# ALGORITHMS FOR FLOWS AND DISJOINT PATHS IN PLANAR GRAPHS

BY

YIPU WANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Doctoral Committee:

Professor Jeff Erickson, Chair
Professor Chandra Chekuri
Assistant Professor Karthekeyan Chandrasekaran
Professor Philip N. Klein, Brown University

**ABSTRACT**

In this dissertation we describe several algorithms for computing flows, connectivity, and disjoint paths in planar graphs. In all cases, the algorithms are either the first polynomial-time algorithms or are faster than all previously-known algorithms.

First, we describe algorithms for the maximum flow problem in directed planar graphs with integer capacities on both vertices and arcs and with multiple sources and sinks. The algorithms are the first to solve the problem in near-linear time when the number of terminals is fixed and the capacities are polynomially bounded. As a byproduct, we get the first algorithm to solve the vertex-disjoint $S-T$ paths problem in near-linear time when the number of terminals is fixed but greater than 2. We also modify our algorithms to handle real capacities in near-linear time when they are three terminals.

Second, we describe algorithms to compute element-connectivity and a related structure called the reduced graph. We show that global element-connectivity in planar graphs can be found in linear time if the terminals can be covered by $O(1)$ faces. We also show that the reduced graph can be computed in subquadratic time in planar graphs if the number of terminals is fixed.

Third, we describe algorithms for solving or approximately solving the vertex-disjoint paths problem when we want to minimize the total length of the paths. For planar graphs, we describe: (1) an exact algorithm for the case of four pairs of terminals on a single face; and (2) a $k$-approximation algorithm for the case of $k$ pairs of terminals on a single face.

Fourth, we describe algorithms and a hardness result for the ideal orientation problem. We show that the problem is NP-hard in planar graphs. On the other hand, we show that the problem is polynomial-time solvable in planar graphs when the number of terminals is fixed, the terminals are all on the same face, and no two of the terminal pairs cross. We also describe an algorithm for serial instances of a generalization of the ideal orientation problem called the $k$-min-sum orientation problem.

**ACKNOWLEDGMENTS**

Writing this dissertation would not have been possible without the support of many people. First, I would like to thank my advisor, Professor Jeff Erickson, for introducing me to many of the problems and techniques that are considered in this dissertation, and for having many fruitful discussions with me in his office. I would also like to thank the other members of the doctoral committee for reading an earlier draft of this dissertation and providing many helpful suggestions to improve the writing.

I would like to thank the National Science Foundation (NSF) for supporting part of the research for this dissertation through NSF grant CCF-1408763.

I would like to thank everyone who was part of either the University of Illinois at Urbana-Champaign (UIUC) theory group from 2013 to 2020 or the Oregon State University theory group when I visited in 2017 for fruitful discussions. Conversations with Professor Chandra Chekuri, Hsien-Chih Chang, Assistant Professor Amir Nayyeri, and Hung Le were particularly helpful.

I would like to thank the reader for taking the time to read this. I hope that you may find something useful in this dissertation, and I apologize for any errors that remain in it.

I would like to thank my parents for being supportive of my personal and career endeavors. Finally, I would like to thank my wife for many things, none of which are the reader's business.

## TABLE OF CONTENTS

# CHAPTER 1: INTRODUCTION

Computer scientists have been trying to develop algorithms for planar and near-planar graphs since the 1950s for two reasons. First, such graphs have substantial structure, and as a result often admit algorithms that are faster and simpler than algorithms for general graphs. Second, such graphs arise often in applications, especially in transportation, geographical routing, computer vision, and VLSI design. Specific problems in planar graphs that have applications include the maximum flow problem, the minimum cut problem, the shortest path problem, and the disjoint paths problem. For example, road networks can be modeled as planar graphs if we ignore bridges and tunnels, and so it is useful to find shortest paths in planar graphs. In computer vision, pixels in images are often arranged in a two-dimensional grid, and they need to be partitioned into clusters using minimum-cut algorithms. In VLSI design, we need to pack components onto a chip so that the wires connecting the components do not cross each other, so it is useful to find disjoint paths.

This thesis is concerned with two classes of problems: *connectivity problems* and *disjoint path problems*. In connectivity problems, we are given, say, two vertices called *terminals* in a graph, and we want to find the minimum number of graph components that we need to remove in order to disconnect the two terminals. The set of removed components is called a *cut*. Here a "component" can mean a vertex, or an edge, or something called an *element* [1, 2, 3]; depending on what we consider to be a component, we get different variations of the problem. We also get different variations depending on whether the input graph is directed or undirected, and whether or not the components of the graph are weighted. (If the components are weighted, then we want to minimize the total weight of the removed components instead of the number of removed components.) In another variation, there may be more than two terminals. In this case, we may be interested in disconnecting any pair of terminals, or we may designate some terminals to be sources and some to be sinks, in which case we are only interested in disconnecting the sources from the sinks.

Theorems by Karl Menger [4] and by Lester R. Ford Jr. and Delbert R. Fulkerson [5] show that there is an intimate connection between connectivity and *maximum flows* in graphs, which are ways of routing commodities through graphs. In many cases, our best algorithms for computing connectivity actually compute flows and then extract connectivity values from those flows. Network flow problems are themselves an important class of problems studied in operations research and computer science. They were first formulated in the 1950s by Theodore E. Harris and Frank S. Ross, who were studying the Soviet rail network in Eastern Europe [6]. Specifically, Harris and Ross wanted to compute how much of certain commodities the network could transport from certain cities to other cities. Soon afterwards, Ford and Fulkerson described an algorithm for this problem (and thus for network flow problems in general) [5]. Since then, network flow has found further applications in transportation, logistics, telecommunications, and scheduling [7].

The second class of problems that this thesis is concerned with are disjoint-paths problems. In disjoint-paths problems, we are given a set of $k$ pairs of vertices called *terminals*, and we want to find $k$ pairwise-disjoint paths such that the $i$-th path connects the $i$-th pair of terminals. Here "pairwise-disjoint" can mean either edge-disjoint [8], vertex-disjoint [9], non-crossing [10], or something a bit less standard called *nonconflicting* [11]. *Shortest* disjoint paths problems are defined similarly, but we also want the $k$ paths to be "short" in some sense; for example, we may require that each path be a shortest path connecting its endpoints [12], or that the sum of the lengths of the $k$ paths be minimized [9, 13], or that the length of the longest of the $k$ paths be minimized [9]. As in the case of connectivity problems, we also get different variations depending on whether the input graph is directed or undirected, and whether the graph is weighted or unweighted. In still another variation, we want to connect as many terminal pairs as possible via disjoint paths [14].

Disjoint paths problems are an important class of problems in graph theory, with applications in VLSI design [15, 16] and network routing [17, 18]. Both maximum-flow problems and the disjoint paths problem are special cases of multicommodity flow problems. Specifically, the maximum flow problem is a fractional multicommodity flow problem with only a single commodity, while the edge-disjoint paths problem is an integral multicommodity flow problem where each edge has unit capacity and each terminal pair has unit demand. In addition, the version of the shortest edge-disjoint paths problem where we wish to minimize the sum of the lengths of the paths is an integral *minimum-cost* multicommodity flow problem where each edge has unit capacity, each terminal pair has unit demand, and each edge has cost equal to its length.

In this thesis we study two connectivity problems and two shortest disjoint paths problems in planar graphs: the maximum flow problem with vertex capacities; the element connectivity problem; the minimum-sum shortest disjoint paths problem; and the *ideal orientation problem* [19], in which we are asked to find nonconflicting paths that are also shortest paths. These problems will be defined more precisely in section 2. All of them involve finding paths, flows, or cuts that are optimal in some sense. For the two connectivity problems, polynomial-time algorithms are already known [3, 20], and the primary open question is whether or not the algorithms can be sped up in planar graphs, ideally to near-linear time. The two shortest-disjoint-path problems are solvable in very special cases [9, 12] and are NP-hard in general [19, 21], but many cases in between still have unknown complexity, even in planar graphs.

This thesis is organized as follows. In Chapter 2 we define the key concepts that will be in used in the rest of the thesis. In Chapter 3 we investigate flow in directed planar graphs with vertex capacities and multiple sources and sinks. First, we show that for unit capacities (equivalently, bounded integer capacities), maximum flows can be found in $O(\min\{k^2 n, n \log^3 n + kn\})$ time, where $k$ is the number of terminals. Second, we show that for integer capacities, maximum flows can be found in $O(k^5 n \text{ polylog}(nU))$ time, where $U$ is the maximum capacity of a single vertex or arc. Third, we show that for three terminals, we can find maximum flows in $O(n \log n)$ time, even when the capacities are non-negative reals. All three results are obtained by extending Kaplan and Nussbaum's algorithm for finding maximum flows in directed planar graphs with vertex capacities and a single source and sink [22]; the second result also uses an algorithm of Borradaile et al. for finding maximum flows in $k$-apex graphs [23].

In Chapter 4, we investigate element connectivity in planar graphs. We show that the global element connectivity can be found in $O(bn)$ time if the terminals can be covered by $b$ faces. In addition, we show that the reduced graph of a planar graph can be found in $O(kn^{5/3} \log^{4/3} n)$ time, where $k$ is the number of terminals.

In Chapter 5, we describe our results for the minimum-sum vertex-disjoint paths problem. We show that for four terminal pairs on the same face of a planar graphs, the problem can be solved in $O(kn^6)$ time. In addition, we describe a $k$-approximation algorithm for the case where there are $k$ terminal pairs on the same face of a planar graph.

In Chapter 6, we describe our results for the orientation problem in planar graphs. We describe four results: (1) an $O(n \log n)$-time algorithm for the ideal orientation problem when all terminals are on the outer face in a certain order we call *serial order*; (2) a polynomial-time algorithm for the ideal orientation problem when all terminals are on the outer face and *non-crossing*, and the number of terminals is fixed; (3) a proof that the ideal orientation problem is NP-hard in planar graphs if the terminals are allowed to be on any face and the number of terminals is part of the input; and (4) an $O(kn^5)$-time algorithm for the minimum-sum orientation problem (in which the nonconflicting paths need not be shortest paths but must have minimum total length) when all the terminals are on the outer face in serial order.

# CHAPTER 2: PRELIMINARIES

## 2.1 GRAPHS

In this thesis, $G$ is a simple plane graph with vertex set $V(G)$, edge or arc set $E(G)$, and face set $F(G)$. When there is no risk of confusion, we will write $V$ for $V(G)$, $E$ for $E(G)$, or $F$ for $F(G)$. The graph $G$ can be either directed or undirected; we will specify when we need $G$ to be one or the other. The graph $G$ is also weighted; depending on the context, the weight function is either a capacity function $c$ or a length function $\ell$. Let $n$ be the number of vertices in $G$; it is well known that Euler's formula implies $|E(G)| = O(n)$. For any vertex $v \in V(G)$, let $\deg_G(v)$ denote the degree of $v$ in $G$. If $G$ is a graph and $W \subseteq V(G)$, then $G \setminus W$ is the induced subgraph of $G$ with vertex set $V(G) \setminus W$. For any integer $N$, let $[N] = \{1, \ldots, N\}$.

**Walks, paths, incoming arcs, outgoing arcs.** We use $uv$ or $(u, v)$ to denote an arc or directed edge that is directed from $u$ to $v$, and $\{u, v\}$ to denote an undirected edge connecting $u$ and $v$. A nontrivial *walk* is a sequence of arcs $((u_1, v_1), \ldots, (u_p, v_p))$ such that $v_i = u_{i+1}$ for all $i \in [1, p-1]$. Such a walk *starts* at $u_1$ and *ends* at $v_p$. If in addition $v_p = u_1$ then $P$ is a *cycle*. We can also have *trivial walks* that consist of no arcs; such walks start and end at the same vertex. In a slight abuse of terminology, we will say that the walk $W$ *uses* the (undirected) edges $\{u_0, v_0\}, \ldots, \{u_p, v_p\}$. The directed walk $W$ is in an undirected graph $G$ if $\{u_i, v_i\}$ is an edge in $G$ for all $i \in \{0, \ldots p-1\}$. A walk $W$ contains a vertex $v$ if one of the arcs of $W$ has $v$ as an endpoint. Thus we will sometimes view paths and cycles as sets of vertices or as sets of arcs instead of as sequences of arcs. The walk $((u_1, v_1), \ldots, (u_p, v_p))$ is a *path* if $u_1, \ldots, u_p, v_p$ are all distinct. For any $v \in V$, let $\text{in}(v) = \{(u, v) \mid (u, v) \in E(G)\}$ be the set of *incoming arcs* of $v$, and let $\text{out}(v) = \{(v, u) \mid (v, u) \in E(G)\}$ be the set of *outgoing arcs* of $v$. Similarly, if $W$ is a set of vertices, then $\text{in}(W) = \{(u, v) \in E(G) \mid u \notin W, v \in W\}$ and $\text{out}(W) = \{(u, v) \in E(G) \mid u \in W, v \notin W\}$.

**Touching, subpaths, concatenation.** Two walks *meet* or *touch* if they have at least one vertex in common. Two regions *touch* or *meet* if their closures have non-empty intersection. For any path $P$ and any vertices $u$ and $v$ on that path, we write $P[u, v]$ to denote the subpath of $P$ from $u$ to $v$. Similarly, let $P[u, v)$ denote the subpath of $P$ from $u$ to the predecessor of $v$, let $P(u, v]$ denote the subpath of $P$ from the successor of $u$ to $v$, and let $P(u, v)$ denote the subpath of $P$ from the successor of $u$ to the predecessor of $v$; these subpaths could be empty. The concatenation of two paths $P$ and $P'$ is denoted $P \circ P'$.

**Predecessors, reversals, $k$-apex graphs.** If $u$ appears before vertex $v$ on the walk $P$, then we write $u \prec_P v$; we will only use this notation when there is no risk of ambiguity. The *reversal* of any arc $(u, v)$, denoted $\mathrm{rev}((u, v))$, is $(v, u)$. We may assume without loss of generality that if $e \in E(G)$, then $\mathrm{rev}(e) \in E(G)$, and both $e$ and $\mathrm{rev}(e)$ are embedded together. If $P$ is a path $(e_1, \ldots, e_p)$, then the *reversal* of $P$, denoted $\mathrm{rev}(P)$, is $(\mathrm{rev}(e_p), \ldots, \mathrm{rev}(e_1))$. A graph $G_a$ is a *$k$-apex graph* if there are $k$ vertices whose removal from the graph would make $G_a$ planar. These $k$ vertices are called *apices*.

**Boundary and degree.** For any plane graph $G$, we write $\partial G$ to denote the boundary of the outer face of $G$; we also informally call $\partial G$ the boundary of $G$. In Chapter 5, we will assume without loss of generality that $\partial G$ is a simple cycle. We write $\deg(v)$ to denote the degree of a vertex $v$.

**Orientations and crossing.** An *orientation* of an undirected graph $G$ is a directed graph $H$ that is formed by replacing each edge $\{u, v\} \in E(G)$ with exactly one of the arcs $uv$ or $vu$. If four terminals $s_i, t_i, s_j, t_j$ are on a common face, then we say that the terminal pairs *cross* if their cyclic order (either clockwise or counterclockwise) on the face is $s_i, s_j, t_i, t_j$; otherwise, the terminals are *noncrossing*. Similarly, two embedded paths $P_1$ and $P_2$ *cross* at a vertex $v$ if there are four arcs or edges of $P_1 \cup P_2$ incident to $v$ and these edges or arcs alternate between $P_1$ and $P_2$ in cyclic order around $v$.

**Duality.** If $G$ is a plane graph, the *dual graph* $G^*$ of $G$ has a vertex $h^*$ for every face $h$ of $G$, and an arc $e^*$ for every arc $e$ of $G$. The arc $e^*$ is directed from the vertex of $G^*$ corresponding to the face in $G$ on the left side of $e$, to the vertex of $G^*$ corresponding to the face in $G$ on the right side of $e$. If $e$ is undirected, then so is $e^*$. Any undirected edge $\{u, v\}$ can be represented by two directed arcs $(u, v)$ and $(v, u)$, each with the same weight as $\{u, v\}$. We put lengths $\ell(e^*)$ on the arcs $e^*$ of $G^*$ as follows: $\ell(e^*) = c(e)$ for every $e \in E(G)$.

## 2.2 FLOWS AND CONNECTIVITY

### 2.2.1 Maximum flows

Suppose that $G$ is a directed planar graph such that two disjoint subsets of $V(G)$ are special: $S$ is a set of *sources* and $T$ is a set of *sinks*. In the context of maximum flow, vertices that are in either $S$ or $T$ are called *terminals*, and $k$ is the number of terminals. Suppose further that each arc $e$ has a non-negative capacity $c(e)$ and each non-terminal vertex $v$ has a positive capacity $c(v)$. In this case we say that $G$ is a *flow network*.

**$f^{in}$, $f^{out}$, flows, feasibility.** Let $f : E(G) \to [0, \infty]$. To lighten notation, in this thesis we will write $f(u, v)$ instead of $f((u, v))$ for any arc $(u, v)$. For each vertex $v$, let

$$f^{in}(v) = \sum_{e \in \text{in}(v)} f(e) \quad \text{and} \quad f^{out}(v) = \sum_{e \in \text{out}(v)} f(e). \tag{2.1}$$

Similarly, if $W$ is a set of vertices, then let

$$f^{in}(W) = \sum_{e \in \text{in}(W)} f(e) \quad \text{and} \quad f^{out}(W) = \sum_{e \in \text{out}(W)} f(e). \tag{2.2}$$

The function $f$ is a *flow in $G$* if it satisfies the following *flow conservation constraints*:

$$f^{in}(v) = f^{out}(v) \qquad \forall v \in V(G) \setminus (S \cup T) \tag{2.3}$$

A flow is *feasible* if in addition it satisfies the following two types of constraints:

$$0 \le f(e) \le c(e) \qquad \forall e \in E(G) \tag{2.4}$$

$$f^{in}(v) \le c(v) \qquad \forall v \in V(G) \setminus (S \cup T) \tag{2.5}$$

Constraints of the first type are *arc capacity constraints* and those of the second type are *vertex capacity constraints*. A flow $f$ *routes* $f(e)$ units of flow through the arc $e$. An arc $e \in \text{in}(v)$ *carries flow into* $v$ if $f(e) > 0$, and an arc $e' \in \text{out}(v)$ *carries flow out of* $v$ if $f(e') > 0$. We assume that $\min\{f(e), f(\text{rev}(e))\} = 0$ for every arc $e$.

**Maximum flow problem.** In the *maximum flow problem*, we are trying to find a feasible flow $f$ with maximum *value*, where the value $|f|$ of a flow $f$ is defined as

$$|f| = \sum_{s \in S} (f^{out}(s) - f^{in}(s)). \tag{2.6}$$

When all arc capacities are 1 and vertex capacities are infinite, the maximum flow problem becomes the *arc-disjoint $S$-$T$ paths problem*, and the value of the maximum flow is the maximum number of arc-disjoint paths from vertices in $S$ to vertices in $T$. Similarly, when all the vertex and arc capacities are 1, the maximum flow problem becomes the *vertex-disjoint*

*S-T paths problem.* In addition, we can define an undirected version of the maximum flow problem by requiring $c(\text{rev}(e)) = c(e)$ for all edges $e$ in undirected graphs. When $G$ is undirected and all edge capacities are 1 and vertex capacities are infinite, the maximum flow problem becomes the *edge-disjoint paths problem.*

Note that capacities may be infinite, and we can assume without loss of generality that terminals have infinite capacity: if a source $s$ has finite capacity $c$, then we can add a node $s'$, an arc $(s', s)$ of capacity $c$, replace $s$ with $s'$ in $S$, and let $s'$ have infinite capacity, all while preserving planarity. A similar reduction eliminates finite capacities on the sinks. Furthermore, we may assume without loss of generality that none of the sources have incoming arcs and none of the sinks have outgoing arcs.

**val**$(G)$ **and circulations.** Let $\text{val}(G)$ be the value of the maximum flow in a flow network $G$ (which may have vertex capacities). A *circulation* is a flow of value 0. A circulation $g$ is *simple* if $g^{in}(v) = g^{out}(v)$ for every terminal $v$. Non-simple circulations only exist if there are more than two terminals. A flow $f$ has a *flow cycle* $C$ if $C$ is a cycle and $f(e) > 0$ for every arc $e$ in $C$, and $f$ is *acyclic* if it has no flow cycles. A flow cycle $C$ of a flow $f$ is *unit* if $f(e) = 1$ for every arc $e$ in $C$. A flow $f$ *saturates* an arc $e$ if $f(e) = c(e)$. A flow is a *path-flow* if its support is a path.

**Basic operations.** We will often add two flows $f$ and $g$ together to obtain a flow $f + g$, or multiply a flow $f$ by some constant $c$ to get a flow $cf$. These operations are defined in the obvious way: for every arc $e$, we have

$$(f + g)(e) = \max\{0, f(e) + g(e) - f(\text{rev}(e)) - g(\text{rev}(e))\} \tag{2.7}$$

$$(cf)(e) = c \cdot f(e) \tag{2.8}$$

**The graph** $G_{st}$**.** Given a flow network with multiple sources and sinks, we can reduce the maximum flow problem to the single-source, single-sink case by adding a supersource $s$, supersink $t$, infinite-capacity arcs $(s, s_i)$ for every $s_i \in S$, and infinite-capacity arcs $(t_i, t)$ for every $t_i \in T$. Call the resulting flow network $G_{st}$. Finding a maximum flow in the original network $G$ is equivalent to finding a maximum flow from $s$ to $t$ in $G_{st}$. The graph $G_{st}$ is not necessarily planar but is a 2-apex graph. In this thesis, we will work in $G_{st}$ instead of $G$ when we want circulations to be unions of flow cycles; in $G$, circulations can consist of source-to-source or sink-to-sink paths.

**The flow graph $f_G$.** Given a flow $f$ in a flow network $G$, the *flow graph* of $f$ is a graph $f_G$ that contains all the vertices of $G$ but only contains the arcs of $G$ that carry non-zero flow; furthermore, each arc $e$ in $f_G$ has weight $f(e)$. Depending on the context, we will interpret these arc weights as either capacities or flow.

**The extended graph $G^\circ$.** Given a flow network $G$ with vertex capacities, Kaplan and Nussbaum [22] defined the *extended graph* $G^\circ$ based on constructions of Khuller and Naor [24], Zhang, Liang, and Jiang [25], and Zhang, Liang, and Chen [26]. Starting with $G_{st}$, we replace each finitely capacitated vertex $v \in V(G_{st})$ with an undirected cycle of $d$ vertices $v_1, \ldots, v_d$, where $d = |\text{in}(v)| + |\text{out}(v)|$ is the degree of $v$. Each edge in the cycle has capacity $c(v)/2$. (An undirected edge $e$ with capacity $c(e)$ can be viewed as two arcs $e$ and $\text{rev}(e)$, each with capacity $c(e)$, so $G^\circ$ can be viewed as a directed flow network.) We make every arc that was incident to $v$ incident to some vertex $v_i$ instead, such that each arc is connected to a different vertex $v_i$, the clockwise order of the arcs is preserved, and the graph remains planar. We also identify the new arc $(u, v_i)$ or $(v_i, u)$ with the old arc $(u, v)$ or $(v, u)$ and denote the cycle replacing $v$ by $C_v$. The resulting graph $G^\circ$ has $O(n)$ vertices and arcs. See Figure 2.1.

This idea of eliminating vertex capacities in planar graphs by replacing each vertex with a cycle has also been used in the context of finding shortest vertex-disjoint paths in planar graphs [13].

