf of the recursion tree. We show how to fill \( \text{Tab}_v \).

If \( v \) does not correspond to a brick (i.e. \( f(v) = v \)), the problem is trivial. Since the weight of any subgraph that is induced by \( \{v\} \) is zero, the value of \( \text{Tab}_v[K] \) is zero for every such configuration \( K \).

Suppose then that \( v \) corresponds to a brick \( B \) (i.e. \( f(v) = B \)). Let \( K \) be a configuration corresponding to the edges of the cut \( \Gamma(V(W(v))) \) in \( B^+(P) \). This is the set of portal edges corresponding to brick \( B \). Let \( H_K \) be the minimum-weight subgraph of \( B \cup \Gamma(V(W(v))) \) that meets the configuration \( K \). Let \( a \) be an edge of \( \Gamma(V(W(v))) \) and let \( x_a \) be the endpoint of \( a \) in \( B \). \( H_K \) is the union of Steiner trees whose leaves correspond to the sets in \( K \) (taking the endpoints of these edges in brick \( B \)). We assign \( \text{Tab}_v[K] \) the sum of the weights of these trees.

Since each set in a configuration has at most \( 2\theta + 1 \) elements, we can compute each Steiner tree in \( O(\theta^3 n_B) \) time where \( n_B \) is the number of vertices of the brick (Theorem 3.1.1). Since there are at most \( 2\theta + 1 \) sets in the configuration, we can compute \( \text{Tab}_v[K] \) in \( O(\theta^4 n_B) \) time.

Running time

The number of non-crossing partitions of an \( n \) element ordered set is the \( n^{th} \) Catalan number, which is at most \( 4^n/(n+1) \) [21, 83]. Therefore, the number of non-crossing sub-partitions is at most \( 4^n \).

It follows that the time to populate \( \text{Tab}_v \) for \( v \) a brick vertex is \( O(\theta^4 \theta^n n_B) \) which is \( O(2^{poly(1/\epsilon)} n_B) \) since \( \theta \) depends polynomially on \( 1/\epsilon \). Since a vertex appears at most twice in the set of bricks, the time needed to solve all the base cases is \( O(2^{poly(1/\epsilon)} n) \) where \( n \) is the number of vertices in the parcel.

For \( v \) not a leaf of the recursion tree, the number of edges in \( \Gamma(V(W(v))) \) in \( B^+(P) \) is at most \( 2\theta \eta + 1 \) (Lemma 3.7.2). It follows that the corresponding number of configurations is \( O(2^{poly(1/\epsilon)}) \) since \( \theta \) and \( \eta \) each depend polynomially on \( 1/\epsilon \). For a recursion tree with \( n \) vertices, the time
required for the dynamic program, not including the base cases is $O(2^{\text{poly}(1/\epsilon)} n)$.

The total running time of the dynamic program is $O(2^{\text{poly}(1/\epsilon)} n)$ where $n$ is the number of vertices in the parcel.

### 3.7.4 2-EC

Given a graph $G$, let $EC(G)$ denote the graph whose nodes are the 2-edge-connected components of $G$ and whose edges denote adjacency of these components in $G$. This structure is familiarly known as a block-cut tree [36]. It is easy to see that $EC(G)$ is a forest. Let $\tilde{EC}(G)$ be the graph obtained from $EC(G)$ by replacing every maximal path having internal degree 2 vertices with an edge.

We call the edge incident to a leaf of a tree a leaf edge.

Let $U$ be a set of vertices and let $E = \Gamma(U)$. We define a configuration $K_E$ corresponding to $E$. First we disregard adjacencies between edges of $E$ that have a common endpoint in $\bar{U}$, creating a new edges set $E'$: if two edges $a$ and $b$ of $E$ have a common endpoint $x$ in $\bar{U}$ (the complement of $U$), introduce a new vertex $x'$ such that $a$’s endpoint in $\bar{U}$ is $x$ and $b$’s endpoint is $x'$. The edges in $E$ are identified with the corresponding edge in $E'$. A configuration $K_E$ is a forest with no degree-2 vertices whose leaf edges are a subset of $E'$ (and also, $E$). See Figure 3.22. Cutting the connectivity of $E$ in $\bar{U}$ implies that if $e \in K_E \cap E$ then $e$ is a leaf edge of $K_E$. We denote the set of all configurations on edge set $E$ by $K_E$.