**The graph $\overline{G}$.** Given a flow network $G$ with vertex capacities, let $\overline{G}$ be the flow network obtained as follows: Starting with $G_{st}$, replace each capacitated vertex $v$ with two vertices $v^{in}$ and $v^{out}$, and add an arc of capacity $c(v)$ directed from $v^{in}$ to $v^{out}$. All arcs that were directed into $v$ are directed into $v^{in}$ instead, and all arcs that were directed out of $v$ are directed out of $v^{out}$ instead. See Figure 2.1. It is well known that every feasible flow in $G_{st}$ corresponds to a feasible flow in $\overline{G}$ of the same value, and vice versa. The graph $\overline{G}$ has $O(n)$ vertices and arcs.

**Restrictions and extensions.** Suppose $G$ and $H$ are flow networks such that every arc in $G$ is also an arc in $H$. If $f'$ is a flow in $H$, then the *restriction* of $f'$ to $G$ is the flow $f$ in $G$ defined by $f(e) = f'(e)$ for all arcs $e \in E(G)$. Conversely, if $f$ is a flow in $G$, then an *extension* of $f$ to $H$ is any flow $f'$ in $H$ such that $f(e) = f'(e)$ for every arc $e \in E(G)$.

Every arc in $G$ or $G_{st}$ is an arc in both $\overline{G}$ and $G^\circ$. Every feasible flow in $\overline{G}$ has a feasible restriction in $G$. Conversely, every feasible flow $f$ in $G$ has a feasible extension $\overline{f}$ in $\overline{G}$, by defining $\overline{f}(v^{in}, v^{out}) = f^{in}(v)$. Every feasible flow in $G^\circ$ has a restriction in $G$; this restriction is a flow but is not necessarily feasible. On the other hand, we have the following lemma:

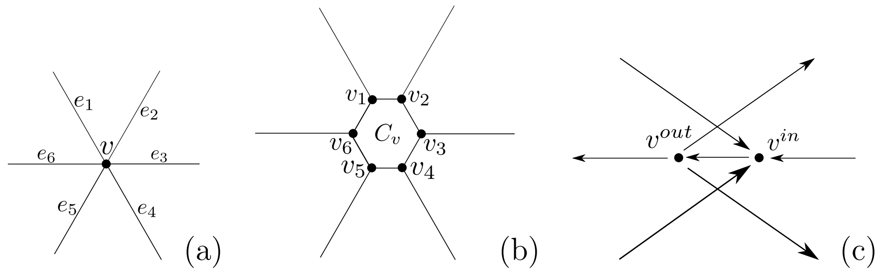


Figure 2.1: (a) capacitated vertex $v \in G$ with capacity $c(v)$ (b) corresponding cycle $C_v$ in $G^\circ$; each edge in $C_v$ has capacity $c(v)/2$ (c) corresponding arc $(v^{in}, v^{out})$ in $\overline{G}$ with capacity $c(v)$

**Lemma 2.1.** Every feasible flow $f$ in $G$ has an extension $f^\circ$ that is feasible in $G^\circ$. Furthermore, we can find $f^\circ$ in $O(n \log^3 n)$ time.

*Proof.* To show that $f^\circ$ exists, we use the well-known flow decomposition theorem, which states that any flow $f$ in $G$ can be decomposed into a sum of flows $f_1, \ldots, f_m$ such that for each $i$, the support of $f_i$ is either a cycle or a path from a source to a sink. For each $i \in [m]$, let $p_i$ be the support of $f_i$ and let $u_i = |f_i|$.

For each capacitated vertex $w \in G$, we define $f^\circ$ on the cycle $C_w$ in $G^\circ$ as follows: for each $i \in [m]$, if some arc in $p_i$ carries $u_i$ units of flow into a vertex $x$ on $C_w$ and another arc in $p_i$ carries $u_i$ units of flow out of a vertex $x'$ on $C_w$, then we route $u_i/2$ units of flow clockwise along $C_w$ from $x$ to $x'$ and $u_i/2$ units of flow counter-clockwise along $C_w$ from $x$ to $x'$. It is easy to see that $f^\circ$ satisfies conservation constraints. Since $f^{in}(C_w) \leq c(w)$, no arc on $C_w$ carries more than $c(w)/2$ units of flow, so $f^\circ$ is feasible.

We now describe how to find $f^\circ$. We must define $f^\circ(e) = f(e)$ for all arcs $e \in E(G)$. We reduce the problem of finding $f^\circ$ on all other arcs to finding a flow in a planar flow network $H$. Let $H$ be the subgraph of $G^\circ$ consisting of all cycles $C_v$ where $v$ is a capacitated vertex in $G$; it suffices to define $f^\circ$ on the arcs of $H$. Recall that for all $v \in V(G)$, the vertices in $C_v$ are $v_1, \ldots, v_d$ in clockwise order, where $d = \deg_G(v)$. For each vertex $v_i$ in $H$, let $e_{i,v}$ be the unique arc in $G$ incident to $v_i$. When it is clear what vertex $v$ is, we will write $e_i$ instead of $e_{i,v}$. For each $v \in V(G)$ and $i \in [\deg_G(v)]$, let

$$\text{demand}(v_i) = \begin{cases} -f(e_i) & \text{if } e_i \in \text{in}(v_i) \\ f(e_i) & \text{if } e_i \in \text{out}(v_i) \end{cases} \tag{2.9}$$

That is, demand$(v_i)$ is the net amount of flow that $f^\circ$ carries out of $v_i$ using only arcs in $E(G)$. For each vertex $v_i$ such that demand$(v_i)$ is negative, let $v_i$ be a source in $H$; similarly, if demand$(v_i)$ is positive, let $v_i$ be a sink in $H$. For each $v \in V(G)$, $\sum_{i=1}^{\deg_G(v)} \text{demand}(v_i) = 0$.

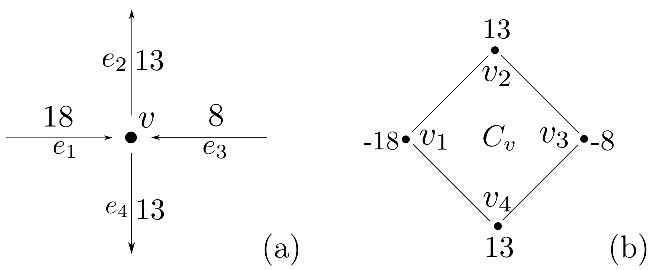


Figure 2.2: Extending a flow from $G$ to $G^\circ$. (a) An example of $f$ at $v$; arcs are labeled with their flow values (b) $H$ at $C_v$ with terminals labeled with their demand values; $v_1$ and $v_3$ are sources; $v_2$ and $v_4$ are sinks

See Figure 2.2.

Since $f^\circ$ exists, there exists a flow $f_H$ in $H$ such that $f_H^{out}(v_i) = -\text{demand}(v_i)$ for every source $v_i$ and $f_H^{in}(v_i) = \text{demand}(v_i)$ for every sink $v_i$. To actually find $f_H$, we do the following. For each source $v_i$ in $H$, we add a vertex $v_i'$ that will be a source instead of $v_i$, and we add an arc $(v_i', v_i)$ with capacity $-\text{demand}(v_i)$; similarly, for each sink $v_j$ in $H$, we add a vertex $v_j'$ that will be a sink instead of $v_j$, and we add an arc $(v_j, v_j')$ with capacity demand$(v_j)$. Then $f_H$ is an acyclic maximum flow in the resulting network. The restriction of $f_H$ to $H$ is exactly $f^\circ$ on the arcs of $H$. Finding $f_H$ requires finding a maximum flow in a union of disjoint "suns" with multiple sources and sinks, where a sun is a cycle in which each vertex has a pendant arc appended to it. This can be done in, say, $O(n \log^3 n)$ time using the algorithm of Borradaile et al. [23]. Simpler and more intuitive algorithms exist but are not necessary, as this will not the bottleneck when we use it. QED.

Thus given a flow in $G$ we can easily compute a corresponding flow in $G^\circ$, and vice versa.

**The residual graph.** If $f$ is a flow in a flow network $G$ with capacity function $c$ and without vertex capacities, then the *residual capacity* of an arc $e$ with respect to $f$ and $c$, denoted $c_f(e)$, is $c(e) - f(e) + f(\text{rev}(e))$. The *residual graph of $G$ with respect to $f$ and $c$* (or just the *residual graph of $G$ with respect to $f$* when $c$ is the capacity function given as input) has the same vertices and arcs as $G$, but each arc $e$ has capacity $c_f(e)$. A *residual arc* of $G$ with respect to $f$ is an arc with positive residual capacity, a *residual path* is a path made up of residual arcs, and a *residual cycle* is a cycle made up of residual arcs. It is well known that a flow $f$ is a maximum flow in a graph $G$ if the residual graph of $G$ with respect to $f$ does not have any residual paths from a source to a sink.

**Fractional and integer flows.** A flow $f^\circ$ in $G^\circ$ is an *integer flow* if $f^\circ(e)$ is an integer for every arc $e$ in $G$; otherwise, $f^\circ$ is *fractional.* A flow in $\overline{G}$ is *integer* if it is integer-valued on all arcs of $\overline{G}$ and is *fractional* otherwise.

We now prove the following lemma:

**Lemma 2.2.** Let $f^\circ$ be a fractional flow in $G^\circ$ such that $|f^\circ|$ is an integer. Then there exists an integer flow $f_1^\circ$ in $G^\circ$ of the same value as $f^\circ$ such that $(f_1^\circ)^{in}(C_v) \leq \lceil (f^\circ)^{in}(C_v) \rceil$ for every vertex $v \in V(G)$. Furthermore, we can find $f_1^\circ$ in $O(n \log^3 n)$ time.

First let $f$ be the restriction of $f^\circ$ to $G$, and let $\overline{f}$ be the extension of $f$ to $\overline{G}$. Since $\overline{G}$ has $O(n)$ arcs, results by Lee et al. [27] and by Kang and Payor [28] imply the following.

**Lemma 2.3.** Let $\overline{f}$ be a fractional flow in $\overline{G}$ such that $|\overline{f}|$ is an integer. Then there exists an integer flow $\overline{f_1}$ in $\overline{G}$ of the same value as $\overline{f}$ such that $\overline{f_1}(e) \leq \lceil \overline{f}(e) \rceil$ for every arc $e$ in $\overline{G}$. Furthermore, we can find $\overline{f_1}$ in $O(n \log n)$ time.

Compute $\overline{f_1}$ as in Lemma 2.3 and let $f_1$ be the restriction of $\overline{f_1}$ to $G$. Finally, we define $f_1^\circ$ to be an extension of $f_1$ to $G^\circ$; by Lemma 2.1, we can do this in $O(n \log^3 n)$ time.

We need to show that $f_1^\circ$ is the desired integer flow. Since $\overline{f_1}$ is an integer flow in $\overline{G}$, $f_1^\circ$ is an integer flow in $G^\circ$. Also, we have $|f_1^\circ| = |f_1| = |\overline{f_1}| = |\overline{f}| = |f| = |f^\circ|$. Lemma 2.3 implies that for every $v \in V(G)$, we have

$$\overline{f_1}(v^{in}, v^{out}) \leq \lceil \overline{f}(v^{in}, v^{out}) \rceil \tag{2.10}$$

$$\implies f_1^{in}(v) \leq \lceil f^{in}(v) \rceil \tag{2.11}$$

$$\implies (f_1^\circ)^{in}(C_v) \leq \lceil (f^\circ)^{in}(C_v) \rceil . \tag{2.12}$$

Thus we have described how to convert a fractional flow $f^\circ$ in $G^\circ$ to an integer flow $f_1^\circ$ in $G^\circ$ of the same value, such that for each $v \in V(G)$, the flow going into $C_v$ under $f_1^\circ$ is at most the flow going into $C_v$ under $f^\circ$ (assuming that $|f^\circ|$ is an integer and $G^\circ$ has integer arc capacities).

**Canceling flow-cycles.** We implicitly use three algorithms that allow us to assume without loss of generality that certain flows are acyclic. The first is by Kaplan and Nussbaum [22]:

**Lemma 2.4.** Suppose we are given a feasible flow $f^\circ$ in $G^\circ$. By canceling flow-cycles, we can compute in $O(n)$ time another feasible flow of the same value as $f^\circ$ whose restriction to $G$ is feasible and acyclic.

We describe this algorithm in more detail in Section 2.2.2. Using this first algorithm, we can assume that whenever we compute a flow in $G^\circ$, the restriction of that flow to $G$ is acyclic. The second algorithm we use implicitly is also by Kaplan and Nussbaum. Using the algorithm of Lemma 2.4, they show the following [22]:

**Lemma 2.5.** Suppose we are given a feasible flow in any planar graph. By canceling flow-cycles, we can compute in $O(n)$ time an acyclic flow of the same value as that of the given flow.

Using this second algorithm, we can assume that any flow we compute in a planar graph is acyclic. The third algorithm that we use implicitly is by Sleator and Tarjan [29]:

**Lemma 2.6.** Given a flow in a flow network with $O(n)$ vertices and arcs, we can compute another flow of the same value that is acyclic in $O(n \log n)$ time by canceling flow-cycles.

Using this third algorithm, we may assume that if we compute a flow in a graph with $O(n)$ arcs in $\Omega(n \log n)$ time, then that flow is acyclic.

#### 2.2.2 Proof sketch of Lemma 2.4

The purpose of this subsection is to describe the algorithm of Lemma 2.4. This is needed for the proof of Lemma 3.11. We will not prove the correctness of the algorithm, as that has been done elsewhere [22] [30].

The algorithm has three steps and is based on an algorithm of Khuller, Naor, and Klein [30] that finds a circulation without clockwise residual cycles in a directed planar graph in $O(n)$ time.

**Finding a circulation without clockwise residual cycles.** We describe the algorithm of Khuller, Naor, and Klein that finds a circulation $g$ in $G^\circ$ without clockwise residual cycles [30].

The graph $G^\circ \setminus \{s, t\}$ is planar. Let $h_\infty$ be the infinite face of $G^\circ \setminus \{s, t\}$, and let $h^*_\infty$ be its dual vertex. Using the algorithm of Henzinger et al. [31], compute the shortest path tree rooted at $h^*_\infty$ in $(G^\circ \setminus \{s, t\})^*$ in $O(n)$ time. For every face $h$ of $G$, let $\Phi(h)$ be the distance in $(G^\circ \setminus \{s, t\})^*$ from $h^*_\infty$ to $h^*$. For any arc $e \in E(G^\circ \setminus \{s, t\})$, we define $g(e)$ as follows. Let $h_\ell$ be the face on the left of $e$ and $h_r$ be the face on the right of $e$. If $\Phi(h_r) \geq \Phi(h_\ell)$, then set $g(e) = \Phi(h_r) - \Phi(h_\ell)$. Otherwise, set $g(e) = 0$ (and $g(\mathrm{rev}(e))$ will automatically be set to $\Phi(h_\ell) - \Phi(h_r)$). Khuller, Naor, and Klein [30] proved that the resulting flow function $g$ is a simple circulation in $G^\circ$ such that $G^\circ$ has no clockwise residual cycles with respect to $g$.

**Finding a flow without clockwise residual cycles.** Let $f^\circ$ be a feasible flow in $G^\circ$. We describe an algorithm due to Kaplan and Nussbaum [22] that computes a flow $f^\circ_1$ in $G^\circ$ with the same value as $f^\circ$ and without clockwise residual cycles. A symmetric algorithm can then compute a flow in $G^\circ$ with the same value as $f^\circ$ and without counterclockwise residual cycles.

Let $G^\circ_f$ be the residual graph of $G^\circ$ with respect to $f^\circ$. Using the algorithm of step 1, find a circulation $g$ in $G^\circ_f$ such that $G^\circ_f$ does not have clockwise residual cycles with respect to $g$. Now define $f^\circ_1 = f^\circ + g$. Computing $f^\circ_1$ takes $O(n)$ time. Kaplan and Nussbaum showed that $f^\circ_1$ is a feasible flow in $G^\circ$ with the same value as $f^\circ$ and without clockwise residual cycles [22].

**Finding an acyclic flow.** Finally, let $f^\circ$ be a feasible flow in $G^\circ$. We describe the algorithm due to Kaplan and Nussbaum [22] that computes a flow of the same value as $f^\circ$ whose restriction to $G$ is acyclic. We will do this by first eliminating counterclockwise flow-cycles to get a flow $f^\circ_1$; a symmetric algorithm then eliminates clockwise flow-cycles.

Define a new capacity function $c_1$ on the arcs of $G^\circ$ by first setting $c_1(e) = f^\circ(e)$ for $e \in E(G)$. This will ensure that we do not increase the flow along any arc of $G$. All other arcs in $G^\circ$ are in $C_v$ for some vertex $v$; for these arcs $e$ we set $c_1(e) = c(e) = c(v)/2$. Now we apply the previous algorithm to $G^\circ$ and $c_1$ to find a flow $f^\circ_1$ with the same value as $f^\circ$ such that there are no clockwise residual cycles in $G^\circ$ with respect to $f^\circ_1$ and $c_1$. Kaplan and Nussbaum [22] showed that the restriction of $f^\circ_1$ to $G$ does not contain counterclockwise flow-cycles.

We now repeat the previous procedure symmetrically, by defining a new capacity $c_2$ that restricts the flow on every arc $e$ of $G$ to be at most $f^\circ_1(e)$, and finding a circulation in $G^\circ$ without counterclockwise residual cycles. This way we get from $f^\circ_1$ a flow $f^\circ_2$ of the same value whose restriction to $G$ does not contain clockwise flow-cycles in $G$. For every $e \in E(G)$, we have $f^\circ_2(e) \leq f^\circ_1(e) \leq f^\circ(e)$, so we did not create any new flow-cycles when going from $f^\circ$ to $f^\circ_1$ to $f^\circ_2$. Thus $f^\circ_2$ is a feasible flow in $G^\circ$ with the same value as $f^\circ$ whose restriction to $G$ is feasible and acyclic.

### 2.2.3 Element-connectivity

Suppose we are given an undirected graph $G$ with two vertices $s$ and $t$. The maximum number of pairwise edge-disjoint paths from $s$ to $t$ is called the *edge-connectivity* between $s$ and $t$ in $G$, and we denote it by $\lambda_G(s,t)$. This quantity $\lambda_G(s,t)$ can be computed by any maximum flow algorithm. By the edge-connectivity version of Menger's theorem, $\lambda_G(s,t)$ is the minimum number of edges whose removal from the graph would disconnect $s$ from $t$ in $G$ [4]. The maximum number of pairwise vertex-disjoint paths from $s$ to $t$ is called the *vertex-connectivity* between $s$ and $t$ in $G$, and we denote it by $\kappa_G(s,t)$. If there is no edge connecting $s$ and $t$, then $\kappa_G(s,t)$ is the minimum number of vertices other than $s$ and $t$ whose removal would disconnect $s$ from $t$.

**Element connectivity.** Again suppose $G$ is an unweighted, undirected graph. Let $T \subset V(G)$ be a set of *terminals*, and in this context let $k = |T|$. Vertices not in $T$ are called *non-terminals*, and an *element* is an edge or a non-terminal. Let $\kappa'(u,v)$ denote the *element-connectivity* between two terminals $u$ and $v$; this is the minimum number of elements whose removal would disconnect $u$ from $v$, which by Menger's theorem is equal to the maximum number of element-disjoint paths between $u$ and $v$ [4]. Finally, the *global element-connectivity* $\kappa'(T)$ is $\min_{u,v\in T} \kappa'(u,v)$. This is the minimum number of elements whose deletion separates some pair of terminals.

By subdividing each edge between terminals using a new non-terminal, we can assume that $T$ is an independent set. In this case, $\kappa'(u,v)$ becomes the minimum number of non-terminals whose removal would disconnect $u$ from $v$. Element-connectivity is related to edge-connectivity and vertex-connectivity as follows. If we set $T = V$, then element-connectivity becomes edge-connectivity. Also, if $|T|$ contains exactly two vertices $s$ and $t$, then $\kappa'_G(s,t) = \kappa_G(s,t)$.

The element-connectivity between $u$ and $v$ is equal to the value of the maximum flow between $u$ and $v$ if every edge and non-terminal has unit capacity. Khuller and Naor [24] showed that when there is a single source and sink, one can eliminate vertex capacities while preserving the maximum-flow value and planarity by replacing each vertex of capacity $c(v)$ with a cycle of $\deg(v)$ bi-directed edges, each with capacity $c(v)/2$. Previously, we defined this new graph to be $G^\circ$. See Figure 2.1. The reduction shows that finding element-connectivity values in planar graphs reduces to finding edge-connectivity values in planar graphs.

**The reduced graph.** Element connectivity has applications in packing disjoint Steiner trees and forests because of a structure called the *reduced graph*, which we now describe. The reduced graph is known to exist because of the following theorem due to Chekuri and Korula [2].

**Theorem 2.1.** Let $G = (V,E)$ be an undirected graph and $T \subseteq V$ be a set of terminals. Let $e$ be any edge whose endpoints are non-terminals. Now let $G_1 = G \setminus e$ and $G_2 = G/e$ Then at least one of the following holds:

- For all $u, v \in T$, we have $\kappa'_G(u,v) = \kappa'_{G_1}(u,v)$.
- For all $u, v \in T$, we have $\kappa'_G(u,v) = \kappa'_{G_2}(u,v)$.

By repeatedly applying the theorem, we get:

**Corollary 2.1.** Let $G = (V, E)$ be an undirected graph and $T \subseteq V$ be a set of terminals. There is a minor $H = (V', E')$ of $G$ such that $T \subseteq V'$; both $T$ and $V' \setminus T$ are independent sets in $H$; and $\kappa'_H(u, v) = \kappa'_G(u, v)$ for all terminals $u, v$.

This graph $H$ is the reduced graph.

## 2.3 DISJOINT PATHS AND ORIENTATIONS

Suppose each edge $e \in E(G)$ has a non-negative length $\ell(e)$. The length of a walk $w$ in $G$, which we denote with $\ell(w)$, is the sum of the lengths of its edges, with appropriate multiplicity if $w$ is not a simple path. The total length of any set of walks $\mathcal{W}$, which we denote with $\ell(\mathcal{W}) = \sum_{w \in \mathcal{W}} \ell(w)$, is just the sum of their lengths. The distance from a vertex $u$ to a vertex $v$ in a graph $G$ is the length of a shortest walk from $u$ to $v$ and is denoted by $d_G(u, v)$; this walk will be a simple path.

In an abuse of terminology, we say that two directed paths are edge-disjoint if their underlying undirected paths are edge-disjoint (we assume each arc $uv$ is embedded together with its reversal $vu$).

In the *vertex-disjoint paths problem*, we are given a graph $G$ along with $k$ vertex pairs $(s_1, t_1), \ldots, (s_k, t_k)$, and we want to find $k$ pairwise vertex-disjoint paths connecting each node $s_i$ to the corresponding node $t_i$. In this context, the vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$ are called *terminals*. The problem is NP-hard [32] and is not to be confused with the previously-defined vertex-disjoint $S - T$ paths problem, which is a special case of the maximum flow problem and thus polynomial-time solvable. The *edge-disjoint paths problem* or *arc-disjoint paths problem* are defined similarly, except that we require the paths to be pairwise edge-disjoint or arc-disjoint, respectively, not vertex-disjoint. Note that for the vertex-disjoint paths problem, we may assume without loss of generality that the terminals are distinct, because otherwise the desired paths do not exist.

### 2.3.1 Shortest disjoint paths

In the *k-min-sum vertex-disjoint paths problem*, we are given a graph $G$, in which every edge $e$ has a non-negative real length $\ell(e)$, and $k$ pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$, and our goal is to compute vertex-disjoint paths $P_1, \ldots, P_k$, where each path $P_i$ is a path from $s_i$ to $t_i$, and the total length $\sum_{i=1}^{k} \ell(P_i)$ is as small as possible. (Here $\ell(P_i) = \sum_{e \in P_i} \ell(e)$.) The

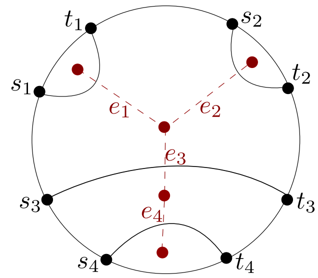


Figure 2.3: An example where $k = 4$ and the order of the terminals on the outer face is $s_1, t_1, s_2, t_2, t_3, t_4, s_4, s_3$. The solid black graph is $G_D \cup \partial G$, and the dashed red graph is the demand tree $T$. Non-terminal vertices on $\partial G$ have not been drawn. Edges $e_1, \ldots, e_4$ are red and dashed.

*k-min-min problem* is similar but requires us to minimize $\min_{i=1}^{k} \ell(P_i)$, while the *k-min-max problem* requires us to minimize $\max_{i=1}^{k} \ell(P_i)$.

Given an instance of the $k$-min-sum problem with all terminals on the outer face of $G$, we define the *demand graph* $G_D$ to be the graph whose vertices are the terminals of $G$ and that has an edge between $s_i$ and $t_i$ for all $i$; furthermore, the vertices of $G_D$ should be embedded in the same places as they are embedded in $G$, and the edges of $G_D$ should be embedded inside $\partial G$. We then define the *demand tree* $T$ as $T = (G_D \cup \partial G)^* \setminus (\partial G)^*$. Here, the union of two graphs $(V, E)$ and $(V', E')$ is $(V \cup V', E \cup E')$, $G^*$ is the dual graph of $G$, and $(\partial G)^*$ denotes the cocycle in $(G_D \cup \partial G)^*$ corresponding to the cycle $\partial G$ in $G_D \cup \partial G$. Each demand pair $(s_i, t_i)$ corresponds to an edge $s_i t_i$ in $G_D$ and an edge in $T$, which we denote by $e_i$. Two demand pairs $(s_i, t_i)$ and $(s_j, t_j)$ are *adjacent* if their corresponding edges $e_i$ and $e_j$ in $T$ are incident to a common vertex. If the demand tree $T$ is rooted, then a demand pair $(s_i, t_i)$ is a *child* of the demand pair $(s_j, t_j)$ if $e_i$ and $e_j$ are adjacent and both endpoints of $e_i$ are descendants of both endpoints of $e_j$; in this case $(s_j, t_j)$ is the *parent* of $(s_i, t_i)$. See Figure 2.3.

In Chapter 5, we will describe an algorithm that solves a 4-min-sum problem. We will assume that our given instance of 4-min-sum and every instance of 2-min-sum and 3-min-sum considered by our algorithm has a unique solution. If necessary, these uniqueness assumptions can be enforced with high probability using the isolation lemma of Mulmuley, Vazirani, and Vazirani [33]:

**Lemma 2.7** (Isolation Lemma)**.** Let $n$ and $N$ be positive integers, and let $\mathcal{F}$ be an arbitrary family of subsets of the universe $[n]$. Suppose each element $x \in [n]$ receives an integer weight $w(x)$, each of which is chosen independently and uniformly at random from $[N]$. With probability at least $1 - n/N$, there is a unique set in $\mathcal{F}$ that has minimum total weight among all sets of $\mathcal{F}$.

To enforce uniqueness, we define a new length function $\ell'(e) = \ell(e) + \varepsilon \cdot w(e)$, where $\varepsilon$ is a formal infinitesimal, and $w(e)$ is chosen uniformly at random from $[N]$. Equivalently, we consider lengths of edges—and by extension, lengths of paths—to be ordered pairs $(\ell, w)$, which we add as vectors and compare lexicographically. Now two (sets of) paths have equal perturbed length only if their actual lengths are equal *and* their infinitesimal perturbation terms are equal.

In our application of the isolation lemma, $\mathcal{F}$ is the family of vertex-disjoint paths connecting corresponding terminals in some $k$-min-sum instance. Our algorithm computes the solutions to $O(n^5)$ $k$-min-sum subproblems (each with $k \leq 3$). Thus, if we set $N = O(n^6)$, the Isolation Lemma implies that with probability at least $1 - \Omega(1/n)$, *all* feasible solutions to *all* such subproblems have distinct perturbation terms, which implies that all such subproblems have unique optimal solutions. In fact, however, we only require a constant number of these subproblems to have unique solutions, so it suffices to set $N = \Theta(n^2)$.

### 2.3.2 Orientation

In orientation problems, $G$ is a simple undirected plane graph, each edge $e \in E(G)$ has a positive length $\ell(e) > 0$, and $(s_1, t_1), \ldots, (s_k, t_k)$ are $k$ pairs of vertices in $G$. As usual, the vertices $s_1, t_1, \ldots, s_k, t_k$ are called *terminals*. An *orientation* of $G$ is a directed graph that is formed by assigning a direction to each edge in $G$. For any orientation $H$ of $G$, let $d'(u, v)$ be the distance from $u$ to $v$ in $H$. In the *orientation problem*, we want to find an orientation $H$ of $G$ such that for all $i \in \{1, \ldots, k\}$, $t_i$ is reachable from $s_i$ in $H$. In the *ideal orientation problem*, we want to find an orientation $G'$ of $G$ such that for all $i$, $d(s_i, t_i) = d'(s_i, t_i)$.

**Nonconflicting paths.** It is possible to reformulate the ideal orientation problem in terms of finding *nonconflicting* shortest paths; we will use this reformulation in Chapter 6. Two directed walks $P$ and $Q$ in $G$ *conflict* if there is an edge $\{u, v\}$ in $G$ such that $uv$ is an arc in $P$ and $vu$ is an arc in $Q$. Two walks are *nonconflicting* if they do not conflict. The ideal orientation problem then asks us to find pairwise nonconflicting directed walks $P_1, \ldots, P_k$ such that $P_i$ is a shortest path from $s_i$ to $t_i$ for all $i \in \{1, \ldots k\}$. We call the set of such paths a *solution* to the instance.

**$k$-min-sum, $k$-min-max, and $k$-min-min problems.** In the *$k$-min-sum orientation problem*, the input is the same as the input to the ideal orientation problem, and we still want to find paths $P_1, \ldots, P_k$ such that $P_i$ connects $s_i$ to $t_i$, but now our goal is to minimize the sum of the lengths of the paths $P_i$ instead of insisting that each $P_i$ be a shortest path.

The $k$-min-sum orientation problem is at least as hard as the ideal orientation problem. The *$k$-min-max orientation problem* is the same as the $k$-min-sum orientation problem except that we want to minimize the length of the longest path $P_i$ instead of the sum of the lengths of the paths, the *$k$-min-min orientation problem* requires us to minimize the length of the shortest path $P_i$.

### 2.3.3 Partially edge-disjoint noncrossing paths

In the partially vertex-disjoint paths problem (PVPP), we are given a directed planar graph $H$, vertices $u_1, v_1, \ldots, u_h, v_h$; subgraphs $H_1, \ldots, H_h$ of $H$; and a set $S$ of pairs $\{i, j\}$ from $\{1, \ldots, h\}$. We wish to find directed paths $Q_1, \ldots, Q_h$ such that

- $Q_i$ connects $u_i$ to $v_i$ for all $i$,
- $Q_i$ is in $H_i$ for all $i$, and
- for all $i, j$, if $\{i, j\} \in S$ then $Q_i$ and $Q_j$ are vertex-disjoint.

Note that we do not require the paths to be shortest paths; in fact the graph $H$ is unweighted. Schrijver [34] solved the partially vertex-disjoint paths problem for fixed $h$ in polynomial time. He does not state the running time of the algorithm, but it appears to be $(\text{poly}(|V(H)|))^{h^2}$.

**Partially noncrossing edge-disjoint paths problem (PNEPP).** PNEPP is the same as PVPP except that if $\{i, j\} \in S$, then we require the directed paths $Q_i$ and $Q_j$ to be noncrossing edge-disjoint paths instead of vertex-disjoint paths. (Recall that by "edge-disjont" we mean that if $Q_i$ uses $e$ then $Q_i$ can use neither $e$ nor $\text{rev}(e)$.)

**Reduction.** We now describe a polynomial-time reduction from PNEPP to PVPP. Suppose we are given an instance $H$ of PNEPP with terminal pairs $(u_1, v_1), \ldots, (u_h, v_h)$ and a set $S$ of pairs of indices of terminals, and subgraphs $H_1, \ldots, H_h$. We construct an instance $H'$ of PVPP by replacing each non-terminal vertex $v$ with a $2h \times 2h$ grid $g_v$ of bidirected edges, where $n = |V(G)|$. (The grid can be made smaller, say $p_v \times p_v$ where $p_v = \max\{k, \deg(v)\}$, but this suffices for our purposes.) Every arc that was incident to vertex $v$ in $H$ is instead incident to a vertex on the boundary of $g_v$; furthermore, we can make it so that no two arcs in $H$ share endpoints in $H'$. See Figure 2.4. The subgraphs $H_1, \ldots, H_h$ and the terminals $s_1, t_1, \ldots, s_h, t_h$ are the same in $H'$ and $H$. To show that this reduction is correct we need to prove the following lemma:

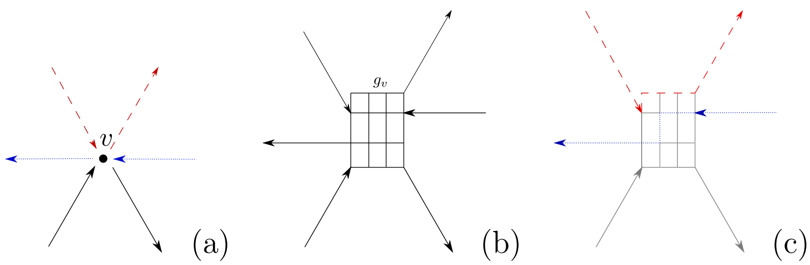


Figure 2.4: (a) a dashed red path and a dotted blue path going through a vertex $v$ in $G$ (b) corresponding grid $g_v$ in $H$ with $k = 2$. (c) routing the dashed red path and dotted blue path through $g_v$, in the proof of Lemma 2.8

**Lemma 2.8.** The following two statements are equivalent:

1. In $G$, there exist paths $P_1, \ldots, P_h$ such that $P_i$ connects $u_i$ to $v_i$, $P_i$ is in $H_i$ for all $i$, and if $\{i, j\} \in S$ then $P_i$ and $P_j$ are noncrossing and edge-disjoint.

2. In $H$, there exist paths $Q_1, \ldots, Q_h$ such that $Q_i$ connects $u_i$ to $v_i$, $Q_i$ is in $H_i$ for all $i$, and if $\{i, j\} \in S$ then $Q_i$ and $Q_j$ are vertex-disjoint.

*Proof.* $\Rightarrow$: Suppose that noncrossing partially edge-disjoint paths $P_1, \ldots, P_h$ exist in $G$. We construct paths $Q_1, \ldots, Q_h$ as follows. For any arc $e$ in $P_i$, we add $e$ to $Q_i$. This defines the portions of the paths $Q_1, \ldots, Q_h$ outside the grids $g_v$; these portions are vertex-disjoint because by construction the endpoints of $G$ are all distinct.

To find the portions of $Q_1, \ldots, Q_h$ inside a single grid $g_v$, we need to solve the following problem. Suppose $k'$ of the paths $P_1, \ldots, P_h$ went through $v$ in $G$. Re-index the paths such that $P_1, \ldots, P_{k'}$ go through $v$ and $P_{k'+1}, \ldots, P_h$ do not. We are given a subgraph $g$ of the $n \times n$ bidirected grid with $k'$ pairs of noncrossing terminals $(w_1, x_1), \ldots, (w_{k'}, x_{k'})$ on the boundary of $g$, and we want to find pairwise vertex-disjoint paths in $g$ such that the $i$-th path $\pi_i$ connects $w_i$ to $x_i$. To solve this problem, we route the paths one by one as follows. List the terminals $w_1, x_1, \ldots, w_{k'}, x_{k'}$ in cyclic order around the outer face of $g$; there must be some $i$ such that the two vertices $w_i$ and $x_i$ appear consecutively in this list. Terminals $w_i$ and $x_i$ split the boundary of $g$ into two segments; we let $\pi_i$ be the portion of the boundary that does not contain any other terminals. Remove the vertices of $\pi_i$ from $g$ and recursively compute the other paths $\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_{h'}$.

Routing $\pi_i$ is possible as long as $g$ is connected. Each time we recurse, the outerplanarity index of the $g$ goes down by at most 1. Initially, $g$ is the $2h \times 2h$ grid, so the outerplanarity index of $g$ starts at $h \geq k'$. Thus our recursive algorithm is able to connect all the pairs $(x_1, w_1), \ldots, (x_{h'}, w_{h'})$.

$\Leftarrow$: Suppose partially vertex-disjoint paths $Q_1, \ldots, Q_h$ exist in $H$. Trivially, the paths $Q_1, \ldots, Q_h$ are noncrossing partially edge-disjoint too. Each path $P_i$ can be defined to be the "projection" of $Q_i$ into $G$ in the obvious way: an arc $e$ of $G$ is in $P_i$ if and only if $e$ was in the original path $Q_i$.

The paths $P_1, \ldots, P_k$ are noncrossing because the paths $Q_1, \ldots, Q_k$ are noncrossing. We now show that the paths $P_1, \ldots, P_k$ are pairwise edge-disjoint. Suppose for the sake of argument that $P_i$ and $P_j$ share an arc $uv$. Arc $uv$ is in the original graph $G$, so it must connect the grid $g_u$ to the grid $g_v$. Since there is only one edge in $H$ that connects $g_u$ to $g_v$, this means that $Q_i$ and $Q_j$ both use this arc, and so are not vertex-disjoint. QED.

The reduction clearly runs in polynomial time.

# CHAPTER 3: MAXIMUM FLOW WITH VERTEX CAPACITIES

As noted in the introduction, the maximum flow problem was introduced in the 1950s by Harris and Ross [6], who were studying the Soviet rail network in Eastern Europe and wanted to compute how much of certain commodities the network could transport from certain cities to other cities. They modeled the rail system as a directed graph: the rail lines become edges and the interchanges become vertices. The quantity they wanted to compute was how much traffic (or flow) the network could handle. Ford and Fulkerson gave an algorithm that solved this problem, and also proved a duality theorem for the maximum flow problem called the *max-flow min-cut theorem* [5]. Given a flow network $G$ with a single source $s$ and single sink $t$, we define a *cut* to be a set of edges, arcs, and vertices whose removal from $G$ would destroy all paths from $s$ to $t$, and the *capacity of a cut* is the sum of the capacities of the edges, arcs, and vertices in that cut. The *minimum cut* is the cut with minimum capacity. The max-flow min-cut theorem says that the value of the maximum flow is always equal to the capacity of the minimum cut. A special case of this theorem is the arc-connectivity version of *Menger's theorem* [4], which states that the maximum number of arc-disjoint paths connecting $s$ to $t$ in a directed graph is equal to the minimum number of arcs whose removal would destroy all paths from $s$ to $t$.

Since Ford and Fulkerson, many researchers have worked on finding fast algorithms for the maximum flow problem, some of which we use in this thesis. Ford and Fulkerson themselves described an augmenting-path algorithm computing maximum flows in general directed graphs with integer capacities in $O(mU^*)$ time, where $m$ is the number of arcs in the flow network and $U^*$ is the value of the maximum flow. Currently, the fastest algorithms for general graphs run in $O(mn)$ time for real capacities [20], $\tilde{O}(m^{10/7})$ time for unit capacities [36], and $O(m \min(n^{2/3}, \sqrt{m}) \log \frac{n^2}{m} \log U)$ time for integer capacities [37], where $U$ is the maximum capacity in $G$. (Here and in the rest of this proposal, $m$ is the number of edges or arcs in the input graph and $n$ is the number of vertices.)

For planar graphs, even faster algorithms are known; we first list some results for the case where there are no vertex capacities. Maximum flows can be found in $O(n)$ time if $G$ is directed $st$-planar, meaning that $s$ and $t$ are on the same face [31]. They can be found in $O(n)$ time if $G$ is directed planar with unit arc capacities [38, 39] (more generally, in $O(nU)$ time if the capacities are integers bounded by $U$). They can be found in $O(n \log \log n)$ time if $G$ is undirected planar [40], in $O(n \log n)$ time if $G$ is directed planar [41], and in $O(n \log^3 n)$ time if $G$ is directed planar and has multiple sources and sinks [23]. In surface graphs of

The results in this chapter appeared in SODA 2019 [35].

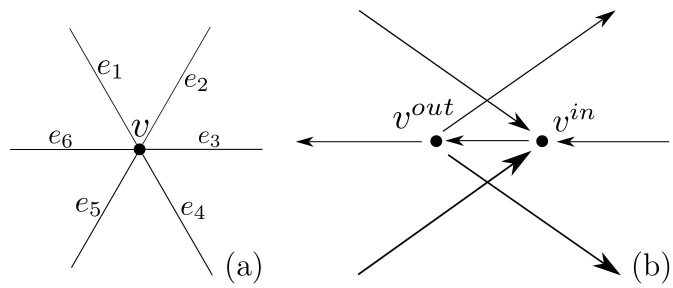


Figure 3.1: (a) before Ford and Fulkerson's reduction eliminating capacity of $v$ [5] (b) after reduction

genus $g$, they can be found in $g^{O(g)}n^{3/2}$ time for real capacities and $O(g^8 n \text{ polylog}(nU))$ time for integer capacities, where again $U$ is the maximum capacity [42]. They can be found in $O(\alpha^3 n \log^3 n)$ if $G$ is an $\alpha$-apex graph (i.e., a graph that can become planar after the removal of $\alpha$ vertices) [23]. They can be found in $O(\beta^3 n \log n)$ time if $G$ is the union of $\beta$ planar graphs with $\beta$ shared vertices [43]. Furthermore, in graphs with multiple sources and sinks, they can be found with a simple pair of nested for-loops: For all sources $s_i$, for all sinks $t_j$, compute the maximum flow from $s_i$ to $t_j$ in the current residual graph [44]. Combining this with previously mentioned algorithms ( [38] [39] [41]) yields algorithms for maximum flows in directed planar graphs with multiple sources and sinks that run in $O(k^2 nU)$ time for integer weights bounded by $U$ and in $O(k^2 n \log n)$ time for real weights. Finally, finding flows in undirected graphs reduces to finding flows in directed graphs because each undirected edge $\{u, v\}$ of capacity $c$ can be modeled as two directed arcs $uv$ and $vu$ of capacity $c$.

The results in the previous paragraph solve the maximum flow problem when there are only edge or arc capacities. In this chapter we are concerned with the case where vertices of the graph also have capacities, which limit the amount of commodity that can go through that vertex. In general directed graphs, adding capacities to the vertices does not make the problem any harder because of a reduction first suggested by Ford and Fulkerson [5]. For each vertex $v$ with finite capacity $c$, we do the following. Replace $v$ with two vertices $v^{in}$ and $v^{out}$, and add an arc of capacity $c$ directed from $v^{in}$ to $v^{out}$. All arcs that were directed into $v$ are directed into $v^{in}$ instead, and all arcs that were directed out of $v$ are directed out of $v^{out}$ instead. See Figure 3.1. Unfortunately, this reduction does not preserve planarity. Consider the complete directed graph on four vertices. This graph is planar, but if we apply the reduction of Ford and Fulkerson on any single vertex, we get a graph whose underlying undirected graph is $K_5$, which is not planar by Kuratowski's Theorem.

When there are vertex capacities, prior work has only been able to exploit planarity in the cases where there is a single source and sink or when the number of vertices with capacities is fixed. Khuller and Naor [24] were the first to consider the case where there is a single source and sink. Currently, the best known algorithm for this case is due to Kaplan and Nussbaum [22], who described an algorithm for maximum flow in directed planar graphs with vertex capacities that runs in $O(n \log n)$ time. In doing so, they fixed a flaw in a paper of Zhang, Liang and Chen [26]. They also give an algorithm that runs in $O(n)$ time when all vertex and arc capacities are unit, solving the vertex-disjoint paths problem in directed planar graphs with a single source and sink. This extended a result by Ripphausen-Lipa, Wagner, and Weihe that only applied to undirected graphs [45]. Zhang, Liang, and Chen [26] described an algorithm that finds maximum flows in undirected $st$-planar graphs with vertex capacities in $O(n)$ time.

In the case of multiple sources and sinks, Borradaile et al. give an algorithm that runs in $O(\alpha^3 n \log^3 n)$ time, where $\alpha$ is the number of vertex capacities [23]. For arbitrary numbers of terminals and vertex capacities, the best-known algorithm prior to the results described in this chapter uses the Ford-Fulkerson reduction described earlier to eliminate vertex capacities, connects a super-source to all sources, connects all sinks to a supersink, and then in the resulting graph applies either Mądry's algorithm [36] for finding maximum flows in unit-capacity networks, Goldberg and Rao's algorithm [37] for finding maximum flows in integer-capacity networks, or Orlin's algorithm [20] for finding maximum flows in sparse real-capacity networks. The resulting algorithm runs in $\tilde{O}(n^{10/7})$ time for unit capacities, $O(n^{3/2} \log n \log U)$ time for integer capacities where $U$ is the largest capacity, and $O(n^2/\log n)$ time for real capacities. Since the work in this chapter first appeared, Liu and Sidford [46] have sped up Mądry's algorithm; thus they are able to compute maximum flows in planar graphs with unit capacities and vertex capacities in $O(n^{4/3+o(1)})$ time.

Maximum flows in directed planar graphs *without* vertex capacities can be computed in near-linear time [23], and one expects to be able to achieve the same time bound even when the graph has arbitrarily many vertex capacities. However, doing so seems to be difficult. The techniques for computing flows in planar graphs without vertex capacities rely heavily on the use of residual graphs, which do not exist when there are vertex capacities. The only tool we have for dealing with vertex capacities in planar graphs is Kaplan and Nussbaum's reduction, but that reduction fails when there are multiple sources and sinks because of *saddles*, which we explain in section 3.1. Even for the case where there are two sources and one sink and all vertex capacities are unit, no near-linear-time algorithms were previously known.

In this chapter, we extend Kaplan and Nussbaum's algorithm to directed planar graphs with integer capacities and a fixed number of sources and sinks. The key observation is that when there are multiple sources and sinks, applying their algorithm results in a flow that is infeasible at only $k-2$ vertices, where $k$ is the number of terminals. For each of these infeasible vertices, we define the excess of the vertex to be the amount by which it is infeasible, and we show that the sum of the excesses of all the infeasible vertices is at most $(k-2)U$. This means that when $U$ is small, the flow returned by Kaplan and Nussbaum's algorithm is close to feasible. We exploit this observation to obtain the following:

**Theorem 3.1.** Let $G$ be a directed planar graph with $k$ terminals and with integer capacities on both arcs and vertices. If all capacities are bounded by a constant, then we can find a maximum flow in $G$ in $O(\min\{k^2 n, n\log^3 n + kn\})$ time.

Thus when $k$ is fixed, we can solve the vertex-disjoint paths problem in directed planar graphs with multiple sources and sinks in near-linear time.