We now define consistent for 2-EC as needed for filling $\text{Tab}$. We say a configuration $K_A$ is consistent with a set of mutually compatible configurations $\{K_{A_1}, K_{A_2}, \ldots\}$ if $K_A$ is isomorphic to the graph $\tilde{EC}(\cup_i K_{A_i})$.

Let $U$ be a set of vertices and let $M$ be a subgraph. Let $M'$ be the subgraph of $M$ induced by the vertex set $U \cup V(\Gamma(U))$. Let $M_U$ be the graph obtained by cutting the connectivity between edges of $M' \cap \Gamma(U)$ in $\bar{U}$. We say that $M$ meets a configuration $K_{\Gamma(U)}$ if $\tilde{EC}(M_U) = K_{\Gamma(U)}$.

![Figure 3.22: A configuration (dotted forest) for a set of edges $E'$ (b) derived from the cut edges $E$ where the vertices $U$ are in the shaded region (a). (c) A subgraph (bold) that meets the configuration.](image)

Solving the base case

Suppose $v$ is a leaf of the recursion tree. We show how to fill $\text{Tab}_v$. Recall that we have doubled the edges in our graph.
If \( v \) does not correspond to a brick (i.e. \( f(v) = v \)), the problem is trivial. Each configuration \( K \) is a star: \( v \) is the center vertex, and the edges of \( \Gamma(\{v\}) \) are the edges of \( K \). Since the weight of any subgraph that is induced by \( \{v\} \) is zero, the value of \( \text{TAB}_v[K] \) is zero for every \( K \).

Suppose then that \( v \) corresponds to a brick \( B \) (i.e. \( f(v) = B \)). Let \( K \) be a configuration corresponding to the edges of the cut \( \Gamma(V(W(v))) \) in \( \mathcal{B}^+(P) \). This is the set of portal edges corresponding to brick \( B \). Let \( H_K \) be the minimum-weight subgraph of \( B \cup \Gamma(V(W(v))) \) that meets the configuration \( K \). Since we need only maintain connectivity between vertices on the boundary of \( B \) (i.e. between portal edges), we know by Lemma 3.3.16 that \( H_K \) is the union of trees. And so we need only consider configurations that correspond to boundary connectivity. For all other configurations, we can assign \( \infty \) to the table entry. Let \( a \) and \( b \) be edges of \( \Gamma(V(W(v))) \) and let \( a' \) and \( b' \) be their duplicates. Note that \( a \) and \( a' \) are parallel and share a common endpoint \( x_a \) in \( B \). If all four edges are mutually adjacent in \( K \) (i.e. they share a common endpoint), then \( c_{H_K}(x_a, x_b) = 2 \). If one of \( a \) and \( a' \) appear in the same component of \( K \) with one of \( b \) or \( b' \), then \( c_{H_K}(x_a, x_b) = 1 \). Otherwise \( c_{H_K}(x_a, x_b) = 0 \).

By Lemma 3.4.1 there are at most \( \theta \) portal edges corresponding to any brick. Including duplicate edges, there are at most \( 2\theta \) leaf edges in \( K \). From \( K \), we can compute \( c_{H_K} \). Using the algorithm \textsc{DecomposeConnectivity}, we can find the sub-partition \( S \) of the vertices of \( \partial B \) that correspond to the leaves of the trees of \( H_K \) (Lemma 3.3.16). We can compute this sub-partition in in \( O(\theta^2) \) time (Lemma 3.3.13). For each set in \( S \) we compute the weight of the Steiner tree connecting those vertices in \( O(\theta^3 n) \) (Theorem 3.1.1). The value assigned to \( \text{TAB}_x[K] \) is the sum of the weights of these trees. We do not include the weights of the portal edges (which have zero weight). Since there are at most \( 2\theta \) sets in \( S \). It takes \( O(\theta^3 n_B) = O(\theta^4 n_B) \) time to compute the length of all the trees corresponding to configuration \( K \) where \( n_B \) is the number of vertices in brick \( B \). The total running time to compute \( \text{TAB}_x[K] \) is therefore \( O(\theta^4 n_B) \).

**Running time**

Recall that a configuration \( K_E \) is a forest with no degree-2 vertices whose leaf edges are a subset of edge-set \( E \). Here we bound number of such configurations, \( |K_E| \). A forest with no degree-2 vertices and at most \( |E| \) leaves has at most \( 2|E| \) vertices. Cayley’s formula states that the number of trees with \( n \) vertices is \( n^{n-2} \). It follows that the number of forests with \( n \) vertices is \( (n + 1)^{n-2} \). The number of forests with at most \( n \) vertices is at most \( n(n+1)^{n-2} \). The set of trees can be computed in \( O(1) \) amortized time per tree [82].