Our second algorithm deals with the case where $U$ may be unbounded. The basic idea is a scaling algorithm. First we guess the value of the maximum flow using binary search; this increases the running time of the algorithm by a factor of $O(\log(nU))$. Starting with a flow with $k-2$ infeasible vertices, we find a way to improve the flow that decreases the maximum excess of the vertices by some factor that depends only on $k$. The improved flow has the same value as the original flow. We show that after $O(k\log(kU))$ improvement phases, each infeasible vertex has $O(k)$ excess. As in the first algorithm, we exploit this observation to obtain the following:

**Theorem 3.2.** Let $G$ be a directed planar graph with $k$ terminals and with integer capacities on both arcs and vertices. If $U$ is the maximum capacity of a single vertex or arc, then we can find a maximum flow in $G$ in $O(k^5 n \text{ polylog}(nU))$ time.

Our third algorithm deals with the special case where $k = 3$. In this case, the fact that there is only one infeasible vertex considerably simplifies the problem, since we can just focus on decreasing the excess of this one vertex without worrying about trade-offs. (Roughly speaking, if there is more than one infeasible vertex, we have to consider that decreasing the excess of one vertex could increase the excess of another vertex.) We show that we can modify our second algorithm such that only one improvement phase is necessary. This third algorithm works even if the capacities are arbitrary real numbers instead of integers.

**Theorem 3.3.** Given a directed planar graph $G$ with three terminals and with capacities on both arcs and vertices, we can find a maximum flow in $G$ in $O(n\log n)$ time.

The outline of this chapter is as follows. In Section 3.1, we prove the structural properties that show that Kaplan and Nussbaum's algorithm almost works when there are multiple sources and sinks. In section 3.2, we describe our algorithm for the case where capacities are integers bounded by a constant. In Section 3.3, we use this algorithm to solve the case of arbitrary integer capacities. In Section 3.4, we describe the modifications to the algorithms that are necessary for the case when $k = 3$ and the capacities are arbitrary reals.

## 3.1 SADDLES AND EXCESS

Suppose $f^\circ$ is a feasible flow in $G^\circ$ whose restriction $f$ to $G$ is acyclic. It is easy to see that $f$ satisfies conservation and arc capacity constraints. In this section, we show that $f$ violates at most $|S| + |T| - 2$ vertex capacity constraints.

Let $f_G$ be the flow graph of $f$. For any vertex $v$ in $f_G$, the *alternation number* of $v$, denoted by $\alpha(v)$, is the number of direction changes (i.e., from in to out or vice versa) of the arcs incident to $v$ as we examine them in clockwise order. Thus $\alpha(u) = 0$ for all terminals $u$, and the alternation number of any vertex is even. A vertex $v$ is a *saddle in $f$* if $\alpha(v) \geq 4$. We let index($v$) denote the *index* of $v$ and define it by $\text{index}(v) = \alpha(v)/2 - 1$.

Since $f_G$ is a plane directed acyclic graph, a result of Guattery and Miller [47] implies the following:

**Lemma 3.1.** If $f_G$ has $k_1$ sources and $k_2$ sinks, then the sum of the indices of the saddles in $f_G$ is at most $k_1 + k_2 - 2$. In particular, a vertex in $f_G$ is a saddle if and only if it has positive index, so $f_G$ has at most $k - 2$ saddles.

*Proof.* First we need a few definitions. For any face $\phi$ in $f_G$, let $\alpha(\phi)$ denote the alternation number of $\phi$; $\alpha(\phi)$ is the number of times the arcs on the boundary of $\phi$ change direction as we traverse this boundary. Thus $\alpha(\phi) = 0$ if the arcs on the boundary of $\phi$ form a directed cycle. We use index($\phi$) to denote the index of a face $\phi$, which is defined by $\text{index}(\phi) = \alpha(\phi)/2 - 1$.

Now we can proceed with the proof. See Figure 3.2. If at each vertex $v$ in $f_G$ we cycle through its incident arcs in order according to the embedding of $f_G$, each transition from one arc $e$ to the next arc $e'$ results in exactly one alternation either for $v$ or for the face on whose boundary the two arcs $e$ and $e'$ lie. Thus

$$2E = \sum_{v \in V(f_G)} \alpha(v) + \sum_{\phi \in F(f_G)} \alpha(\phi) \tag{3.1}$$

$$\implies E = \sum_{v \in V(f_G)} (\text{index}(v) + 1) + \sum_{\phi \in F(f_G)} (\text{index}(\phi) + 1) \tag{3.2}$$

$$\implies E = V + F + \sum_{v \in V(f_G)} \text{index}(v) + \sum_{\phi \in F(f_G)} \text{index}(\phi) \tag{3.3}$$

$$\implies -2 = \sum_{v \in V(f_G)} \text{index}(v) + \sum_{\phi \in F(f_G)} \text{index}(\phi) \tag{3.4}$$

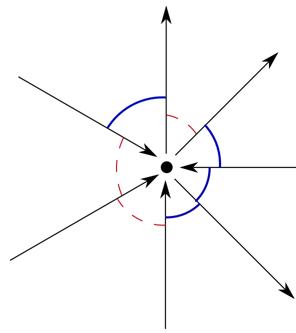

Figure 3.2: Proof of Lemma 3.1. Solid blue transitions contribute one alternation to a vertex; dashed red transitions contribute one alternation to a face.

where in the last line we have used Euler's formula $V(f_G) - E(f_G) + F(f_G) = 2$. Since $f_G$ is acyclic, $\text{index}(\phi) \geq 0$ for each face $\phi$, so $-2 \geq \sum_{v \in V(f_G)} \text{index}(v)$. Finally, $\text{index}(v) = -1$ for each terminal $v$, so

$$k_1 + k_2 - 2 \geq \sum_{v:\text{index}(v) \geq 1} \text{index}(v). \tag{3.5}$$

A vertex $v$ is a saddle if and only if $\text{index}(v) \geq 1$, so this shows that the sum of the indices of the saddles in $f_G$ is at most $k_1 + k_2 - 2$. QED.

A vertex $v \in V(G)$ is *infeasible* under the flow $f$ if $f^{in}(v) > c(v)$ and is *feasible* otherwise. For any vertex $v \in V(G)$, let $\text{ex}(f^\circ, v)$ and $\text{ex}(f, v)$ denote the *excess* of the vertex $v$ under $f^\circ$ or $f$:

$$\text{ex}(f^\circ, v) = \text{ex}(f, v) = \max\{0, f^{in}(v) - c(v)\} \tag{3.6}$$

The excess of a vertex is positive if and only if the vertex is infeasible. We also define $\text{ex}(f^\circ) = \text{ex}(f) = \max_{v \in V(G)} \text{ex}(f, v)$. We say that $f$ has excess $\text{ex}(f, v)$ on $v$.

**Lemma 3.2.** Let $\text{index}(v)$ be defined for each vertex $v$ in $G$ using the flow graph $f_G$ of $f$. For each vertex $v$ in $f_G$, we have $\text{ex}(f, v) \leq \text{index}(v)c(v)$.

*Proof.* Let $f_G^\circ$ be the flow graph of $f^\circ$. We have $\alpha(v) = 2 \cdot \text{index}(v) + 2$. Thus, if we examine the arcs in $f_G$ incident to $v$ in clockwise order, there are $\text{index}(v) + 1$ groups of consecutive incoming arcs. Consider such a group of consecutive incoming arcs $(u_i, v), \ldots, (u_j, v)$ in $f_G$. We can view these as arcs $(u_i, v_i), \ldots, (u_j, v_j)$ in $f_G^\circ$, where $v_i, \ldots, v_j$ are consecutive vertices in $C_v$. In $f_G^\circ$, the only two arcs in $\text{out}(\{v_i, \ldots, v_j\})$ are $(v_i, v_{i-1})$ and $(v_j, v_{j+1})$, which have total capacity $c(v)$. Thus, for each vertex $v$ in $f_G$, each group of consecutive incoming arcs in $f_G$ has total weight at most $c(v)$. This shows that $f^{in}(v) \leq (\text{index}(v) + 1)c(v)$ for any vertex $v$, from which the lemma follows. QED.

Combining Lemmas 3.1 and 3.2, we see that the sum of the excesses of all vertices under $f$ is at most $(k-2)U$. Lemma 3.2 implies that $f$ is only infeasible at saddles of $f_G$.

## 3.2 BOUNDED-INTEGER-CAPACITY CASE

Suppose that all vertex and arc capacities are integers less than some constant $U$. Let $f^\circ$ be an integral maximum flow in $G^\circ$, and let $f$ be its restriction to $G$. By Lemma 2.4 we may assume without loss of generality that $f$ is acyclic. Since the capacities of $G^\circ$ are integers and half-integers bounded by $U$, computing $f$ takes $O(n \log^3 n)$ time using the algorithm of Borradaile et al. [23] or $O(k^2 nU)$ time using $O(k^2)$ invocations of the algorithm of Brandes and Wagner [38] or of Eisenstat and Klein [39]. The flow $f$ may be infeasible at up to $k-2$ vertices $x_1, \ldots, x_{k-2}$. By Lemma 3.1 and 3.2, the sum of the excesses of the infeasible vertices is at most $(k-2)U$. After finding $f$, the algorithm has two steps.

**Step 1.** In this step, we remove $\mathrm{ex}(f, x_i)$ units of flow through each infeasible vertex $x_i$ to get a feasible flow $f_1$ in $G$. The flow $f_1$ will have lower value than $f$. To do this, let $f_G$ be the flow graph of $f$. The graph $f_G$ is a directed acyclic graph; let $v_1, \ldots, v_n$ be a topological ordering of $f_G$. To remove all of the excess flow through an infeasible vertex $x_i$, we do the following:

- Push $\mathrm{ex}(f, x_i)$ units of flow back from $x_i$ to the sources of $f_G$, as follows. Process the vertices in $f_G$ in the order $v_n, \ldots, v_1$. To process a vertex $v_j$, check whether there is too much flow going through $v_j$ (either because $v_j = x_i$ or because flow conservation is violated at $v_j$). If so, then decrease the flow on incoming arcs of $v_j$ in $f_G$ until there is no longer too much flow going into $v_j$ (i.e., until the flow going into $v_j$ is at most $c(v_j)$ if $v_j = x_i$, or until flow is conserved at $v_j$ if $v_j \neq x_i$).
- Pull $\mathrm{ex}(f, x_i)$ units of flow back to $x_i$ from the sinks of $f_G$, using a similar algorithm as in the previous bullet point.

Let $f_1$ be the resulting feasible flow. Removing excess flow through a vertex $x_i$ takes $O(n)$ time, so step 1 takes $O(kn)$ time.

**Step 2.** Let $\overline{f_1}$ be the extension of $f_1$ to $\overline{G}$. In this step, we do the following:

- Compute a maximum flow $\overline{f_2}$ in the residual graph of $\overline{G}$ with respect to $\overline{f_1}$ using the classical Ford-Fulkerson algorithm.

- Return the restriction of $\overline{f_1} + \overline{f_2}$ to $G$.

Since $\overline{f_2}$ is a maximum flow in the residual graph of $\overline{G}$ with respect to $\overline{f_1}$, we see that $\overline{f_1} + \overline{f_2}$ is a maximum flow in $\overline{G}$. It follows that the restriction of $\overline{f_1} + \overline{f_2}$ to $G$ is a maximum flow in $G$, as desired.

We have $\text{val}(\overline{G}) \leq \text{val}(G^\circ) = |f| \leq |f_1| + (k-2)U$. Thus the value of $\overline{f_2}$ is at most $(k-2)U$, so computing $\overline{f_2}$ takes $O(knU)$ time. Hence step 2 takes $O(knU)$ time. Thus if $U$ is bounded by a constant, the entire algorithm runs in $O(\min\{k^2 n, n\log^3 n + kn\})$ time.

### 3.3 INTEGER CAPACITIES

Suppose all vertex and arc capacities are integers. Let $\lambda^* = \text{val}(G)$. The basic structure of the algorithm is as follows:

- Guess $\lambda^*$ via binary search.

  1. Suppose we guess the value of the maximum flow of $G$ to be $\lambda$. Find a flow $f^\circ$ in $G^\circ$ of value $\lambda$. By Lemma 2.4, we may assume that the restriction $f$ of $f^\circ$ to $G$ is acyclic.
  2. While $\text{ex}(f) > 2k$, improve $f$.
  3. Fix $f$ using the algorithm from section 3.2.

One can see that the algorithm has three main steps which we call *phases*. In phase 2, improving $f$ means that we find a flow $f_2$ of the same value as $f$ such that

$$\text{ex}(f_2) \leq \left\lceil \frac{k-1}{k} ex(f) \right\rceil . \tag{3.7}$$

We then set $f$ to be the new flow $f_2$. We will eventually show that a single improvement of $f$ can be done in $O(k^4 n \log^3 n)$ time. In phase 3, fixing $f$ means that we remove $\text{ex}(f,x)$ units of flow through each infeasible vertex $x$ to get flow $f'$, extend $f'$ to a flow $\overline{f'}$ in $\overline{G}$, and then use the Ford-Fulkerson algorithm to find a maximum flow $\overline{f''}$ in the residual graph of $\overline{G}$ with respect to $\overline{f'}$; we then set $f$ to be the restriction of $\overline{f'} + \overline{f''}$ to $G$. To do the binary search, we use the fact that $\lambda \leq \lambda^*$ if the result of phase 3 is a feasible flow of value $\lambda$, and $\lambda > \lambda^*$ if either phase 2 fails or if the flow that results from phase 3 has value less than $\lambda$.

Before we describe how phase 2 is implemented, let us analyze the running time of the algorithm. If $U$ is the maximum capacity of a single vertex, then $\lambda^* \leq nU$, so the binary search for $\lambda^*$ only requires $O(\log(nU))$ guesses. Computing $f^\circ$ in phase 1 takes $O(n \log^3 n)$

time using the algorithm of Borradaile et al. [23]. By Lemma 3.2, at the beginning of phase 2, $\text{ex}(f) \leq (k-2)U$. The following lemma shows that phase 2 takes $O(k^5 n \log^3 n \log(kU))$ time:

**Lemma 3.3.** After $O(k\log(kU))$ iterations of the while-loop in phase 2, $\text{ex}(f,x) \leq 2k$ for every vertex $x \in V(G)$.

*Proof.* After each iteration, $\text{ex}(f)$ decreases roughly by a factor $1 + 1/(k-1) \geq 1 + 1/k$. Thus we only require $O(\log_{1+1/k}(kU)) = O(\frac{\ln(kU)}{\ln(1+1/k)})$ iterations. For $k \geq 1$ we have

$$e^{1/2} < (1+1/k)^k < e \tag{3.8}$$

$$\implies 1/2 < k\ln(1+1/k) < 1 \tag{3.9}$$

$$\implies \frac{1}{2k} < \ln(1+1/k) < \frac{1}{k}. \tag{3.10}$$

This means that $O(k \log kU)$ iterations suffice. QED.

In phase 3, the same reasoning as in Section 3.2 shows that computing $\overline{f'} + \overline{f''}$ takes $O(k^2 n)$ time. The total running time of the algorithm is thus

$$O(\log(nU)[n\log^3 n + k^5 n \log^3 n \log(kU) + k^2 n]) = O(k^5 n \log^3 n \log kU \log nU) \tag{3.11}$$

$$= O(k^5 n \text{ polylog}(nU)). \tag{3.12}$$

The rest of this section describes one iteration of the while-loop in phase 2. Specifically, given a feasible flow $f^\circ$ whose restriction $f$ to $G$ has at most $k-2$ infeasible vertices, we compute a flow $f_2^\circ$ in $G^\circ$ whose restriction $f_2$ to $G$ has at most $k-2$ infeasible vertices, each of which has excess at most $\lceil \frac{k-1}{k} ex(f) \rceil$. Let $X$ be the set of infeasible vertices under $f$, and for each $x \in X$, define $ex_x = \text{ex}(f,x)$. The procedure that finds $f_2^\circ$ has three stages. In stage 1, we find a circulation $g^\circ$ such that $f^\circ + g^\circ$ is feasible in $G^\circ$ and $(f^\circ + g^\circ)^{in}(C_x) \leq c(x)$ for every $x \in X$. However, the restriction of $f^\circ + g^\circ$ to $G$ may have large excesses on vertices not in $X$. To fix this, in stage 2 we use $g^\circ$ to compute a feasible flow $f_1^\circ$ in $G^\circ$ satisfying $\text{ex}(f_1^\circ) \leq \lceil \frac{k-1}{k} \text{ex}(f) \rceil$. Intuitively, $f_1^\circ$ approximates $f^\circ + g^\circ/k$ while being an integer circulation. In stage 3, we use Lemma 2.4 on $f_1^\circ$ to get a flow $f_2^\circ$ of the same value whose restriction to $G$ is acyclic and has at most $k-2$ infeasible vertices. If $\lambda > \lambda^*$, then $g^\circ$ may not exist and stage 1 may fail; if $\lambda \leq \lambda^*$, then $g^\circ$ exists and all three stages will work.

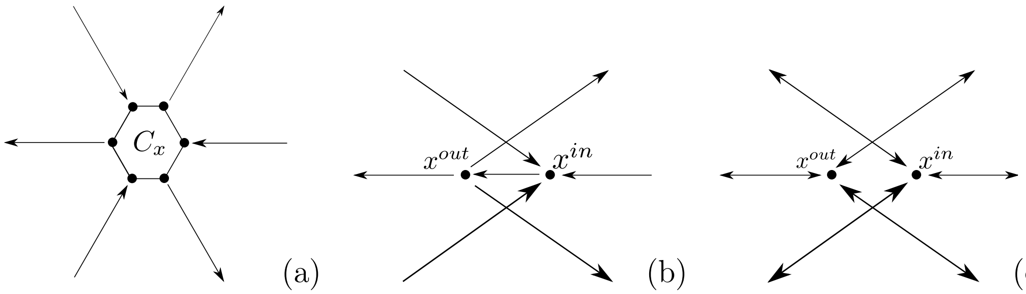


Figure 3.3: (a) $C_x$ in $G^\circ$ if $x \in X$ (b) $x^{in}$ and $x^{out}$ in $G^\times$ (c) source $x^{in}$ and sink $x^{out}$ in $H_i$ if $x = x_i$

### 3.3.1 Stage 1

To get $g^\circ$, we first convert $f^\circ$ to a feasible flow $f^\times$ of the same value in a flow network $G^\times$ such that the restrictions of $f^\circ$ and $f^\times$ to $G$ are equal. Then, we find a circulation $g^\times$ in $G^\times$ such that the restriction of $f^\times + g^\times$ to $G$ has no excesses on the vertices of $X$. Finally, we convert $f^\times + g^\times$ to a feasible flow $f^\circ + g^\circ$ in $G^\circ$, from which we get $g^\circ$.

We construct $G^\times$ as follows. Starting with $G^\circ$, we do the following for each vertex $x \in X$:

- Replace $C_x$ with an arc $(x^{in}, x^{out})$ of capacity $c(x)$.
- Every arc of a capacity $c$ going from a vertex $u$ to a vertex in the cycle $C_x$ is now an arc $(u, x^{in})$ of capacity $c$.
- Every arc of a capacity $c$ going from a vertex in the cycle $C_x$ to a vertex $x$ is now an arc $(x^{out}, u)$ of capacity $c$.

In a slight abuse of terminology, we say that a flow in $G^\circ$ is an extension of a flow in $G^\times$ if the two flows have the same restriction to $G$. Similarly, a flow in $G^\times$ is a restriction of a flow in $G^\circ$ if the two flows have the same restriction to $G$. See Figure 3.3. To define $f^\times$, let $f^\times(u,v) = f^\circ(u,v)$ for all arcs $(u,v) \in E(G^\times) \cap E(G^\circ)$, and let $f^\times(x^{in}, x^{out}) = (f^\circ)^{in}(C_x)$ for all $x \in X$. It is easy to see that $f^\times$ is a flow from $s$ to $t$ whose only infeasible arcs are $(x^{in}, x^{out})$ for all $x \in X$. Furthermore, $(f^\times)^{out}(x^{out}) = (f^\times)^{in}(x^{in}) = f^{in}(x)$ for all $x \in X$. We have the following lemma:

**Lemma 3.4.** For each $x \in X$, let $\omega_x \geq 0$. The following two statements are equivalent:

1. There exists a feasible circulation $g^\circ$ in the residual graph of $G^\circ$ with respect to $f^\circ$ such that

$$(f^\circ + g^\circ)^{in}(C_x) = (f^\circ + g^\circ)^{out}(C_x) = f^{in}(x) - \omega_x \tag{3.13}$$

for all $x \in X$.

2. There exists a circulation $g^\times$ in $G^\times$ such that $f^\times + g^\times$ has a feasible extension in $G^\circ$, $f^\times + g^\times$ is feasible in $G^\times$ except at arcs $(x^{in}, x^{out})$ for all $x \in X$, and

$$(f^\times + g^\times)^{in}(x^{in}) = (f^\times + g^\times)(x^{in}, x^{out}) = (f^\times + g^\times)^{out}(x^{out}) = f^{in}(x) - \omega_x \quad (3.14)$$

for all $x \in X$.

*Proof.* <u>(1) ⇒ (2) :</u> Suppose (1) holds. Let $g^\times$ be the circulation in $G^\times$ defined by $g^\times(e) = g^\circ(e)$ for each $e \in E(G^\circ) \cap E(G^\times)$ and $g^\times(x^{in}, x^{out}) = (g^\circ)^{in}(C_x)$ for all $x \in X$. That is, $g^\times$ is the restriction of $g^\circ$ to $G^\times$. The circulation $g^\times$ satisfies conservation constraints at $x^{out}$ because $(g^\times)^{in}(x^{out}) = g^\times(x^{in}, x^{out}) = (g^\circ)^{in}(C_x) = (g^\circ)^{out}(C_x) = (g^\times)^{out}(x^{out})$; a similar argument shows that $g^\times$ satisfies conservation constraints at $x^{in}$. Also, $g^\times$ satisfies conservation constraints at all other vertices because $g^\circ$ does.

Since $(f^\circ)^{in}(C_x) = (f^\times)^{in}(x^{in})$ and $(g^\circ)^{in}(C_x) = (g^\times)^{in}(x^{in})$, we have $(f^\circ + g^\circ)^{in}(C_x) = (f^\times + g^\times)^{in}(x^{in})$. A symmetric argument shows that $(f^\circ + g^\circ)^{out}(C_x) = (f^\times + g^\times)^{out}(x^{out})$. Flow conservation implies $(f^\times + g^\times)^{in}(x^{in}) = (f^\times + g^\times)(x^{in}, x^{out})$. The flow $f^\times + g^\times$ is feasible at all arcs in $E(G^\times) \cap E(G^\circ)$ because $f^\circ + g^\circ$ is.

<u>(2) ⇒ (1) :</u> Suppose (2) holds. There is a feasible extension $h^\circ$ of $f^\times + g^\times$ to $G^\circ$. Let $g^\circ$ be the circulation in $G^\circ$ such that $f^\circ + g^\circ = h^\circ$. Since $f^\circ + g^\circ$ is feasible in $G^\circ$, $g^\circ$ is feasible in the residual graph of $G^\circ$ with respect to $f^\circ$. It is easy to see that $g^\circ$ is an extension of $g^\times$.

Since $(f^\circ)^{in}(C_x) = (f^\times)^{in}(x^{in})$ and $(g^\circ)^{in}(C_x) = (g^\times)^{in}(x^{in})$, we have $(f^\circ + g^\circ)^{in}(C_x) = (f^\times + g^\times)^{in}(x^{in})$. A symmetric argument shows that $(f^\circ + g^\circ)^{out}(C_x) = (f^\times + g^\times)^{out}(x^{out})$. QED.

If $\lambda \leq \lambda^*$, then there exists a feasible flow $f_\lambda$ in $G$ of value $\lambda$ that can be extended to feasible flows $f_\lambda^\times$ in $G^\times$ and $f_\lambda^\circ$ in $G^\circ$. Thus statement (1) of Lemma 3.4 holds for the circulation $f_\lambda^\circ - f^\circ$ in $G^\circ$ and for some choices of $\omega_x$ where $\omega_x \geq ex_x$ for all $x \in X$. Lemma 3.4 then implies that there exists a circulation $g^\times$ in $G^\times$ such that $(f^\times + g^\times)^{in}(x^{in}) = (f^\times + g^\times)(x^{in}, x^{out}) = (f^\times + g^\times)^{out}(x^{out}) \leq c(x)$ for all $x \in X$, meaning that $f^\times + g^\times$ is feasible in $G^\times$. If $\lambda > \lambda^*$, then $g^\times$ may not exist and its computation may fail. Let $g$ be the restriction of $g^\times$ to $G$.

We will compute the circulation $g^\times$ as the sum of $k-2$ circulations $\phi_1^\times, \ldots, \phi_{k-2}^\times$. Let $x_1, \ldots, x_{k-2}$ be an arbitrary ordering of the vertices in $X$. For all $i \in [k-2]$, let $\gamma_i^\times = \phi_1^\times + \cdots + \phi_i^\times$, and let $\gamma_i$ be the restriction of $\gamma_i^\times$ to $G$. In particular, $\gamma_0^\times$ is the zero flow and $\gamma_{k-2}^\times = g^\circ$. We will find the circulations $\phi_1^\times, \ldots, \phi_{k-2}^\times$ one by one, and we will choose $\phi_i^\times$ to satisfy the following property:

**Lemma 3.5.** For all $i \in [k-2]$, $f^\times + \gamma_i^\times$ is a flow in $G^\times$ that is feasible except at some arcs $(x_j^{in}, x_j^{out})$ where $j > i$. Furthermore, the restriction of $f^\times + \gamma_i^\times$ to $G$ has at most $\mathrm{ex}(f)$ excess on $x_{i+1}, \ldots, x_{k-2}$ and at most $i \cdot \mathrm{ex}(f)$ excess on vertices in $V(G) \setminus X$.

Intuitively, the lemma states that $\phi_i^\times$ gets rid of the excess on $x_i$ without increasing any of the excesses on $x_1, \ldots, x_{i-1}$ above 0, without increasing any of the excesses on $x_{i+1}, \ldots, x_{k-2}$ above $\mathrm{ex}(f)$, and without increasing any other excesses by more than $\mathrm{ex}(f)$.

We now describe how to find $\phi_i^\times$ inductively. Suppose $h^\times = f^\times + \gamma_{i-1}^\times$ is a feasible flow in $G^\times$ whose restriction $h$ to $G$ has no excess on $x_1, \ldots, x_{i-1}$, at most $\mathrm{ex}(f)$ excess on $x_i, \ldots, x_{k-2}$, and at most $(i-1) \cdot \mathrm{ex}(f)$ excess on all vertices in $V(G) \setminus X$. Our goal is to find a circulation $\phi_i^\times$ in $G^\times$ such that $h^\times + \phi_i^\times$ is a feasible flow in $G^\times$ satisfying Lemma 3.5. Finding $\phi_i^\times$ reduces to finding a flow $\phi_{i,H}$ in an $O(k)$-apex graph $H_i$, and we construct $H_i$ as follows: Starting with the residual graph of $G^\times$ with respect to $h^\times$, delete arcs $(x_i^{in}, x_i^{out})$ and $(x_i^{out}, x_i^{in})$. Let the source be $x_i^{in}$ and the sink be $x_i^{out}$. For all $j > i$, the arc $(x_j^{in}, x_j^{out})$ has capacity $c(x_j) + \mathrm{ex}(f) - h^\times(x_j^{in}, x_j^{out})$ and the arc $(x_j^{out}, x_j^{in})$ has capacity $h^\times(x_j^{in}, x_j^{out})$. See Figure 3.3. We have the following lemma:

**Lemma 3.6.** Let $\omega \geq 0$. For all $i \in [k-2]$, the following two statements are equivalent:

1. There exists a circulation $\phi_i^\times$ in $G^\times$ such that $h^\times + \phi_i^\times$ is feasible in $G^\times$ except possibly at arcs $(x_j^{in}, x_j^{out})$ for all $j > i$, where $(h^\times + \phi_i^\times)(x_j^{in}, x_j^{out}) \leq c(x_j) + \mathrm{ex}(f)$. Also,

$$(h^\times + \phi_i^\times)^{in}(x_i^{in}) = (h^\times + \phi_i^\times)(x_i^{in}, x_i^{out}) = (h^\times + \phi_i^\times)^{out}(x_i^{out}) = h^{in}(x_i) - \omega. \quad (3.15)$$

2. There exists a feasible flow $\phi_{i,H}$ in $H_i$ of value $\omega$.

*Proof.* (1) $\Rightarrow$ (2) : Suppose (1) holds. Let $\phi_{i,H}$ be the restriction of $\phi_i^\times$ to $H_i$. The flow $\phi_{i,H}$ is feasible in $H_i$ by the definition of $H_i$.

Since $(h^\times)(x_i^{in}, x_i^{out}) = (h)^{in}(x_i)$ and $(h^\times + \phi_i^\times)(x_i^{in}, x_i^{out}) = h^{in}(x_i) - \omega$, we have that $\phi_i^\times(x_i^{out}, x_i^{in}) = \omega$. Since $x_i^{in}$ is not a source in $G^\times$, flow conservation at $x_i^{in}$ implies that $(\phi_i^\times)^{out}(x^{in}) = \omega$. This means that $|\phi_{i,H}| = \phi_{i,H}^{out}(x_i^{in}) = \omega$.

(2) $\Rightarrow$ (1) : Suppose (2) holds. Define an extension $\phi_i^\times$ of $\phi_{i,H}$ to a circulation in $G^\times$ by setting $\phi_i^\times(x_i^{out}, x_i^{in}) = \omega$. It is easy to see that $g^\times$ satisfies conservation constraints. The arc capacities in $H_i$ ensure $h^\times + \phi_i^\times$ is feasible in $G^\times$ except possibly at arcs $(x_j^{in}, x_j^{out})$ with $j > i$, where $(h^\times + \phi_i^\times)(x_j^{in}, x_j^{out}) \leq c(x_j) + \mathrm{ex}(f)$.

Since $h^\times(x_i^{in}, x_i^{out}) = h^{in}(x_i)$ and $\phi_i^\times(x_i^{out}, x_i^{in}) = \omega$, we have $(h^\times + \phi_i^\times)(x^{in}, x^{out}) = h^{in}(x_i) - \omega$. On the other hand, $x_i^{in}$ and $x_i^{out}$ are not terminals in $G^\times$, so flow conservation implies $(h^\times + \phi_i^\times)^{in}(x_i^{in}) = (h^\times + \phi_i^\times)(x_i^{in}, x_i^{out}) = (h^\times + \phi_i^\times)^{in}(x_i^{out})$. QED.

By the existence of $g^\times$, we know that there exists a circulation $\phi_i^\times$ such that $h^\times + \phi_i^\times$ is feasible in $G^\times$, so statement (1) in Lemma 3.6 holds for some $\omega \geq \mathrm{ex}(h, x_i)$. By Lemma 3.6, there must exist a flow $\phi_{i,H}$ of value $\mathrm{ex}(h, x_i)$ in $H_i$. We compute $\phi_{i,H}$ as follows: Starting with $H_i$, we add a vertex $x^s$ that will be the source instead of $x_i^{in}$, and we add an arc $(x^s, x_i^{in})$ with capacity $\mathrm{ex}(h, x_i)$; similarly, we add a vertex $x^t$ that will be the sink instead of $x_i^{in}$, and an arc $(x_i^{out}, x^t)$ with capacity $\mathrm{ex}(h, x_i)$. The resulting graph has an acyclic maximum flow that saturates every arc incident to a terminal and so has value $\mathrm{ex}(h, x_i)$, and the restriction of this flow to $H_i$ is $\phi_{i,H}$. We have assumed (by induction) that $\mathrm{ex}(h^\times, x_i) \leq \mathrm{ex}(f)$, so $|\phi_{i,H}| \leq \mathrm{ex}(f)$.

We need to show that our choice of $\phi_i^\times$ satisfies Lemma 3.5. By Lemma 3.6, the flow $\phi_{i,H}$ corresponds to a circulation $\phi_i^\times$ in $G^\times$ such that $h^\times + \phi_i^\times$ has no excess on $x_1, \ldots, x_i$ and is feasible in $G^\times$ except possibly at arcs $(x_j^{in}, x_j^{out})$ for all $j > i$, where $(h^\times + \phi_i^\times)(x_j^{in}, x_j^{out}) \leq c(x_j) + \mathrm{ex}(f)$. The restriction of $h^\times + \phi_i^\times$ to $G$ is thus feasible at $x_1, \ldots, x_i$ and has at most $\mathrm{ex}(f)$ excess at $x_{i+1}, \ldots, x_{k-2}$. If $\lambda > \lambda^*$, then $\phi_{i,H}$ may not exist and might have value strictly less than $\mathrm{ex}(h, x_i)$ when we try to compute it. If this happens, then the restriction of $h^\times + \gamma_i^\times$ to $G$ will have positive excess on $x_i$.

Let $\phi_i$ be the restriction of $\phi_i^\times$ to $G$. Let $v$ be any vertex in $V(G) \setminus X$. Since $\phi_{i,H}$ is acyclic, $\phi_i$ sends at most $|\phi_{i,H}|$ units of flow through $v$. Thus for all $v \in V(G) \setminus X$, we have

$$\mathrm{ex}(h + \phi_i, v) \leq \mathrm{ex}(h, v) + |\phi_{i,H}| \tag{3.16}$$
$$\leq (i-1) \cdot \mathrm{ex}(f) + \mathrm{ex}(f) \tag{3.17}$$
$$\leq i \cdot \mathrm{ex}(f). \tag{3.18}$$

This proves Lemma 3.5.

When $i = k-2$, we get that $f^\times + \gamma_i^\times = f^\times + g^\times$ is a feasible flow in $G^\times$ where $\mathrm{ex}(f+g, v) \leq (k-2)\mathrm{ex}(f)$ for all $v \in V(G) \setminus X$. The flow $f^\times + g^\times$ has no excess on the vertices of $X$, so the proof of Lemma 2.1 implies that $f^\times + g^\times$ has an extension in $G^\circ$. Lemma 3.4 then implies that $g^\times$ corresponds to a circulation $g^\circ$ in $G^\circ$ such that $\mathrm{ex}(f^\circ + g^\circ, x) = 0$ for all $x \in X$ and $\mathrm{ex}(f^\circ + g^\circ, v) \leq (k-2)\mathrm{ex}(f)$ for all $v \in V(G) \setminus X$; we can compute $g^\circ$ in $O(n \log^3 n)$ time. We have proved the following lemma:

**Lemma 3.7.** For any vertex $x \in X$, $\mathrm{ex}(f^\circ + g^\circ, x) = 0$. For any vertex $v \in V(G) \setminus X$, $\mathrm{ex}(f^\circ + g^\circ, v) \leq (k-2)\mathrm{ex}(f)$.

Computing $\phi_i^\times$ requires us to compute a maximum flow in a graph with $O(k)$ apices (these are $s$, $t$, and $x^{in}$ and $x^{out}$ for all $x \in X$), which takes $O(k^3 n \log^3 n)$ time using the algorithm of Borradaile et al. [23]. Since we need to compute $k-2$ such flows, computing $g^\circ$ takes

$O(k^4 n \log^3 n)$ time.

### 3.3.2 Stage 2

Having found an integer circulation $g^\circ$ in $G^\circ$, we construct the fractional circulation $g^\circ/k$ in $G^\circ$. Then, using the algorithm of Lemma 2.2, we let $f_1^\circ$ be an integer flow in $G^\circ$ such that

$$(f_1^\circ)^{in}(C_v) \leq \left\lceil (f^\circ + g^\circ/k)^{in}(C_v) \right\rceil \implies \text{ex}(f_1^\circ, v) \leq \lceil \text{ex}(f^\circ + g^\circ/k, v) \rceil . \tag{3.19}$$

for every vertex $v \in V(G)$.

**Lemma 3.8.** For any vertex $v \in V(G)$, $\text{ex}(f_1^\circ, v) \leq \left\lceil \frac{k-1}{k} \text{ex}(f) \right\rceil$.

*Proof.* If $x \in X$, then we have $\text{ex}(f^\circ, x) \leq ex(f)$ and $\text{ex}(f^\circ + g^\circ, x) = 0$ by Lemma 3.7. Thus

$$\text{ex}(f_1^\circ, x) \leq \lceil \text{ex}(f^\circ + g^\circ/k, x) \rceil \leq \left\lceil \frac{k-1}{k} ex(f) \right\rceil . \tag{3.20}$$

If $v \in V(G) \setminus X$, we have $\text{ex}(f^\circ, v) = 0$ and $\text{ex}(f^\circ + g^\circ, v) \leq (k-2)ex(f)$ by Lemma 3.7. Thus

$$\text{ex}(f_1^\circ, v) \leq \lceil \text{ex}(f^\circ + g^\circ/k, v) \rceil \leq \left\lceil \frac{k-2}{k} ex(f) \right\rceil . \tag{3.21}$$

QED.

Using the algorithm of Lemma 2.2, computing $f_1^\circ$ takes $O(n \log^3 n)$ time.

### 3.3.3 Stage 3

In this stage, we finally get $f_2^\circ$. Using Lemma 2.4, we find a flow $f_2^\circ$ of the same value as $f_1^\circ$ such that the restriction $f_2$ of $f_2^\circ$ to $G$ is acyclic. By Lemma 3.1, $f_2^\circ$ has at most $k-2$ infeasible vertices. Since $f_2^\circ(e) \leq f_1^\circ(e)$ for all arcs $e$, we still have $\text{ex}(f_2^\circ, v) \leq \text{ex}(f_1^\circ, v) \leq \left\lceil \frac{k-1}{k} \text{ex}(f) \right\rceil$ for all vertices $v \in V(G)$. Stage 3 takes $O(n)$ time, so the total running time of stages 1-3 is $O(k^4 n \log^3 n)$.

## 3.4 THREE TERMINALS WITH REAL CAPACITIES

In the case of three terminals, we can find a maximum flow in $O(n \log n)$ time even if $G$ has arbitrary real capacities. Without loss of generality, we may assume that there are two sources and one sink. Let $f^\circ$ be a maximum flow in $G^\circ$. We can compute $f^\circ$ in $O(n \log n)$

time by using the algorithm of Borradaile and Klein [41]: first find a maximum flow $f_1^\circ$ from $s_1$ to $t$, and then find a maximum flow $f_2^\circ$ from $s_2$ to $t$ in the residual graph of $G^\circ$ with respect to $f_1^\circ$. The desired flow $f^\circ$ is just $f_1^\circ + f_2^\circ$. By Lemma 2.4 we may assume without loss of generality that the restriction $f$ of $f^\circ$ to $G$ is acyclic. By Lemma 3.1, the flow graph $f_G$ of $f$ has at most one saddle $x$, and has index 1. If $f$ is feasible at $x$, then $f$ is the maximum flow in $G$ and we are done, so assume $f$ is infeasible at $x$.

### 3.4.1 Almost-feasible flows

Let $\delta = \mathrm{val}(G^\circ) - \mathrm{val}(G)$. Suppose $f_\delta^\circ$ is a maximum flow in $G^\circ$ whose restriction $f_\delta$ to $G$ is acyclic and has a single infeasible vertex $x_\delta$. Lemmas 3.1 and 3.2 guarantee that $x_\delta$ has excess at most $c(x_\delta)$, but this excess might still be greater than $\delta$. If it turns out that $\mathrm{ex}(f_\delta, x_\delta) = \delta$, then $f_\delta^\circ$ and $f_\delta$ are *almost feasible*. Given an almost-feasible flow $f_\delta$ in $G$, we can remove $\delta$ units of flow through $x_\delta$ to get a maximum flow in $G$. This can be done in $O(n)$ time using an algorithm similar to that of Step 1 in Section 3.2. In this subsection, we show that almost-feasible flows exist.

**Lemma 3.9.** There exists a maximum flow $f_\delta^\circ$ in $G^\circ$ such that the restriction $f_\delta$ of $f_\delta^\circ$ to $G$ is acyclic and has a single infeasible vertex $x$ with $\mathrm{ex}(f_\delta, x) = \delta$.

*Proof.* Let $f_{max}$ be a maximum flow in $G$, and let $f_{max}^\circ$ be an extension of $f_{max}$ to $G^\circ$. In the residual graph of $G^\circ$ with respect to $f_{max}^\circ$, find an acyclic maximum flow $g^\circ$. Let $(f')^\circ = f_{max}^\circ + g^\circ$ and let $f'$ be the restriction of $(f')^\circ$ to $G$. Since $g^\circ$ is acyclic with a single sink and has value $\delta$, it can be decomposed into arc-disjoint paths whose total flow value is $\delta$. Therefore, for every vertex $v$ in $G$, the flow through $v$ in $f'$ is larger than the flow through $v$ in $f_{max}$ by at most $\delta$. Hence, the excess of every vertex under $f'$ is at most $\delta$. By Lemma 2.4, there is a flow $f_\delta^\circ$ with the same value as $(f')^\circ$ whose restriction $f_\delta$ to $G$ does not contain flow-cycles.

Since $g^\circ$ is a maximum flow in the residual graph of $G^\circ$ with respect to $f^\circ$, $(f')^\circ$ and $f_\delta^\circ$ are maximum flows in $G^\circ$. Since $f_\delta(e) \le f'(e)$ for every arc $e$, we have $\mathrm{ex}(f_\delta) \le \delta$. Since $f_\delta$ is acyclic, Lemma 2.4 implies that $f_\delta$ has at most one infeasible vertex $x$.

If $\mathrm{ex}(f_\delta, x) < \delta$, then, starting with $f_\delta$, we can remove $\mathrm{ex}(f_\delta, x)$ units of flow through $x$ to get a feasible flow in $G$ with value strictly higher than $|f_\delta| - \delta = \mathrm{val}(G)$, a contradiction. Thus $\mathrm{ex}(f_\delta, x) = \delta$. QED.

### 3.4.2 Getting an almost-feasible flow

It remains to show how to compute an almost-feasible flow. We will describe an algorithm that finds a circulation $g^\circ$ in $G^\circ$ such that the restriction of $f^\circ + g^\circ$ to $G$ is almost feasible. Let $ex_x = \mathrm{ex}(f)$.

Construct the flow network $G^\times$ the same way as in Section 3.3.1, and construct $H$ in the same way as $H_i$ is constructed in section 3.3.1 for $i = 1$. In other words, $H$ is constructed as follows. Starting with the residual graph of $G^\circ$ with respect to $f^\circ$, we do the following:

- Replace $C_x$ with vertices $x^{in}$ and $x^{out}$.
- Every arc of capacity $c$ going from a vertex $u$ to a vertex in the cycle $C_x$ is now an arc from $(u, x^{in})$ of capacity $c$
- Every arc of capacity $c$ going from a vertex in the cycle $C_x$ to a vertex $u$ is now an arc $(x^{out}, u)$ of capacity $c$.
- Let $x^{in}$ be the source instead of $s$ and $x^{out}$ be the sink instead of $t$. (Recall from the definition of $G_{st}$ in section 2 that $s$ is a supersource connected to the original two sources $s_1$ and $s_2$.)

Lemmas 3.4 and 3.6 both apply for $H = H_1$. Thus our goal is now to find a maximum flow $g_H$ in $H$ that can be extended to a circulation in the residual graph of $G^\circ$ with respect to $f^\circ$.

We will now show that we can make two simplifications to $H$. The goal of these simplifications is to eliminate the apices $s, x^{in}$, and $x^{out}$ so that $H$ becomes planar. First, since we are ultimately trying to find a flow in $H$ from $x^{in}$ to $x^{out}$, we may assume without loss of generality that arcs of the form $(u, x^{in})$ and $(x^{out}, v)$ do not exist in $H$. As a result, the only arcs in $H$ that are incident to $x^{in}$ are arcs of the form $(x^{in}, u)$ where $f(u, x) > 0$. If we consider these arcs as arcs in $G$, then, since $x$ has index 1, these arcs form two intervals in the cyclic order around $x$. Therefore, we can replace $x^{in}$ with two sources $x_1^{in}$ and $x_2^{in}$, replacing arcs $(x^{in}, u)$ in the first interval with $(x_1^{in}, u)$ and arcs $(x^{in}, u)$ in the second interval with arcs $(x_2^{in}, u)$. A similar simplification eliminates $x^{out}$. See Figure 3.4. (We could not perform this simplification in Section 3.3.1 because the desired flow in $H$ could send flow from $x^{in}$ to some $(x')^{in}$ to $x^{out}$.) One effect of this simplification is that every flow $g_H$ in $H$ automatically extends to a circulation $g^\circ$ in the residual graph of $G^\circ$ with respect to $f^\circ$. This is because $f$ has an extension $f^\circ$ in $G^\circ$ and $(f + g_H)(e) \leq f(e)$ for any arc $e$ incident to $x$ in $G$, so $f + g_H$ has an extension to $G^\circ$.

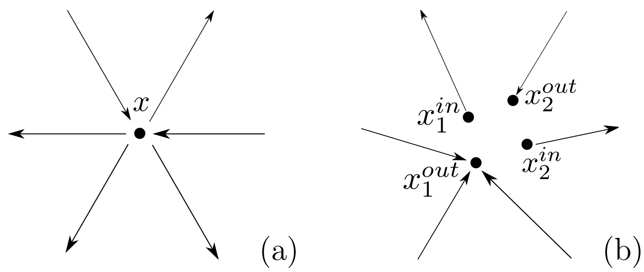


Figure 3.4: (a) The flow graph of $f$ at the unique infeasible vertex (b) sources and sinks of $H$ after eliminating apices

Second, we show that we can delete the arcs $(s, s_1)$ and $(s, s_2)$. This eliminates the apex $s$.

**Lemma 3.10.** If there is an augmenting path $\pi$ in $H$ (i.e., a path from a source to a sink in $H$) containing $s$, then there is an augmenting path $\pi'$ in $H$ not containing $s$.

*Proof.* See Figure 3.5. In this proof, we say that an arc $e$ carries flow if $f(e) > 0$, and a path carries flow if all of its arcs carry flow. Consider two arcs $e, e'$ carrying flow out of $x$ such that as we cyclically traverse the arcs incident to $x$ in clockwise order, some arc between $e$ and $e'$ carries flow into $x$, and some arc between $e'$ and $e$ carries flow into $x$. There must be a path $P$ from $x$ to $t$ starting with $e$ that carries flow. Similarly, there must be a path $P'$ from $x$ to $t$ starting with $e'$ that carries flow. Without loss of generality, assume $P$ and $P'$ do not cross. Let $u$ be the first vertex on $P$ after $x$ that also appears on $P'$. The vertex $u$ must also be the first vertex on $P'$ after $x$ that also appears on $P$, because otherwise $f$ has flow-cycles. Let $Q$ be the prefix of $P$ that ends at the arc of $P$ that goes into $u$, and let $Q'$ be the prefix of $P'$ that ends at the arc of $P'$ that goes into $u$. These prefixes are well-defined because $f$ is acyclic. Since both $Q$ and $Q'$ go from $x$ to $u$, their union partitions the plane into two regions. Denote the inner region by $R$ and the outer region by $R'$.