For the base case, \( |E| \) is the number of portal edges corresponding to a brick, which is at most \( 2\theta \), including duplication. Populating \( \text{TAB}_v \) for \( v \) corresponding to brick \( B \) takes time \( O(\theta(\theta+1)^{\theta-1}n_B) \) which is bounded by \( O(2^{\text{poly}}(1/\epsilon) n_B) \) since \( \theta \) depends polynomially on \( 1/\epsilon \). Populating \( \text{TAB}_v \) for \( v \) not corresponding to a brick takes time \( O(1) \). Since each vertex appears in at most twice in the set of bricks, the time to solve all the base cases is \( O(2^{\text{poly}}(1/\epsilon) n) \).

For \( v \) not a leaf of the recursion tree, the number of edges in \( \Gamma(V(W(v))) \) in \( \mathcal{B}^+(P) \) is at most \( 4\theta\eta + 2 \), including duplicates (Lemma 3.7.2). It follows that the corresponding number of
configurations is \(O(2^{\text{poly}(1/\epsilon)})\) since \(\theta\) and \(\eta\) each depend polynomially on \(1/\epsilon\). There are \(O(n)\) vertices of the recursion tree and so the time required for the dynamic program, not including the base cases is \(O(2^{\text{poly}(1/\epsilon)n})\).

The total running time of the dynamic program is \(O(2^{\text{poly}(1/\epsilon)n})\) where \(n\) is the number of vertices in the parcel.

### 3.7.5 Correctness

**Theorem 3.7.3.** The dynamic program finds the optimal solution.

**Proof.** Refer to Table 3.8.

The connecting property guarantees that the final solution is feasible (satisfying the connectivity requirements). The definitions of compatible and consistent guarantee the inductive hypothesis.

We show that Step 5 computes the length of a minimum-weight subgraph \(H_{u_0}\) induced by the vertices of \(W(u_0)\) that meets the configuration \(K_0\). We have shown that this is true for the leaves of the recursion tree. Since \(K\) is the configuration corresponding to the cut \(\Gamma(\{u_0\})\), \(K\) is a star. Therefore \(w(K)\) is the weight of the edges of \(\Gamma(\{u_0\})\): \(K\) is both the configuration and a minimum-weight subgraph that meets that configuration. Further, \(w(K \cap (\cup_{i=1}^s K_i))\) is the weight of the edges of \(K\) that are in \(K_i\) (for \(i = 1, \ldots, s\)). \(w(\cap_{i=1}^s K_i)\) is equal to the weight of the edges common to \(K_1\) and \(K_2\) if \(s = 2\) and zero otherwise. By the inductive hypothesis the weight computed is that of a \(H_{u_0}\): the subgraph induced by the vertices in \(W(u_0)\) of a minimum-weight graph that meets this configuration.

Consider the entries of \(\text{TAB}_u\) where \(u\) is the root of the recursion tree. Since \(\Gamma(W(u))\) is empty, there is only one configuration corresponding to this subproblem: the trivial configuration. 

By Lemma 3.4.3, combining parcel solutions forms a valid solution in our input graph for terminal set \(Q\). We need to compare the length of the output to the length of an optimal solution. Let \(\hat{Q}\) denote the set of new terminals.

Let \(H\) be the optimal solution in \(\mathcal{B}^+(MG)\) spanning the original terminals. By the spannable property of the new terminals, for each parcel \(P\), there is a (possibly empty) solution \(H_P\) in \(\mathcal{B}^+(P)\) consisting of edges of \(H \cup \partial H\) (where \(\partial H\) is the set of boundary edges of all parcels) for the new and original terminals in \(P\). We have:

\[
\text{OPT}(\mathcal{B}^+(P), Q \cup \hat{Q}) \leq \ell(H_P) = \ell(H_P \setminus \partial H) + \ell(H_P \cap \partial H).
\]

Every edge of \(H\) not in \(\partial H\) appears in \(H_P\) for exactly one parcel \(P\), and so \(\sum_{P \in \mathcal{N}} \ell(H_P \setminus \partial H) \leq \ell(H) = \text{OPT}(\mathcal{B}^+(MG), Q)\). Every edge of \(\partial H\) appears in two parcels, and so \(\sum_{P \in \mathcal{N}} \ell(H_P \cap \partial H) \leq 2 \cdot \ell(\partial H)\). Thus the length of the output is

\[
\ell(\partial H) + \sum_{P \in \mathcal{N}} \text{OPT}(\mathcal{B}^+(P), Q \cup \hat{Q}) \leq \text{OPT}(\mathcal{B}^+(MG), Q) + 3\ell(\partial H).
\]