Since there are arcs in both $R$ and $R'$ carrying flow into $x$ and $f$ is acyclic, one source must be in $R$ and the other must be in $R'$. Furthermore, there is some path $Q_s$ from $s_2$ to $x$ carrying flow, and there is some path from $s_1$ to $x$ carrying flow.

Without loss of generality, suppose the augmenting path $\pi$ in $H$ starts in $x^{in}$, goes to $s_1 \in R$, uses arcs $(s_1, s)$ and $(s, s_2)$, and ends by going from $s_2 \in R'$ to $x^{out}$. We can replace it by an augmenting path $\pi'$ that starts at $x^{in}$, follows $\text{rev}(Q_s)$ to $s_2$, and then follows $\pi$ from $s_2$ to $x^{out}$. The augmenting path $\pi'$ does not contain $s$.

QED.

Let $g_H$ be the maximum flow in $H$ and let $g^\circ$ be its extension to $G^\circ$. We apply Lemma 2.4 to find a flow $f_3^\circ$ with the same value as $f^\circ + g^\circ$ whose restriction $f_3$ has no flow-cycles.

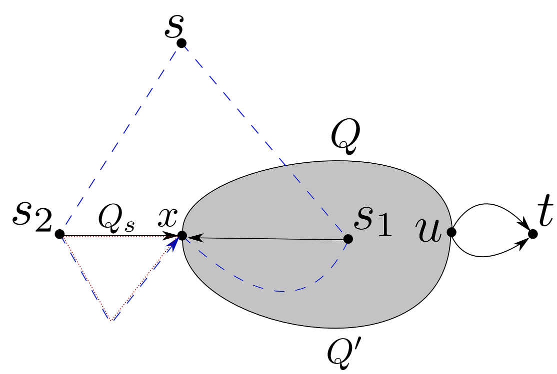


Figure 3.5: $H$ in the proof of Lemma 3.10 with all terminals merged into the vertex $x$. The dashed blue path is $\pi$. The dotted red path is $\pi'$. Note that $\pi$ and $\pi'$ overlap. The shaded region is $R$.

By Lemma 3.1, the flow $f_3^\circ$ is infeasible at a single vertex $y$. If $y = x$, then $f_3^\circ$ must be almost feasible. This is because Lemma 3.4 implies that if $g_H$ is a maximum flow in $H$, then $f^\circ + g^\circ$ is a maximum flow in $G^\circ$ that minimizes the excess of $x$. Furthermore, $f_3(e) \leq (f^\circ + g^\circ)(e)$, so $f_3$ is a maximum flow in $G^\circ$ that minimizes the excess of $x$.

If $y \neq x$, then we define a function $F : E(G) \times [0, 1] \to \mathbb{R}$ for each arc $e \in G$. $F(e, \beta)$ is defined as follows. We apply Lemma 2.4 to $f^\circ + \beta g^\circ$ to get a flow $f_\beta^\circ$ whose restriction $f_\beta$ to $G$ is acyclic. We then define $F(e, \beta) = f_\beta(e)$. For all arcs $e \in E(G)$, we have $F(e, 0) = f(e)$ and $F(e, 1) = f_3(e)$.

Clearly, $F(\cdot, \beta)$ has an extension that is feasible in $G^\circ$ for all $\beta$, and $F(e, \cdot)$ is continuous for any arc $e \in E(G)$. Consider how $F(\cdot, \beta)$ changes as $\beta$ increases from 0 to 1. We start with excess on $x$ and no other vertices, and end with excess on $y$ but no other vertices. Moreover, no matter what $\beta$ is, there is at most one infeasible vertex. Thus, at some point, say when $\beta = \beta_0$, we must have no infeasible vertices. Since $F(\cdot, \beta_0)$ is a maximum flow in $G^\circ$, it must be a maximum flow in $G$.

To compute $\beta_0$, we need the following lemma.

**Lemma 3.11.** For every fixed arc $e \in E(G)$, $\frac{\partial F(e,\beta)}{\partial \beta}$ is constant.

*Proof.* The proof requires understanding the details of the algorithm of Lemma 2.4, which can be found in Section 2.2.2. Here we summarize how the flow $F(\cdot, \beta)$ is computed:

1. Compute $f_\beta^\circ = f^\circ + \beta g^\circ$. Define a capacity function $c'$ by $c'(e) = f_\beta^\circ(e)$ for all $e \in E(G)$ and $c'(e) = c(e)$ for all $e \notin E(G)$. Construct the residual graph $G_\beta^\circ$ of $G^\circ$ with respect to $f_\beta^\circ$ and $c'$. Let $h_\infty$ be the infinite face of $G^\circ \setminus \{s\}$. For each face $h$ of $G_\beta^\circ \setminus \{s\}$, let $\Phi(h)$ be the distance of $h^*$ from $h_\infty^*$ in $(G_\beta^\circ \setminus \{s\})^*$. For each arc $e$ in $G^\circ \setminus \{s\}$, let $h_\ell$ be the face on the left of $e$ and let $h_\ell$ be the face on the right. Let $g_\beta(e) = \Phi(h_r) - \Phi(h_\ell)$

for each arc $e$ in $G^\circ \setminus \{s\}$; $g_\beta$ is a simple circulation. Finally, let $f^\circ_\gamma = f^\circ_\beta + g_\beta$, and let $f_\gamma$ be the restriction of $f^\circ_\gamma$ to $G$. The flow $f_\gamma$ has no counter-clockwise flow-cycles.

2. Define a new capacity function $c''(e) = f^\circ_\gamma(e)$ for $e \in E(G)$ and $c''(e) = c(e)$ for $e \notin E(G)$. Construct the residual graph $G^\circ_\gamma$ of $G^\circ$ with respect to $c''$ and $f^\circ_\gamma$. For each face $h$ of $G^\circ_\gamma \setminus \{s\}$, let $\Phi(h)$ be he distance of $h^*$ from $h^*_\infty$ in $(G^\circ_\gamma \setminus \{s\})^*$. Let $g_\gamma(e) = \Phi(h_\ell) - \Phi(h_r)$. Finally, $F(\cdot, \beta)$ is the restriction of $f^\circ_\gamma + g_\gamma$ to $G$.

It suffices to show that the shortest path trees $T_\beta$ in $(G^\circ_\beta \setminus \{s\})^*$ and $T_\gamma$ in $(G^\circ_\gamma \setminus \{s\})^*$ rooted at $h^*_\infty$ do not change as $\beta$ increases. Suppose for the sake of argument that $T_\beta$ changes as $\beta$ increases. Then, there exist vertices $u^*$ and $v^*$ in $(G^\circ_\beta \setminus \{s\})^*$ and two internally disjoint paths $P^*_1$ and $P^*_2$ from $u^*$ to $v^*$ in $(G^\circ_\alpha \setminus \{s\})^*$ whose lengths are changing at different rates as $\beta$ increases. Let $H$ be the region bounded by $P^*_1$ and $P^*_2$, and suppose that $P_1 \circ \mathrm{rev}(P_2)$ is a clockwise cycle. The change in the length of $P^*_1$ in $(G^\circ_\beta \setminus \{s\})^*$ is the change in the capacity of the cut $P_1$ in $G^\circ_\beta \setminus \{s\}$, which is the change in the amount of flow $f^\circ + \beta g^\circ$ sends out of $H$ through the arcs of $P_1$. Similarly, the change in the length of $P^*_2$ is the change in the amount of flow $f^\circ + \beta g^\circ$ sends into $H$ through the arcs of $P_2$. This means that the net amount of flow that $f^\circ + \beta g^\circ$ carries into $H$ is changing as $\beta$ increases, but this is impossible, since $g^\circ$ is a simple circulation. We conclude that $T_\beta$ does not increase as $\beta$ increases. A similar argument shows that since $g_\beta$ is a simple circulation and $f^\circ_\gamma = f^\circ + \beta g^\circ + g_\beta$, $T_\gamma$ does not change as $\beta$ increases.

QED.

Lemma 3.11 implies that $\frac{d}{d\beta} ex(F(\cdot, \beta), x)$ is constant, and we can find it because

$$\frac{d}{d\beta} ex(F(\cdot, \beta), x) = \mathrm{ex}(F(\cdot, 1), x) - \mathrm{ex}(F(\cdot, 0), x) \tag{3.22}$$

$$= \mathrm{ex}(f_3, x) - \mathrm{ex}(f, x), \tag{3.23}$$

We then let

$$\beta_0 = -\frac{\mathrm{ex}(F(\cdot, 0), x)}{\frac{d}{d\beta}\mathrm{ex}(F(\cdot, \beta), x)}. \tag{3.24}$$

and $F(\cdot, \beta_0)$ is a maximum flow in $G$.

The algorithm takes $O(n \log n)$ time to compute $f^\circ$. It takes $O(n \log n)$ time to compute $g_H$, from which we can obtain $g^\circ$, $f^\circ_3$, and $f_3$ in linear time. If $y = x$, then we have an almost-feasible flow that can be turned into a maximum flow in $G$ in linear time. If $y \neq x$, then we can compute $\beta_0$ and $F(\cdot, \beta_0)$ in linear time. The entire algorithm takes $O(n \log n)$ time.

## 3.5 OPEN PROBLEMS

When it comes to finding flow in planar graphs with vertex capacities, three main open problems remain: (1) eliminate the dependence on $k$ in the running times of the algorithms of Sections 3.2 and 3.3; (2) generalize our algorithms to graphs with real capacities and more than three terminals; and (3) generalize any of our algorithms to surface-embedded graphs. Unfortunately, the first two open problems seem difficult. For the first problem, all of our algorithms relies on Lemma 3.1, so designing an algorithm whose running time does not depend on the number of terminals will probably require completely new techniques. For the second problem, we can either try to generalize the algorithm of Section 3.3 to handle real capacities or try to generalize the algorithm of Section 3.4 to handle more than three terminals. The difficulty with generalizing the algorithm of Section 3.3 is that this algorithm is fundamentally a scaling algorithm, which can only handle integer capacities. If we apply the algorithm to a graph with real capacities, then the flow could keep approaching the maximum flow without ever getting to it. The difficulty with generalizing the algorithm of Section 3.4 to the case where $k > 3$ is as follows. Suppose that $k = 4$. We can define a maximum flow in $G^\circ$ as being almost feasible if we can remove $\delta$ units of flow to get a feasible flow in $G$, where again $\delta = \text{val}(G^\circ) - \text{val}(G)$. We can prove that almost-feasible flows always exist. The main problem seems to be that there is no easy way of characterizing or getting an almost-feasible flow. For example, minimizing the sum of the excesses of the two infeasible vertices $x$ and $x'$ does not necessarily work. Suppose there is one flow where the infeasible vertices both have excesses of 10, and another flow where the vertices both have excesses of 7. If $\delta = 10$, then it could be the case that the first flow is almost feasible because removing a unit of flow through $x$ may simultaneously remove a unit of flow through $x'$ (i.e., we can decompose the first flow into paths and cycles such that some paths pass through both $x$ and $x'$), while the second flow is not almost feasible because removing a unit of flow through $x$ does not simultaneously remove a unit of flow through $x'$, and vice versa.

The third open problem – generalizing an algorithm to surface graphs – may be easier. Consider the algorithm of Section 3.3. This algorithm exploits planarity in four ways. First, it relies on Lemma 3.1, which bounds the number of saddles, uses the fact that the graph is planar. However, the lemma can be easily generalized to surface graphs as follows:

**Lemma 3.12.** Any directed acyclic graph embedded on an orientable surface of genus $g$ with $k_1$ sources and $k_2$ sinks has at most $k_1 + k_2 - 2 + 2g$ saddles.

Second, our algorithm needs to compute "lifts" of feasible flows in $G$ into a supergraph $G^\circ$ in near-linear time. If $G$ is planar, this reduces to computing a flow in a planar graph that is just the union of disjoint cycles. If $G$ has genus $g$, then we need to compute flows in a surface-embedded graph that is just the union of cycles. Clearly such a graph is still planar, so no changes in the algorithm are needed.

Third, our algorithm uses an algorithm by Borradaile et al. [23] that computes flows in $O(k)$-apex graphs in $O(k^3 n \log^3 n)$ time. (An $O(k)$-apex graph is a graph that can become a planar graph with the removal of $O(k)$ vertices.) If $G$ is a graph of genus $g$, then instead we need to compute integer flows in graphs that become *surface* graphs with the removal of $O(k)$ vertices. Currently, no such algorithm is known. However, we may be able to construct one using the same strategy that Borradaile et al. use. Essentially, they Hochstein and Weihe's algorithm [43] to their own $O(n \log^3 n)$-time algorithm for multiple-source multiple-sink maximum flow in planar graphs. The result is an algorithm for $O(k)$-apex graphs with multiple sources and sinks that runs in $O(k^3 n \log^3 n)$ time. In other words, adding $k$ vertices arbitrarily to the graph increases the running time by a factor $O(k^3)$. Now in our case, we have algorithms that can compute integer flows in surface graphs with a single source and sink in $O(g^8 n \text{ polylog}(nU))$ time [42]. We can generalize this to multiple sources and sinks in $O(k^2 g^8 n \text{ polylog}(nU))$ time by computing flows from each source to each sink while updating the residual graph [44]. Thus, it seems we can use Hochstein and Weihe's algorithm to get an algorithm that finds integer flows in graphs that become a surface graphs with the removal of $O(k)$ vertices in $O(k^5 g^8 n \text{ polylog}(nU))$ time.

Fourth, our algorithm uses a subroutine by Kaplan and Nussbaum that cancels cycles in planar flows in a particular way in $O(n)$ time. Roughly speaking, the subroutine first cancels clockwise cycles, and then cancels counterclockwise cycles. However, in surface graphs, cycles can be neither clockwise nor counterclockwise. Thus, generalizing the subroutine of Kaplan and Nussbaum appears to be the main technical barrier to generalizing our algorithm to surface graphs. Specifically, given a surface graph and a homotopy or homology class, we would like to be able to find a circulation in the graph such that the resulting residual graph contains no cycles in the class, in near-linear time. Even for unit-capacity flow networks embedded in the torus, no polynomial-time algorithms are known. A related problem is the following: given a maximum flow in a surface graph, compute an acyclic maximum flow in the surface graph. Currently, the fastest algorithm known takes $O(m \log n)$ time for general graphs and thus $O((n + g) \log n)$ time for surface graphs [29], but Kaplan and Nussbaum showed that $O(n)$ time is achievable in planar graphs [22]. Are there faster algorithms that exploit the topology of surface graph? More generally, are there algorithms that can compute acyclic flows in near-planar graphs faster than $O(m \log n)$ time?

# CHAPTER 4: ELEMENT CONNECTIVITY

In this chapter, $G$ is an undirected, unweighted graph, $T \subset V(G)$ is a set of terminals, and $k = |T|$. In the preliminaries, we defined the element-connectivity $\kappa'(u, v)$ between two terminals $u$ and $v$. Element-connectivity was first defined by Jain et al. [1].

Chekuri, Rukkanchanunt, and Xu [3] proved that element-connectivity admits a structure called a *Gomory-Hu tree*. This is a tree $\tau$ whose vertices are the terminals of $G$; furthermore, for any two terminals $u$ and $v$, $\kappa'(u, v)$ is the weight of the lightest edge on the path between $u$ and $v$ in $\tau$. The existence of this tree implies that there are only $k - 1$ distinct element-connectivity values. They used this to show that the element-connectivity between every pair of terminals can be computed in $O(k\ \mathrm{MF}(n, m))$ time, where $\mathrm{MF}(n, m)$ is the time required to compute a maximum flow in a unit-capacitated graph. In general graphs, it is known that $\mathrm{MF}(n, m) = O(\sqrt{n}m)$ [48], while in planar graphs we have $\mathrm{MF}(n, m) = O(n)$ [39] and in surface graphs of genus $g$ we have $\mathrm{MF}(n, m) = O(g^8 n \log^4 n)$ [42]. Thus we can compute all element-connectivity values in $O(kn)$ time in planar graphs and $O(kg^8 n \log^4 n)$ time in surface graphs. Alternatively, Borradaile et al. [49] showed that Gomory-Hu trees in surface graphs can be computed in $2^{O(g^2)} n \log^3 n$ time. The algorithm can be easily modified to find the element-connectivity between every pair of terminals in surface graphs in $2^{O(g^2)} n \log^3 n$ time. Chekuri et al. [3] also gave an algorithm that will compute the reduced graph in $O(kmn)$ time. Applying that algorithm to planar graphs gives us a running time of $O(kn^2)$.

In this chapter we describe some minor optimizations to the results mentioned in the previous paragraph. First, we show that the global element connectivity of a planar graph can be computed in $O(bn)$ time when the terminals can be covered by $b$ faces. This is an improvement over previous results when $b \in o(k)$. Second, we show that the reduced graph of a planar graph can be found in $O(kn^{5/3} \log^{4/3} n)$ time.

## 4.1 GLOBAL ELEMENT CONNECTIVITY

Suppose $G$ is planar and all the terminals can be covered by $b$ faces. We show how to find the global element-connectivity in $O(bn)$ time. As mentioned earlier, global element-connectivity in $G$ reduces to global edge-connectivity in $G^\circ$, so we will assume that we are finding global edge-connectivity, which is just the capacity of the minimum cut in $G^\circ$ that separates two terminals.

First we consider $b = 1$; we may assume that all terminals are on the outer face $F$. Let $(G^\circ)*$ denote the dual of $G^\circ$; each terminal in $G^\circ$ becomes a terminal face in $(G^\circ)^*$, and face

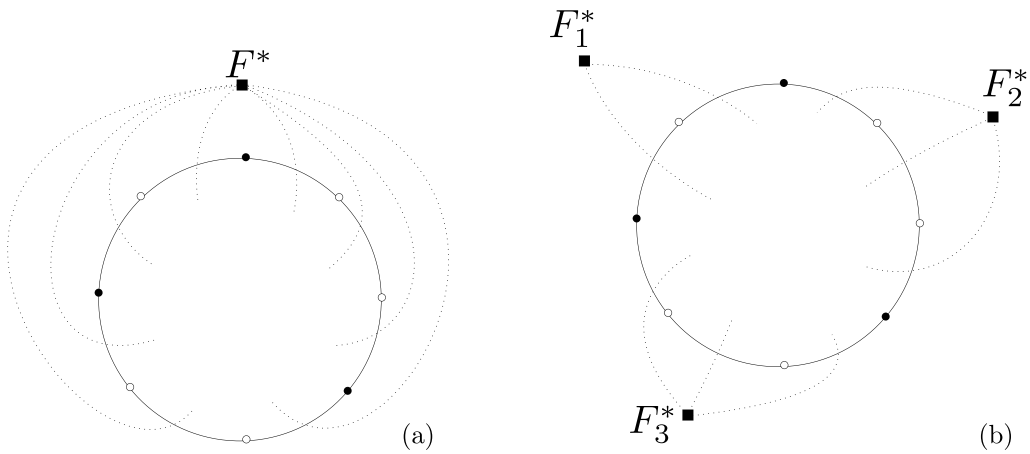


Figure 4.1: Solid edges and circular vertices are primal; dotted edges and square vertices are dual. Circles with black interiors are terminals.

$F$ becomes a dual vertex $F^*$. We are looking for the minimum cut in $G^\circ$ that separates two terminals; this is equivalent to the shortest cycle $C$ in $(G^\circ)^*$ that separates two terminal faces. Clearly $C$ must go through $F^*$. Split $F^*$ into $k$ vertices $F_1^*, \ldots, F_k^*$, as shown in Figure 4.1. The global edge-connectivity is then just the shortest distance between $F_i^*$ and $F_j^*$, where $i \neq j$ and $i$ and $j$ range over $\{1, \ldots, k\}$. We can solve this problem in $O(n)$ time by simultaneously growing $k$ breadth-first-search trees rooted at $F_1^*, \ldots, F_k^*$, and stopping when two of these trees meet. Alternatively, Borradaile [50] showed how to find this shortest distance in $O(n)$ time even when the graph is weighted.

Now suppose $b > 1$. The minimum cut that separates two terminals either separates two terminals on the same face or it separates two of the $b$ faces that cover the terminals. We can take care of the first case in $O(bn)$ time using the algorithm for the $b = 1$ case $b$ times. To take care of the second case, go through each of the $b$ faces that cover the terminals and for each face arbitrarily pick one of the terminals on the face as a representative. While there exist two unseparated representative terminals, find the minimum cut separating them that does not cross any previously-computed minimum cut. (Two cuts cross if their dual cycles cross). Computing a single such cut can be done in $O(n)$ time using the algorithm of Eisenstat and Klein [39]. We can compute $b - 1$ cuts in this way before all representatives are separated. Thus this second case takes $O(bn)$ total time. We conclude:

**Theorem 4.1.** In planar graphs where the terminals can be covered by $b$ faces, global element-connectivity can be found in $O(bn)$ time.

## 4.2 THE REDUCED GRAPH

In Section 2.2.3 we defined the reduced graph of $G$. The naive algorithm for computing the reduced graph when the input graph is planar (or of bounded genus) proceeds as follows. First, we compute the Gomory-Hu tree for element-connectivity in $O(n \log^3 n)$ time [49]. Then, we iterate over the edges of $G$. For each edge $e$, we either contract or delete $e$, whichever action preserves element-connectivity values between the endpoints of each of the $k-1$ edges of the Gomory-Hu tree. Checking whether or not a single element-connectivity value is preserved reduces to recomputing a maximum flow in a graph with vertex capacities when an edge is deleted or contracted; this takes $O(n)$ time. Thus the algorithm takes $O(kn^2)$ time.

We can improve this naive algorithm in planar graphs. First, recall that maximum flow values in $G$ are equal to maximum flow values in $G^\circ$, which has no vertex capacities. To improve the naive algorithm, we use a result of Italiano et al. [40], who described a dynamic maximum-flow algorithm in undirected planar graphs (without vertex capacities) that is able to insert edges, delete edges and answer maximum-flow value queries between any pair of vertices. Their algorithm computes an $r$-division and runs in

$$O\left(\frac{T_1}{r} + T_2 + r + \frac{n}{\sqrt{r}} \log^2 n\right) \tag{4.1}$$

time per operation, where $T_1$ is the time required to compute an $r$-division, and $T_2$ is the time required to compute a dense distance graph on $r$ vertices. We have $T_1 = O(n)$ by results of Goodrich [51] and of Klein, Mozes, and Sommer [52], and in unit-capacitated graphs we have $T_2 = O(r)$ by an algorithm of Eisenstat and Klein [39]. Setting $r = n^{2/3} \log^{4/3} n$, we see that dynamic maximum-flow can be solved in $G^\circ$ in $O(n^{2/3} \log^{4/3} n)$ time per operation. Furthermore, a single contraction or deletion of an edge in $G$ can be simulated by a constant number of insertions and deletions of edges in $G^\circ$. We conclude

**Theorem 4.2.** We can compute the reduced graph in $O(kn^{5/3} \log^{4/3} n)$ time if $G$ is planar.

## 4.3 OPEN PROBLEMS

Several open problems exist. First, we showed that if all the terminals are on a single face, then the global (i.e., smallest) element connectivity can be computed in $O(n)$ time. How fast can we compute all $k-1$ element connectivity values? $O(kn)$-time algorithms are known, but it would be nice to get an $O(n+k)$-time algorithm.

Next, there is the question of computing a reduced graph faster. One idea might be to speed up the algorithm for computing dynamic maximum flows, since we are not adding any edges to $G$ and we are only interested in the case of unweighted graphs. Thus, every time we delete or contract an edge in $G$, we know that each maximum flow value either remains the same or decreases by exactly 1. On the other hand, even for unweighted graphs, there may be some lower bounds to keep in mind. Specifically, Abboud and Dahlgaard [53] showed the following theorem.