Combining the parcel-boundary length property (Section 3.6.1), the definition of \(\eta\) (Equation (3.7)), and the mortar-graph length property (Section 3.2), we obtain \(\ell(\partial H) \leq \frac{1}{2} \epsilon \text{OPT}(G, Q)\).
Finally, by Theorem 3.4.4, the length of the output is at most \((1 + c\epsilon) \text{OPT}(G, Q)\). This shows that we have a PTAS for the Steiner-tree and 2-EC problems that, combining the running times of the dynamic program and brick decomposition (and all steps in between) is \(O(2^{\text{poly}(1/\epsilon)n} + n \log n)\), giving:

**Theorem 3.7.4.** There is an approximation scheme for solving the Steiner-tree and 2-EC problems (allowing duplication of edges) in planar graphs. The running time is \(O(2^{\text{poly}(1/\epsilon)n} + n \log n)\).

### 3.8 An exact algorithm for the boundary 2-EC problem

We give an algorithm (Table 3.9) for the following problem: given a weighted, planar graph \(G\) and a set of requirements \(r\) that are only non-zero for vertices of \(\partial G\), find a minimum-weight multi-subgraph \(H\) of \(G\) that satisfies the requirements (i.e. there are \(\min\{r(x), r(y)\}\) edge-disjoint \(x\)-to-\(y\) paths in \(H\)). This will prove Theorem 3.1.2 that was stated in Section 3.1.2.

**Table 3.9:** An algorithm for computing the minimum-weight multi-subgraph of \(G\) that satisfies requirements on the boundary vertices.

<table>
<thead>
<tr>
<th>Boundary2EC(G, Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Let (q_1, q_2, \ldots) be the cyclic ordering of vertices ({v \in V(\partial G) : r(v) = 2}).</td>
</tr>
<tr>
<td>2. For (i = 1, \ldots), let (X_i = {v \in V(\partial G[q_i, q_{i+1}])}).</td>
</tr>
<tr>
<td>3. For (i = 1, \ldots), let (T_i) be the minimum Steiner tree spanning (X_i).</td>
</tr>
<tr>
<td>4. Return the disjoint union (\cup_i T_i).</td>
</tr>
</tbody>
</table>

We will use the following lemma to give an efficient implementation of **Boundary2EC**. The idea is similar to one used in [75].

**Lemma 3.8.1.** Let \(a, b\) and \(c\) be vertices ordered along the clockwise boundary \(\partial G\) of a planar graph \(G\). Let \(T_a\) be the shortest-path tree rooted at \(a\). Then for any set of terminals \(Q\) in \(\partial G[b, c]\), there is a minimum Steiner tree connecting them that enclosed by the cycle \(\partial G[b, c] \circ T_a[c, b]\).

**Proof.** Let \(C = \partial G[b, c] \circ T_a[c, b]\). Let \(T\) be a minimum Steiner tree in \(G\) connecting \(Q\). Suppose some part of \(T\) is not enclosed by \(C\). Let \(T'\) be a maximal subtree of \(T\) not enclosed by \(C\). The leaves of \(T'\) are on \(T_a[c, b]\). Let \(P\) be the minimum subpath of \(T_a[b, c]\) that spans these leaves. Let \(P'\) be the start\((P)\)-to-end\((P)\) path in \(T'\). See Figure 3.23.

We consider the case when start\((P')\) is a vertex of \(T_a[a, b]\) and end\((P')\) is a vertex of \(T_a[a, c]\) (the other cases are similar). Then \(P'\) must cross \(T_a[a, x]\) where \(x\) is the last vertex common to \(T_a[a, b]\) and \(T_a[a, c]\). Let \(y\) be a vertex of \(P' \cap T_a[a, x]\). Since \(T_a\) is a shortest-path tree in an undirected
path, every subpath of $T_a[a, z]$ and $T_a[z, a]$, for any vertex $z$, is a shortest path. We have that:

\[
\begin{align*}
\text{w}(P') &= \text{w}(P'[\text{start}(P'), y]) + \text{w}(P'[y, \text{end}(P')]) \\
&\geq \text{w}(T_a[\text{start}(P'), y]) + \text{w}(T_a[y, \text{end}(P')]) \\
&\geq \text{w}(T_a[\text{start}(P'), \text{pend}(P')]) \\
&\geq \text{w}(P)
\end{align*}
\]

Let $\tilde{T} = T \setminus T' \cup P$. By construction, $\tilde{T}$ spans $Q$. Using that $\text{w}(P') \geq \text{w}(P)$, we have that $w(\tilde{T}) = w(T) - w(T') + w(P) \leq w(T) - w(T') + w(P') \leq w(T)$ since $P'$ is a subpath of $T'$.

Repeating this process for every subtree of $T$ not enclosed by $C$ results in a tree enclosed by $C$ spanning $Q$ that is no longer than $T$.

---

**Figure 3.23**: There is a tree $\tilde{T}$ that is just as short as $T$ (dotted) and spans the terminals between $b$ and $c$ but is enclosed by $C$ (whose interior is shaded). $\tilde{T}$ is composed of the portion of $T$ enclosed by $C$ plus $P$, the thick grey path.

We describe an $O(k^3 n)$-time implementation of **Boundary2EC** (where $k$ is the number of terminals). Compute a shortest-path tree $T$ rooted at terminal $q_1$ in linear time. For each $i$, consider the graph $G_i$ enclosed by $C_i = \partial G[q_i, q_{i+1}] \cap T[q_{i+1}, q_i]$. Compute the minimum Steiner tree spanning $X_i$ in $G_i$. By Lemma 3.8.1, $T_i$ has the same length as the minimum spanning tree spanning $X_i$ in $G$. Since each edge of $G$ appears in at most two subgraphs $G_i$ and $G_j$, the trees $T_i$ can be computed in $O(k^3 n)$ time (Theorem 3.1.1). This is a correct implementation of **Boundary2EC**.

We now argue that **Boundary2EC** finds the minimum-weight multi-subgraph of $G$ satisfying the requirements. The vertices selected in Step 1 form the only 2-requirement clique among the terminals. **DECOMPOSECONNECTIVITY**(V($\partial G$), $r$) would find the sets $X_i$ computed in Step 2. By Lemma 3.3.16, the optimal solution $H$ is the disjoint union of trees $\cup T_i'$ such that $T_i'$ spans $X_i$. Since $w(T_i) \leq w(T_i')$, the output of **Boundary2EC** has weight at most $w(H)$.

Note: if the requirements are such that $r(v) \in \{0, 2\}$ for every vertex $v$ on the boundary of $G$, then the sets $X_i$ have cardinality 2. Instead of computing Steiner trees in Step 3, we need only compute shortest paths. The running time for this special case is therefore linear.

This completes the proof of Theorem 3.1.2.
3.9 Open Problems

We have given polynomial-time approximation schemes for two subset connectivity problems: the Steiner-tree problem and the 2-edge connected problem. The schemes (using either the framework of Klein [67] or the more direct method given in Section 3.6) rely on the brick decomposition. Recall that the brick decomposition is based heavily on a construction used to give a PTAS for the subset tour problem in planar graphs [69]. We originally gave the Structure Theorem for the Steiner-tree problem [15] only recently generalizing this to 2-edge connectivity. The development of these three results certainly point to the generality of the brick decomposition. We expect that the brick decomposition can be used to give approximation schemes for other connectivity problems.

For example, it may be possible to extend the framework to get a PTAS for higher–connectivity problems. Lemma 3.3.16 showed that a graph can be partitioned into trees while maintaining 2-edge-connectivity between vertices on the boundary of the graph. If this can be generalized to maintain k-edge connectivity, then a PTAS for k-edge connectivity follows for fixed k.

In this thesis, we have allowed duplication of edges to obtain the required connectivity. Using the techniques of this thesis alone, it does not seem possible to overcome this relaxation. While the partitioning of a solution within a brick into trees (Lemma 3.3.16), and grouping these trees into forests does not cause any obstacle, the use of the Theorem 3.3.3 does. We use Theorem 3.3.3 to replace each forest embedded in a brick of the aforementioned partition with a simpler forest. However, parts of each forest is redirected along the boundary of the brick, introducing duplicate edges (Figure 3.24).