**Theorem 4.3.** No algorithm can solve the dynamic shortest path problem in unit weight planar graphs on $N$ nodes with amortized query time $O(N^{1/3-\epsilon})$ and update time $O(N^{1/3-\epsilon})$ for any $\epsilon > 0$ unless the $OMv$ conjecture is false. (In the online boolean matrix-vector multiplication problem, we are given an $n \times n$ matrix $M$ and $n$ column-vectors $v_1, \ldots, v_n$ of size $n$, one by one. We need to compute $Mv_i$ before we are allowed to see $v_{i+1}$. The OMv conjecture says that no algorithm can solve this problem in $O(n^{3-\epsilon})$ time.)

We can modify this reduction to apply to dynamic maximum flow in unit-weight planar graphs. The idea is that a shortest path is a shortest cycle if the endpoints of the path are the same, and the shortest cycle in a planar graph corresponds to a minimum cut or maximum flow in its dual.

Another potential idea for computing the reduced graph is the following. Recall from Theorem 2.1 that each edge connecting two non-terminals can be either contracted or deleted without affecting the element-connectivity between any two terminals. Say an edge between two non-terminals is *contractible* if it can be contracted without affecting any element-connectivity values, and say the edge is *deletable* if it can be deleted without affecting any such values; some edges can be both contractible and deletable. We can view the element-connectivity between two vertices $u$ and $v$ as the capacity of the minimum cut separating $u$ and $v$, where a cut can contain edges and non-terminals. Note that an edge in any minimum cut between two terminals must be contracted, because deleting the edge would decrease the capacity of the cut. Conversely, any edge that is not in any minimum cut can be deleted; since the graph is unweighted, their deletion would not decrease any minimum cut values between terminals. Now it would seem that all we need to do is to compute all minimum cuts between the terminals by computing the Gomory-Hu tree, and then every edge between two non-terminals can be classified as either contractible or deletable. We can contract all contractible edges at once because after contracting a single edge, all other contractible edges remain contractible. The main issue with this idea is that we cannot delete all deletable edges at once: after deleting an edge, some edges that were deletable may no longer be deletable, since some cuts that were not minimum before become minimum.

# CHAPTER 5: SHORTEST DISJOINT PATHS

The vertex-disjoint paths problem is a special case of multicommodity flows. This problem is NP-hard [32], even if $G$ is undirected planar [55] or if $G$ is directed and $k = 2$ [21]. On the other hand, it can be solved in polynomial time if $G$ is undirected and $k$ is bounded [56, 57] or if $G$ is directed acyclic and $k$ is bounded [21]. Furthermore, the problem is fixed-parameter tractable with respect to the parameter $k$ in directed planar graphs [58, 59]. Other related results can be found in the survey by Naves and Sebő [60].

The $k$-min-sum problem has been previously considered in the context of network routing, where the goal is to minimize the amount of energy required to send packets [17, 18]. Middendorf and Pfeiffer [55] proved that the $k$-min-sum problem is NP-hard when the parameter $k$ is part of the input, even in undirected 3-regular plane graphs. However, surprisingly little is known about the complexity of the planar $k$-min-sum when $k$ is fixed. In fact, no non-trivial algorithms or hardness results are known for either the 2-min-sum problem in directed planar graphs or the 5-min-sum problem in undirected planar graphs, even when all terminals are required to lie on a single face.

Polynomial-time algorithms for the planar $k$-min-sum problem are known for *arbitrary* $k$ when all $2k$ terminals lie on a single face, in one of two patterns. In a *parallel* instance, the terminals appear in cyclic order $s_1, \ldots, s_k, t_k, \ldots, t_1$, and an a *serial* instance, the terminals appear in cyclic order $s_1, t_1, s_2, t_2, \ldots, s_k, t_k$. Even in directed planar graphs, parallel instances of $k$-min-sum can be solved using a straightforward reduction to minimum-cost flows [61] in $O(kn)$ time. A recent algorithm of Borradaile, Nayyeri, and Zafarani [62] solves any serial instance of $k$-min-sum in an undirected planar graph in $O(kn^5)$ time.

If we allow arbitrary patterns of terminals, fast algorithms are known for only very small values of $k$. Kobayashi and Sommer [9] describe two algorithms, one running in $O(n^3 \log n)$ time when $k = 2$ and all four terminals are covered by at most two faces, the other running in $O(n^4 \log n)$ time when $k = 3$ when all terminals are incident to a single face. Colin de Verdière and Schrijver [13] describe an $O(kn \log n)$-time algorithm for directed planar graphs where all sources $s_i$ lie on one face and all targets $t_i$ lie on another face. Finally, if $k \leq 3$, every planar instance of $k$-min-sum with all terminals on the same face is either serial or parallel.

Zafarani [63] proved an important structural result for the planar $k$-min-sum problem. Consider an undirected edge-weighted plane graph $G$ with terminals $s_1, t_1, \ldots, s_k, t_k$ on its outer face, and suppose $s_k$ and $t_k$ are adjacent in cyclic order of the terminals. (The other

This chapter is based on joint work with Prof. Jeff Erickson [54].

$2k-2$ terminals can appear in any order.) Let $Q_1, Q_2, \ldots, Q_k$ be the shortest vertex-disjoint paths in $G$ connecting all $k$ terminal pairs, and let $P_1, P_2, \ldots, P_{k-1}$ be the shortest vertex-disjoint paths in $G$ connecting every pair except $s_k, t_k$, where the subscript on each path indicates which terminals it connects. Zafarani's Structure Theorem states that if two paths $P_i$ and $Q_j$ cross, then $i = j$.

Finally, Datta *et al.* [64] recently proved that the $k$-min-sum problem in *unweighted* plane graphs, with all terminals on the outer face, can be solved in polynomial time for arbitrary fixed $k$ and arbitrary terminal patterns. Specifically, they described a randomized algorithm that runs in $O(4^k n^{\omega+1})$ expected time, and a deterministic algorithm that runs in $O(4^k n^{\omega})$ time where $O(n^{\omega})$ is the time for fast matrix multiplication. Their algorithms rely on subtle inclusion-exclusion techniques that appear difficult to generalize to weighted graphs.

Both the $k$-min-min and $k$-min-max problems appear to be harder than the $k$-min-sum problem. Van der Holst and de Pina [61] proved that $k$-min-max is strongly NP-hard when $k$ is not fixed, when all terminals lie on the outer face. Yang, Zheng, and Lu [65] proved that the problem is NP-hard when $k = 2$ and all terminals can be covered by two faces, and Yang, Zheng, and Katukam [66] showed that vertex-disjoint $k$-min-min is NP-hard in general graphs when $k = 2$ and both paths share the same pair of endpoints.

In this chapter, we describe three results. First, we describe the first polynomial-time algorithm to solve the 4-min-sum problem in undirected edge-weighted planar graphs with all eight terminals on a common face. If the given instance is parallel or serial, it can be solved using existing algorithms; otherwise, the terminals can be labeled $s_4, s_3, s_1, t_1, s_2, t_2, t_3, t_4$ in cyclic order around their common face. To solve these instances, our algorithm first computes a solution to the 3-min-sum problem for the terminal pairs $s_1t_1$, $s_2t_2$, $s_4t_4$, using an existing algorithm [9, 62]. We identify a small set of key *anchor* vertices where the 3-min-sum solution intersects the 4-min-sum solution we want to compute. For each possible choice of anchor vertices, our algorithm connects these vertices to the terminals by solving parallel min-sum problems in three carefully constructed subgraphs of $G$. Overall, our algorithm runs in $O(n^6)$ time.

Second, we sketch a method of extending our 4-min-sum algorithm to larger values of $k$ when the terminals appear in order $s_1, t_1, s_2, t_2, t_3, \ldots, t_k, s_k, \ldots, s_1$ along the outer face. Our extended algorithm runs in polynomial time for any fixed $k$.

Third, we describe a $k$-approximation for the $k$-min-sum vertex-disjoint paths when all $k$ terminal pairs are on the outer face of a planar graph. In this algorithm, we construct an integer program for the problem, solve a linear program relaxation of this integer program, and then round the resulting fractional solution.

Our algorithms search for pairwise vertex-disjoint *walks* with minimum total length that connect corresponding terminals, rather than explicitly seeking simple paths. Because all edge lengths are non-negative, the shortest set of walks will of course consist of simple paths.

This chapter is organized as follows. Sections 5.1- 5.4 deal with our 4-min-sum algorithm. In Section 5.1 we describe the algorithm for solving parallel instances of the $k$-min-sum problem. This algorithm is not new but is included for completeness. In Sections 5.2 and 5.3 we prove several structural properties that will be used in our 4-min-sum algorithm. In Section 5.4 we describe the 4-min-sum algorithm. In Section 5.5 we describe our slight extension of the 4-min-sum algorithm. In section 5.6 we describe the $k$-approximation algorithm; this section does not rely on any of the previous sections in this chapter.

## 5.1 ALGORITHM FOR PARALLEL INSTANCES

Our 4-min-sum algorithm relies on a black-box subroutine to solve parallel instances of 2-min-sum and 3-min-sum. Van der Holst and de Pina [61] observed that any parallel instance of $k$-min-sum can be solved in polynomial time by reduction to minimum-cost flow problem. In fact, these instances can reduced in $O(n)$ time to a *planar* instance of minimum-cost flow, by replacing each vertex with a clockwise directed unit-capacity cycle, as described by Colin de Verdiére and Schrijver [13] and Kaplan and Nussbaum [67]. The resulting minimum-cost flow problem can then be solved $O(kn)$ time by performing $k$ iterations of the classical successive shortest path algorithm [68, 69, 70], using the $O(n)$-time shortest-path algorithm of Henzinger *et al.* [31] at each iteration.

**Lemma 5.1.** Any parallel planar instance of the $k$-min-sum problem can be solved in $O(kn)$ time.

*Proof.* Let $G$ denote the input plane graph, without loss of generality embedded so that all $2k$ terminals lie on the outer face $\partial G$. We assume $G$ is directed; otherwise, replace every undirected edge with two oppositely oriented directed edges with equal length.

First, we convert the input graph $G$ into a planar flow network $G'$ *with vertex capacities* as follows. We add a new source vertex $\hat{s}$, with unit-capacity edges to each terminal $s_i$, and a new target vertex $\hat{t}$, with unit capacity edges from each terminal $t_i$. Finally, we assign every edge of $G$ unit capacity and cost equal to its length, and we assign each vertex of $G$ capacity 1.

Now we need to compute a minimum-cost flow in $G'$ from $\hat{s}$ to $\hat{t}$ with value $k$. If no such flow exists, the given instance of $k$-min-sum is infeasible. Otherwise, the minimum-cost flow decomposes into $k$ vertex-disjoint paths from $\hat{s}$ to $\hat{t}$, each carrying one unit of flow. Removing the new vertices $\hat{s}$ and $\hat{t}$ leaves us with $k$ vertex-disjoint paths in $G$, each connecting some terminal $s_i$ to the corresponding terminal $t_j$.

To compute the minimum-cost flow quickly, we further reduce $G'$ to a planar flow network $H$ with only *edge* capacities, by replacing each vertex $v$ (except $\hat{s}$ and $\hat{t}$) with a clockwise cycle $C_v$ of $\deg(v)$ directed edges, each with unit capacity and zero cost. We also redirect the edges incident to $v$ to distinct vertices of $C_v$, so that the resulting graph remains planar. We then compute a minimum-cost flow in $H$ in $O(kn)$ time by performing $k$ iterations of the classical successive shortest path algorithm [68, 69, 70], using the $O(n)$-time shortest-path algorithm of Henzinger *et al.* [31] at each iteration. Finally, we project the resulting flow back to $G'$ by contracting each cycle $C_v$ to its original vertex $v$.

It remains only to prove that the algorithm is correct. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be the set of $k$ paths that comprise a minimum-cost flow with value $k$ in $H$. Because each edge in $H$ has unit capacity, the paths in $\mathcal{P}$ are pairwise edge-disjoint. Without loss of generality, the $i$th path $P_i$ contains the edges $\{\hat{s}, s_i\}$ and $\{t_i, \hat{t}\}$.

The paths in $\mathcal{P}$ partition the bounded faces of $G$ into $k + 1$ regions, each of which is bounded by at most two paths in $\mathcal{P}$. If a path $P_i$ runs clockwise along the boundary of a region, then we say that $P_i$ is the *left border* of every face in the region. Every bounded face of $H$ has at most one left border path. In particular, for any vertex $v$ in $G$, the face of $H$ bounded by $C_v$ has at most one path as its left border; on the other hand, any path in $\mathcal{P}$ that uses edges in $C_v$ must be the left border of $C_v$. Thus, at most one path in $\mathcal{P}$ uses edges in $C_v$.

We conclude that the set of paths in $G$ corresponding to $\mathcal{P}$ is a feasible solution to the $k$-min-sum problem. Conversely, any set of $k$ vertex-disjoint paths in $G$ can be transformed into a feasible flow in $H$ with value $k$. QED.

## 5.2 STRUCTURE

Let $G$ be an undirected plane graph with non-negative edge lengths, and let $s_4$, $s_3$, $s_1$, $t_1$, $s_2$, $t_2$, $t_3$, $t_4$ be eight distinct vertices in clockwise order around the outer face, as shown in Figure 5.1. Let $\mathcal{Q} = \{Q_1, \ldots, Q_4\}$ denote the unique optimal solution to this 4-min-sum instance, where each path $Q_i$ connects $s_i$ to $t_i$, and let $\mathcal{P} = \{P_1, P_2, P_4\}$ denote the unique optimal solution to the induced 3-min-sum problem that omits the demand pair $s_3t_3$, where again each path $P_i$ connects $s_i$ to $t_i$. We can compute $\mathcal{P}$ in $O(n^4 \log n)$ time using the algorithm of Kobayashi and Sommer [9], or in $O(n^5)$ time using the more general algorithm of Borradaile *et al.* [62].

We assume without loss of generality that the paths in $\mathcal{P}$ and $\mathcal{Q}$ do not use edges on the outer face. If necessary to enforce this assumption, we can connect the terminals using an outer cycle of eight infinite-weight edges.

The paths in $\mathcal{P}$ divide $G$ into four regions, as shown in Figure 5.1(a). Let $X$ be the unique region adjacent to all the paths in $\mathcal{P}$. For each index $i \neq 3$, let $C_i$ denote the subpath of $\partial G$ from $s_i$ to $t_i$ that shares no edges with $X$, let $R_i$ denote the closed region bounded by $P_i$ and $C_i$, and let $R_i^\circ$ denote the half-open region $R_i \setminus P_i$.

### 5.2.1 Envelopes

Fix a reference point $z$ on the boundary path $C_4$. Let $\pi$ be some path from $s_i$ to $t_i$, for some index $i$. We say that a point $x \notin \pi$ lies *below* $\pi$ if $x$ lies on the same side of $\pi$ as the point $z$, and *above* $\pi$ otherwise.

Now fix two indices $i \leq j$. Let $\pi$ be an arbitrary path from $s_i$ to $t_i$, and let $\rho$ be an arbitrary path from $s_j$ to $t_j$; these two paths may intersect arbitrarily. If $i = j$, let $D$ be the path in $\partial G$ from $s_i$ to $t_i$ that lies above $\pi$ and $E$ be the path in $\partial G$ from $s_j$ to $t_j$ that lies below $\rho$. Otherwise, let $D$ and $E$ be the unique disjoint paths in $\partial G$ from $s_i$ to $t_i$ and from $s_j$ to $t_j$, respectively. The paths $\pi$ and $\rho$ divide the interior of $G$ into connected regions. Let $U$ be the unique region with the entire path $D$ on its boundary, and let $L$ be the unique region with the entire path $E$ on its boundary. Finally, let $U(\pi, \rho) = \partial U \setminus D$ and $L(\pi, \rho) = \partial L \setminus E$.

Intuitively, for most choices of $i$ and $j$, $U(\pi, \rho)$ is the "upper envelope" of $\pi$ and $\rho$, and $L(\pi, \rho)$ is the "lower envelope" of $\pi$ and $\rho$. However, when $i = 1$ and $j = 2$, the path $U(\pi, \rho)$ is better thought of as the "left envelope" (because it lies *below* $\rho$), and $L(\pi, \rho)$ is better thought of as the "right envelope" (because it lies *above* $\rho$); fortunately, this exception arises only in the proof of Lemma 5.3.

**Lemma 5.2.** For any terminal-to-terminal paths $\pi$ and $\rho$, we have $\ell(U(\pi, \rho)) + \ell(L(\pi, \rho)) \leq \ell(\pi) + \ell(\rho)$.

*Proof.* Each component of $U(\pi, \rho) \setminus \pi$ is an open subpath of $\rho$ that lies entirely above $\pi$ and therefore is disjoint from $L(\pi, \rho)$. It follows that every edge in $U(\pi, \rho) \cap L(\pi, \rho)$ is an edge of $\pi$. Similarly, every edge in $U(\pi, \rho) \cap L(\pi, \rho)$ is an edge of $\rho$. QED.

### 5.2.2 How the paths in P and Q intersect

We begin by proving several structural properties of the 4-min-sum solution $\mathcal{Q}$ that will help us compute it quickly once we know the 3-min-sum solution $\mathcal{P}$. Our structural observations are summarized in the following theorem:

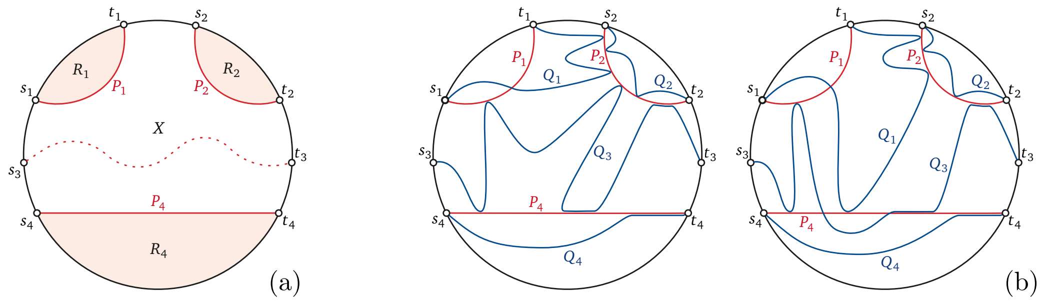

Figure 5.1: (a) Terminals, paths in $\mathcal{P}$, and the regions they define. (b) Typical structures for $\mathcal{Q}$.

**Theorem 5.1.** If $Q_i$ crosses $P_j$, then either $i = j = 1$, or $i = j = 2$, or $i = 3$ and $j = 4$. Moreover, either $Q_1 \subset R_1$ or $Q_2 \subset R_2$ or both, and $Q_4 \subset R_4$.

Figure 5.1(b) shows two typical structures for $\mathcal{Q}$ that are consistent with this theorem. We prove Theorem 5.1 using a series of exchange arguments, with the following high-level structure. Suppose some pair of paths $P_i$ and $Q_j$ cross, in violation of Theorem 5.1. By considering upper and lower envelopes of various paths in $\mathcal{P}$ and $\mathcal{Q}$, we construct new sets $\mathcal{P}'$ and $\mathcal{Q}'$ of vertex-disjoint paths. Then we argue, usually via Lemma 5.2, that $\ell(\mathcal{P}) + \ell(\mathcal{Q} \geq \ell(\mathcal{P}') + \ell(\mathcal{Q}')$, contradicting the unique optimality of $\mathcal{P}$ and $\mathcal{Q}$.

**Lemma 5.3.** $Q_1$ does not cross $P_2$, and $Q_2$ does not cross $P_1$.

*Proof.* Suppose for the sake of argument that $Q_1$ crosses $P_2$. Let $P_2'$ be the "right envelope" $L(Q_1, P_2)$ and let $Q_1'$ be the "left envelope" $U(Q_1, P_2)$. By definition, $P_2'$ is a path from $s_2$ to $t_2$, and $Q_1'$ is a path from $s_1$ to $t_1$. Let $\mathcal{P}' = \{P_1, P_2', P_4\}$ and $\mathcal{Q}' = \{Q_1', Q_2, Q_3, Q_4\}$.

The path $P_2$ separates $P_2'$ from both $P_1$ and $P_4$, so the paths in $\mathcal{P}'$ are vertex-disjoint. Similarly, $Q_1$ separates $Q_1'$ from $Q_2$, $Q_3$, and $Q_4$, so the paths in $\mathcal{Q}'$ are vertex-disjoint.

Lemma 5.2 implies that $\ell(\mathcal{P}) + \ell(\mathcal{Q}) \geq \ell(\mathcal{P}') + \ell(\mathcal{Q}')$. However, the unique optimality of $\mathcal{P}$ implies $\ell(\mathcal{P}) < \ell(\mathcal{P}')$, and the unique optimality of $\mathcal{Q}$ implies that $\ell(\mathcal{Q}) < \ell(\mathcal{Q}')$, so we have a contradiction. We conclude that $Q_1$ does not cross $P_2$.

A symmetric argument implies that $Q_2$ does not cross $P_1$. QED.

**Lemma 5.4.** $Q_1$ and $Q_2$ do not cross $P_4$.

*Proof.* The proof is similar to that of Lemma 5.3 Suppose for the sake of argument that $Q_1$ crosses $P_4$. Let $P_4' = L(Q_1, P_4)$ and $Q_1' = U(Q_1, P_4)$. Let $\mathcal{P}' = \{P_1, P_2, P_4'\}$ and $\mathcal{Q}' = \{Q_1', Q_2, Q_3, Q_4\}$. $P_4$ separates $P_4'$ from $P_1$ and $P_2$, so the walks in $\mathcal{P}'$ are pairwise vertex-disjoint. $Q_1$ separates $Q_1'$ from $Q_2$, $Q_3$, and $Q_4$, so the walks in $\mathcal{Q}'$ are pairwise vertex-disjoint.

The optimality of $\mathcal{P}$ implies $\ell(\mathcal{P}) < \ell(\mathcal{P}')$, and the optimality of $\mathcal{Q}$ implies that $\ell(\mathcal{Q}) < \ell(\mathcal{Q}')$. On the other hand, $\ell(\mathcal{P}) + \ell(\mathcal{Q}) \geq \ell(\mathcal{P}') + \ell(\mathcal{Q}')$, a contradiction.

A symmetric argument implies that $Q_2$ does not cross $P_4$. QED.

**Lemma 5.5.** $Q_3$ crosses neither $P_1$ nor $P_2$.

*Proof.* We prove that $Q_3$ does not cross $P_1$; the proof for the other statement is symmetric.

Suppose for the sake of argument that $Q_3$ crosses $P_1$. Let $P_1' = U(P_1, Q_3)$, $P_2' = U(P_2, Q_3)$, and $P_4' = U(P_4, Q_4)$. Let $Q_3' = L(P_1, L(P_2, Q_3))$ and $Q_4' = L(P_4, Q_4)$. Finally, let $\mathcal{P}' = \{P_1', P_2', P_4'\}$ and $\mathcal{Q}' = \{Q_1, Q_2, Q_3', Q_4'\}$. As in the previous proofs, we claim that $\mathcal{P}'$ and $\mathcal{Q}'$ are sets of *vertex-disjoint* paths.

$P_1$ separates $P_1'$ from $P_2'$. Suppose for the sake of argument that $P_1'$ meets $P_4'$ at a vertex $x$. Since $x$ is on $P_1'$, it is inside $R_1$ and it is on or above $Q_3$. Since $x$ is on $P_4'$, it is either on $P_4$ or $Q_4$. If $x$ is on $P_4$, then since $x$ is inside $R_1$, $P_1$ touches $P_4$. If $x$ is on $Q_4$, then since $x$ is on or above $Q_3$, $Q_3$ touches $Q_4$. In both cases we obtain a contradiction. A similar argument shows that $P_2'$ does not meet $P_4'$, so the walks in $\mathcal{P}'$ are pairwise vertex-disjoint.