![Figure 3.24: A simple example illustrating the introduction of duplicate edges as a result of the Structure Theorem. The intersection of a 2-EC graph (solid) with a brick (dashed) in (a) is decomposed into 2 forests, grey and black (b), each of which is replaced by spanning subpaths of the southern boundary of the brick (c) due to Lemma 3.3.4 thereby introducing duplicate edges on the southern boundary of the brick.](image)

In [7], Berger and Grigni gave a PTAS for the problem of finding a 2-edge-connected subgraph that spans all the vertices of the input graph but not allowing duplicate edges. The algorithm builds
a spanner to be used in the framework (Table 3.4), just as we do for the subset case. However, in order to prevent the duplication of edges, in each step of the dynamic program a constant number of edges that do not appear in the spanner are considered. It may be possible to combine this technique with ours.

The brick decomposition and Klein’s PTAS framework are amenable to problems that have full connectivity constraints. That is, the solution spans the set of input terminals. On the other extreme, Baker outlined a method for obtaining approximation schemes in planar graphs for problems with no connectivity constraints (for example, maximum independent set and minimum vertex cover) [5]. Baker’s scheme involves contracting every \( k^{th} \) breadth-first search layer in the input graph (rather than a spanner subgraph) to find a graph of treewidth \( O(k) \). The problem is solved for each possible residual of breadth-first search layers. For some residual, the weight of the edges that are contracted is bounded by \( \text{OPT}/k \).

There are host of problems between these two extremes. Most notably is the Steiner forest problem. The input to the Steiner forest problem is a set of terminal pairs \( s_i, t_i \). The output is a minimum-weight forest that connects \( s_i \) to \( t_i \) for every \( i \). While \( s_i \) need not be connected to \( t_j \) (for \( i \neq j \)), the minimum-weight solution may happen to connect \( s_i \) to \( t_j \) (see Figure 3.25. The connected components of the optimal solution induce a partition of the terminal set. If one could find this partition, the problem would reduce to a series of Steiner tree problems and the PTAS given in this thesis would imply a PTAS for the Steiner forest problem.

![Figure 3.25](image)

Figure 3.25: Given a set of terminal pairs the optimal solution might not connect different pairs (a), or connect pairs together (b) and (c).

Of course, finding this partition is non-trivial. It is not necessary to find the partition corresponding to an optimal solution, but only a partition corresponding to a nearly optimal solution. Again, if such a partition could be found the Steiner-tree PTAS can be used to give a PTAS for finding the Steiner forest. One could start with a 2-optimal solution [2], thereby satisfying the connectivity requirements, and add edges to achieve connectivity sufficient for a \( (1+\epsilon) \)-optimal solution. We have recently given a PTAS for the Steiner forest problem in the Euclidean plane using this tactic [18].

If one is able to give a PTAS for the Steiner forest problem, then it is natural to consider the most general of connectivity of requirements: for every pair \( x, y \) of terminals require that there are \( r(x, y) \in \{0, \ldots, k\} \) edge-disjoint \( x \)-to-\( y \) paths. Here, the final solution is not required to span the
terminals.

There is a lifetime’s worth of connectivity (and disconnectivity) problems to consider when you add vertex weights, vertex-disjointness and multi-terminal cut problems to the mix.
Appendix A

Notation

\(x\) a vector in \(R^A\), \(R^E\) or \(R^D\)
\(\delta(a)\) a vector satisfying \(\delta(x) = 0\) for \(x \neq a\) and \(\delta(a) = 1\)
\(\delta(v)\) for \(v\) a vertex \(\sum_{\text{tail}(d) = v} \delta(d)\)
\(\Gamma(S)\) the set of edges with one endpoint in \(S\) and the other in \(\overline{S}\)
\(\Gamma^+(S)\) the set of darts whose tails are in \(S\) and heads are in \(\overline{S}\)
\(\partial G\) the boundary of the graph \(G\) as a clockwise cycle
\(P[x, y]\) the \(x\)-to-\(y\) subpath of \(P\)
\(P(x, y)\) the \(x\)-to-\(y\) subpath of \(P\) with \(x\) deleted
\(P[\cdot, y]\) the prefix of path \(P\) ending in vertex \(y\)
\(P \circ Q\) the concatenation of paths \(P\) and \(Q\)
\(\text{head}(d)\) the vertex that dart \(d\) points to
\(\text{tail}(d)\) the vertex that dart \(d\) points out of
\(\text{start}(P)\) the first vertex of path \(P\)
\(\text{end}(P)\) the last vertex of path \(P\)
\(T[x, y]\) the minimal path in tree \(T\) from vertex \(x\) to vertex \(y\)
\(T[x]\) the minimal path in rooted tree \(T\) from vertex \(x\) to the root of \(T\)
\(T(v)\) the subtree of rooted-tree \(T\) rooted at \(v\) containing all descendents of \(v\)
Bibliography