$Q_1$ and $Q_2$ are trivially disjoint, and $Q_3$ separates $Q_1$ and $Q_2$ from $Q_3'$ and $Q_4'$. Suppose $Q_3'$ intersects $Q_4'$ at a vertex $x$. Since $x$ is on $Q_4'$, it is inside $R_4$ and on or below $Q_4$. Because $x$ is on $Q_3'$, it is either in $P_1$, $P_2$, or $Q_3$. If $x$ is on $Q_3$, then because $x$ is on or below $Q_4$, $Q_3$ crosses below $Q_4$. If $x$ is on $P_1$ or $P_2$, then since $x$ is in $R_4$, either $P_1$ or $P_2$ touches $P_4$. In all cases we obtain a contradiction, so the paths in $\mathcal{Q}'$ are pairwise vertex-disjoint.

Each component of $Q_3' \setminus Q_3$ is an open subpath of $P_1$ or $P_2$ that lies entirely below $Q_3$ and therefore is not contained in $P_1'$ or $P_2'$. Similarly, each component of $P_1' \setminus P_1$ is an open subpath of $Q_3$ that lies entirely above $P_1$ and therefore is not contained in $P_2'$ or $Q_3'$, and each component of $P_2' \setminus P_2$ is an open subpath of $Q_3$ that lies entirely above $P_2$ and therefore is not contained in $P_1'$ or $Q_3'$.

It follows that $\ell(P_1') + \ell(P_2') + \ell(Q_3') \leq \ell(P_1) + \ell(P_2) + \ell(Q_3)$, and therefore $\ell(\mathcal{P}) + \ell(\mathcal{Q}) \geq \ell(\mathcal{P}') + \ell(\mathcal{Q}')$, contradicting the unique optimality of $\mathcal{P}$ and $\mathcal{Q}$. QED.

**Corollary 5.1.** $Q_4$ does not meet $P_1$ or $P_2$.

**Lemma 5.6.** $Q_4$ lies entirely in $R_4$.

*Proof.* For the sake of argument, suppose $Q_4$ leaves $R_4$. Define two new paths $P_4' = U(P_4, Q_4)$ and $Q_4' = L(P_4, Q_4)$. Let $\mathcal{P}' = \{P_1, P_2, P_4'\}$ and $\mathcal{Q} = \{Q_1, Q_2, Q_3, Q_4'\}$.

Corollary 5.1 implies that $P_4'$ does not meet $P_1$ or $P_2$, so the walks in $\mathcal{P}'$ are pairwise vertex-disjoint. On the other hand, $Q_4$ separates $Q_4'$ from $Q_1$, $Q_2$, and $Q_3$, so the paths in $\mathcal{Q}'$ are pairwise vertex-disjoint. Lemma 5.2 implies $\ell(P_4') + \ell(Q_4') \leq \ell(P_4) + \ell(Q_4)$, and therefore $\ell(\mathcal{P}') + \ell(\mathcal{Q}') \leq \ell(\mathcal{P}) + \ell(\mathcal{Q})$, contradicting the unique optimality of $\mathcal{P}$ and $\mathcal{Q}$. QED.

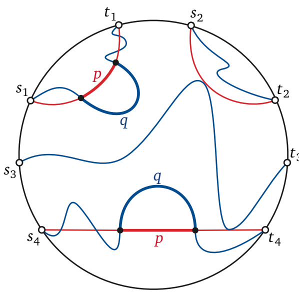


Figure 5.2: An impossible configuration of optimal paths, for the proofs of Lemmas 5.6 and 5.8.

To complete the proof of Theorem 5.1, we must consider two cases, depending on whether or not $Q_3$ crosses $P_4$. Typical solutions for these two cases are illustrated in Figure 5.1(b).

**The case where $Q_3$ does not cross $P_4$.**

**Lemma 5.7.** If $Q_3$ does not cross $P_4$, then $Q_1$ and $Q_2$ do not meet $P_4$.

*Proof.* $Q_3$ separates $s_1, t_1, s_2, t_2$ from $s_4$ and $t_4$. Thus, $Q_3$ separates $Q_1$ and $Q_2$ from $P_4$. QED.

**Lemma 5.8.** If $Q_3$ does not cross $P_4$, then every component of $Q_1 \setminus R_1^\circ$ meets $P_2$, and every component of $Q_2 \setminus R_2^\circ$ meets $P_1$.

*Proof.* Suppose some component $q$ of of $Q_1 \setminus R_1^\circ$ does not meet $P_2$, as shown at the top of Figure 5.2. We can derive a contradiction using a similar exchange argument to Lemma 5.6.

The endpoints $x$ and $y$ of $q$ must lie on $P_1$; let $p$ denote the subpath $P_1[x, y]$. Define two new paths $P_1' = P_1 \setminus p \cup q$ and $Q_1' = Q_1 \setminus q \cup p$. Clearly $P_1'$ and $Q_1'$ are both walks from $s_1$ to $t_1$. Let $\mathcal{P}' = \{P_1', P_2, P_4\}$ and $\mathcal{Q} = \{Q_1', Q_2, Q_3, Q_4\}$. Lemma 5.7 and our assumption that $q$ does not meet $P_2$ imply that the walks in $\mathcal{P}'$ are pairwise vertex-disjoint. On the other hand, $p$ lies in the disk enclosed by $P_1' \cup C_1$, which implies that the walks in $\mathcal{Q}'$ are also pairwise vertex-disjoint. The optimality of $\mathcal{P}$ implies that $\ell(\mathcal{P}) < \ell(\mathcal{P}')$, and the optimality of $\mathcal{Q}$ implies that $\ell(\mathcal{Q}) < \ell(\mathcal{Q}')$, but clearly $\ell(\mathcal{P}) + \ell(\mathcal{Q}) = \ell(\mathcal{P}') + \ell(\mathcal{Q}')$, so we have a contradiction.

A symmetric argument implies every component of $Q_2 \setminus R_2^\circ$ meets $P_1$. QED.

**Lemma 5.9.** If $Q_3$ does not cross $P_4$, then either $Q_1 \subset R_1$ or $Q_2 \subset R_2$ or both.

*Proof.* For the sake of argument, suppose $Q_1$ leaves $R_1$ and $Q_2$ leaves $R_2$. Let $S_1$ be the closed region bounded by $Q_1 \cup C_1$ and let $S_2$ be the closed region bounded by $Q_2 \cup C_2$. We call each component of $S_1 \setminus R_1^\circ$ a *left finger*, and each component of $S_2 \setminus R_2^\circ$ a *right finger*.

Lemma 5.8 and the Jordan curve theorem imply that each finger is a topological disk that intersects both $P_1$ and $P_2$. Thus, the fingers can be linearly ordered by their intersections with $P_1$ from $s_1$ to $t_1$ (from bottom to top in Figure 5.3). Because $Q_1$ is a simple path, the fingers intersect $Q_1$ in the same order. Without loss of generality, suppose the last finger in this order is a right finger. Let $s$ be the last left finger, and let $s'$ be the right finger immediately after $s$.

Let $w$ be the last node of $P_1$ (closest to $t_1$) that lies in $s$, and let $y$ be the last node of $P_2$ (closest to $t_2$) that that lies in $s'$. We define four subpaths $p_1 = P_1[w, t_1]$, $q_1 = Q_1[w, t_1]$, $p_2 = P_2[s_2, y]$, and $q_2 = Q_2[s_2, y]$, as shown on the left of Figure 5.3. (Paths $p_2$ and $q_2$ could enclose more than one right finger.)

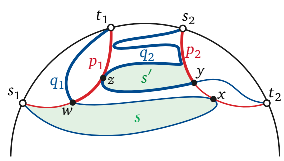


Figure 5.3: Another impossible configuration, for the proof of Lemma 5.9.

Now exchange the subpaths $p_1 \leftrightarrow q_1$ and $p_2 \leftrightarrow q_2$ to define four new walks $P_1' = P_1 \setminus p_1 \cup q_1$, $Q_1' = Q_1 \setminus q_1 \cup p_1$, $P_2' = P_2 \setminus p_2 \cup q_2$, and $Q_2 = Q_2 \setminus q_2 \cup p_2$. Finally, let $\mathcal{P}' = \{P_1', P_2', P_4\}$ and $\mathcal{Q}' = \{Q_1', Q_2', Q_3, Q_4\}$. As in previous lemmas, we argue that $\mathcal{P}'$ and $\mathcal{Q}'$ are sets of vertex-disjoint walks.

Lemma 5.5 implies that $Q_3$ does not cross $P_1$ or $P_2$, and trivially $Q_3$ does not cross $Q_1$. Thus, none of the paths $Q_3, P_4, Q_4$ touches any of the paths $p_1, q_1, p_2, q_2$. It follows that $P_4$ does not touch either $P_1'$ or $P_2'$, and similarly, $Q_3$ and $Q_4$ does not touch either $Q_1'$ or $Q_2'$.

We define two more auxiliary nodes $x$ and $z$, as shown on the right in Figure 5.3. Let $x$ be the first vertex of $P_2$ also on $Q_1$. Vertex $y$ must precede $x$ on $P_2$, because $x \in s$ and $y \in s'$. Let $z$ be the first vertex of $P_1$ also on $q_2$. Vertex $w$ must precede $z$ on $P_1$, because $w \in s$ and $z \in s'$.

Trivially, $q_1$ does not meet $q_2$, and $P_1 \setminus p_1$ does not meet $P_2 \setminus p_2$. Any left finger formed from $q_1$ must succeed $s$. Because $s$ is the last left finger, $q_1$ does not form any left fingers and does not touch $P_2$. By definition, $z$ is the first node of $P_1$ also on $q_2$. On the other hand, all vertices of $P_1 \setminus p_1$ (except $w$) precede $w$ on $P_1$, which in turn strictly precedes $z$ on $P_1$, so $P_1 \setminus p_1$ is disjoint from $q_2$. We conclude that $P_1'$ does not meet $P_2'$, implying that the walks in $\mathcal{P}'$ are vertex-disjoint:

Trivially, $p_1$ does not meet $p_2$, and $Q_1 \setminus q_1$ does not meet $Q_2 \setminus q_2$. Since $q_1$ does not meet $P_2$, $x$ is the first vertex of $P_2$ also on $Q_1 \setminus q_1$. On the other hand, all vertices of $p_2$ (except $y$) precede $y$ on $P_2$, which in turn strictly precedes $x$ on $P_2$, so $Q_1 \setminus q_1$ is disjoint from $p_2$. Any right finger whose boundary contains a subpath of $Q_2 \setminus q_2$ must precede $s'$, and any right finger that meets $p_1$ must succeed $s$. Because no right fingers lie strictly between $s$ and $s'$, the path $Q_2 \setminus q_2$ does not form any right fingers that meet $p_1$. We conclude that $Q_1'$ does not meet $Q_2'$, which implies that the walks in $\mathcal{Q}'$ are vertex-disjoint.

Finally, we clearly have $\ell(\mathcal{P}) + \ell(\mathcal{Q}) = \ell(\mathcal{P}') + \ell(\mathcal{Q}')$, contradicting the unique optimality of $\mathcal{P}$ and $\mathcal{Q}$. QED.

**The case where $Q_3$ crosses $P_4$.**

**Lemma 5.10.** If $Q_3$ crosses $P_4$, then every component of $Q_1 \setminus R_1$ meets $P_2$ or $P_4$ or both, and every component of $Q_2 \setminus R_2$ meets $P_1$ or $P_4$ or both.

*Proof.* The proof is the same as that of Lemma 5.8. QED.

**Lemma 5.11.** If $Q_3$ crosses $P_4$, then either $Q_1$ or $Q_2$ (or both) touches $P_4$.

*Proof.* The proof is similar to that of Lemma 5.3. Suppose for the sake of argument that $Q_1$ and $Q_2$ do not touch $P_4$.

Let $q$ be a maximal component of $Q_3 \cap R_4$, and let $a$ and $b$ be the endpoints of $q$. Let $p = P_4[a, b]$, and define two new paths $P_4' = P_4 \setminus p \cup q$ and $Q_3' = Q_3 \setminus q \cup p$. Let $\mathcal{P}' = \{P_1, P_2, P_4'\}$ and $\mathcal{Q}' = \{Q_1, Q_2, Q_3', Q_4\}$.

$P_4$ separates $P_1$ and $P_2$ from $q$, so $P_1$ and $P_2$ are disjoint from $P_4'$ and the walks in $\mathcal{P}'$ are pairwise vertex-disjoint. By assumption, $Q_1$ and $Q_2$ do not touch $p$, so $Q_1$ and $Q_2$ are disjoint from $Q_3'$. Also, $P_4'$ separates $p$ from $Q_4$, so $Q_3'$ is disjoint from $Q_4$. It follows that the walks in $\mathcal{Q}'$ are pairwise vertex-disjoint.

The unique optimality of $\mathcal{P}$ implies that $\ell(\mathcal{P}') < \ell(\mathcal{P}')$, and the unique optimality of $\mathcal{Q}$ implies that $\ell(\mathcal{Q}') < \ell(\mathcal{Q}')$, but clearly $\ell(\mathcal{P}) + \ell(\mathcal{Q}) = \ell(\mathcal{P}') + \ell(\mathcal{Q}')$, a contradiction. QED.

In the rest of this subsection we assume without loss of generality that $Q_1$ touches $P_4$. Our goal is to show that $Q_2 \subset R_2$. Let $u$ be the last vertex on $Q_1 \cap P_4$, and let $b$ be the first vertex on $P_4[u, t_4]$ that is on $Q_3$, as shown in Figure 5.4(a) and (b) below.

**Lemma 5.12.** If vertex $u$ precedes vertex $v$ in $P_4$, then either $u$ precedes $v$ in $Q_3$, or $P_4[u, v] = \text{rev}(Q_3[v, u])$.

*Proof.* Suppose for the sake of argument that $u$ precedes $v$ in $P_4$, $v$ precedes $u$ in $Q_3$, and $P_4[u, v] \neq \text{rev}(Q_3[v, u])$. Without loss of generality, assume that none of the vertices in $Q_3(v, u)$ are on $P_4$. Let $q_3 = Q_3[v, u]$ and $p_4 = P_4[u, v]$. Define $P_4'$ by removing all cycles from $P_4 \setminus p_4 \cup \text{rev}(q_3)$, and define $Q_3'$ by removing all cycles from $Q_3 \setminus q_3 \cup \text{rev}(p_4)$. This means that $Q_3'$ is a simple path from $s_3$ to $t_3$ that does not cross $Q_3$, and $P_4'$ is a simple path from $s_4$ to $t_4$ that does not cross $P_4$. Let $\mathcal{P} = \{P_1, P_2, P_4'\}$ and $\mathcal{Q} = \{Q_1, Q_2, Q_3', Q_4\}$.

If $Q_3(v, u) \subseteq R_4$, then $p_4$ does not meet $Q_4$ by Lemma 5.6, and $Q_3$ separates $Q_3'$ from $Q_1$ and $Q_2$. It follows that the walks in $\mathcal{Q}$ are pairwise vertex-disjoint. Path $P_4$ separates $q_3$ from $P_1$ and $P_2$, so the paths in $\mathcal{P}$ are pairwise vertex-disjoint.

If $Q_3(v, u) \cap R_4 = \varnothing$, then $p_4$ does not meet $Q_1$ or $Q_2$ by Lemma 5.4, and $Q_3$ separates $Q_3'$ from $Q_4$. It follows that the walks in $\mathcal{Q}$ are pairwise vertex-disjoint. Walk $P_4[s_4, u] \cup Q_3[u, t_3]$ separates $q_3$ from $P_1$ and $P_2$, so the paths in $\mathcal{P}$ are pairwise vertex-disjoint.

The optimality of $\mathcal{P}$ implies that $\ell(\mathcal{P}') < \ell(\mathcal{P}')$, and the optimality of $\mathcal{Q}$ implies that $\ell(\mathcal{Q}') < \ell(\mathcal{Q}')$, but clearly $\ell(\mathcal{P}) + \ell(\mathcal{Q}) = \ell(\mathcal{P}') + \ell(\mathcal{Q}')$. QED.

**Lemma 5.13.** Suppose $Q_3$ crosses $P_4$ and $Q_1$ touches $P_4$. If $u$ and $b$ are defined as above, then $Q_2$ does not touch $P_4[u, b]$.

*Proof.* Suppose for the sake of contradiction that $Q_2$ touches $P_4[u, b]$. We define six special vertices $v$, $y$, $z$, $w$, $x$, and $a$, as shown in Figure 5.4(a):

- Vertex $v$ is the first vertex on $Q_2 \cap P_4$. By assumption, $v$ is on $P_4[u, b]$.
- If $Q_3[s_3, b]$ touches $P_1$, then $y$ is the last vertex in their intersection. Otherwise, $y = s_1$.
- If $Q_3[b, t_3]$ touches $P_2$, then $z$ is the first vertex in their intersection. Otherwise, $z = t_2$.
- Vertex $w$ is the first vertex on $P_1[y, t_1]$ that is also on $Q_1$.
- Vertex $x$ is the last vertex on $P_2[s_2, z]$ that is also on $Q_2$.
- Vertex $a$ is the first vertex on $Q_3[y, t_3]$ that is also on $P_4$.

Let $p_1 = P_1[w, t_1]$, $q_1 = Q_1[w, t_1]$, $p_2 = P_2[s_2, x]$, $q_2 = Q_2[s_2, x]$, $q_3 = Q_3[a, b]$, and $p_4 = P_4[a, b]$. Let $P_1' = P_1 \setminus p_1 \cup q_1$, $Q_1' = Q_1 \setminus q_1 \cup p_1$, $P_2' = P_2 \setminus p_2 \cup q_2$, and $Q_2' = Q_2 \setminus q_2 \cup p_2$. Let $P_4' = L(P_4, P_4 \setminus p_4 \cup q_3)$ and $Q_3' = U(Q_3, Q_3 \setminus q_3 \cup p_4)$. Finally, let $\mathcal{P}' = \{P_1', P_2', P_4'\}$ and $\mathcal{Q}' = \{Q_1', Q_2', Q_3', Q_4\}$.

$Q_1[u, t_1] \cup P_4$ separates $P_1 \setminus p_1$ from $q_2$, and $Q_2[s_2, v] \cup P_4$ separates $P_2 \setminus p_2$ from $q_1$. It follows that $P_1'$ and $P_2'$ are disjoint. Any vertex on both $P_1'$ and $P_4'$ must lie on $q_1$, because $P_4' \subset R_4$, but $Q_3$ separates $q_1$ from $P_4'$. It follows that $P_1'$ and $P_4'$ are disjoint. A symmetric argument implies that $P_2'$ and $P_4'$ are disjoint. We conclude that the walks in $\mathcal{P}'$ are pairwise vertex-disjoint.

$Q_1[u, t_1] \cup P_4$ separates $Q_1 \setminus q_1$ from $p_2$, and $Q_2[s_2, v] \circ P_4$ separates $Q_2 \setminus q_2$ from $p_1$, so $Q_1'$ and $Q_2'$ are disjoint. The definition of $w$ implies that $Q_3[s_3, b]$ does not meet $p_1$, and

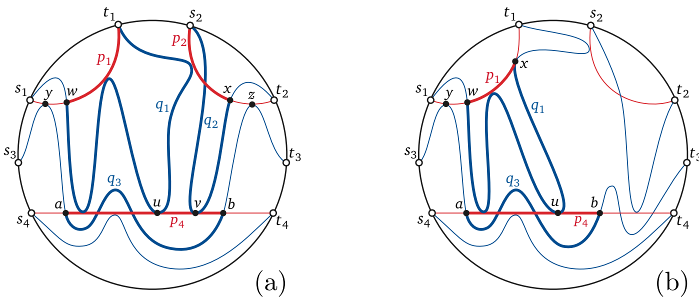


Figure 5.4: More impossible configurations, for the proofs of (a) Lemma 5.13 and (b) Lemma 5.15.

the Jordan Curve Theorem implies that $Q_3[b, t_3]$ does not meet $p_1$. Thus, $p_1$ and $Q_3$ are disjoint, which implies that $Q'_1$ and $Q_3$ are disjoint. It follows that if $Q'_1$ and $Q'_3$ share a vertex $c$, we must have $c \in Q'_4 \setminus Q_3 \subseteq p_4$ and therefore $c \in Q_1 \setminus q_1$. But this is impossible, because $Q_3[s_3, y] \cup P_1[y, w] \cup Q_1[w, t_1]$ separates $Q_1 \setminus q_1$ from $p_4$. A similar argument shows that $Q'_2$ is disjoint from $Q'_3$. Finally, $Q_3$ separates $Q'_3$ from $Q_4$. We conclude that the walks in $\mathcal{Q}'$ are pairwise vertex-disjoint. One can show show that $\ell(P'_4) + \ell(Q'_3) \leq \ell(P_4) + \ell(Q_3)$; for details, see Lemma 5.14. It follows that $\ell(\mathcal{P}) + \ell(\mathcal{Q}) \leq \ell(\mathcal{P}') + \ell(\mathcal{Q}')$, contradicting the unique optimality of $\mathcal{P}$ and $\mathcal{Q}$. QED.

**Lemma 5.14.** In the proof of Lemma 5.13, we have $\ell(P'_4) + \ell(Q'_3) \leq \ell(P_4) + \ell(Q_3)$.

*Proof.* Suppose $e$ is an edge in $P'_4$ and $Q'_3 \setminus Q_3$. The edge $e$ is strictly above $Q_3$ and on $p_4$. Thus $e$ is not in $P_4 \setminus p_4 \cup q_3$ and must be strictly below it. But Lemma 5.12 implies that $e \in p_4$ cannot be both strictly above $Q_3$ and strictly below $P_4 \setminus p_4 \cup q_3$. It follows that any edge in $P'_4$ and $Q'_3$ must be in $Q_3$. A similar argument shows that any edge in $P'_4$ and $Q'_3$ must be in $P_4$. It follows that $\ell(P'_4) + \ell(Q'_3) \leq \ell(P_4) + \ell(Q_3)$. QED.

**Lemma 5.15.** If $Q_3$ crosses $P_4$ and $Q_1$ touches $P_4$, then some component of $Q_1 \setminus R_1$ touches both $P_2$ and $P_4$.

*Proof.* Lemma 5.13 implies that $Q_2$ does not touch $P_4[u, b]$. Suppose for the sake of argument that no component of $Q_1 \setminus R_1$ touches both $P_4$ and $P_2$. We define four special vertices $y$, $w$, $x$, and $a$, as shown in Figure 5.4(b):

- If $Q_3[s_3, b]$ touches $P_1$, then $y$ is the last vertex in their intersection. Otherwise, $y = s_1$.
- Vertex $w$ is the first vertex on $P_1[y, t_1]$ that is also on $Q_1$.
- Vertex $x$ is the first vertex on $Q_1[u, t_1]$ that is also on $P_1$.
- Vertex $a$ is the first vertex on $Q_3[y, t_3]$ that is also on $P_4$.

Let $p_1 = P_1[w, x]$, $q_1 = Q_1[w, x]$, $p_4 = P_4[a, b]$, and $q_3 = Q_3[a, b]$. Let $P_1' = P_1 \setminus p_1 \cup q_1$ and $Q_1' = Q_1 \setminus q_1 \cup p_1$. Define $P_4' = L(P_4, P_4 \setminus p_4 \cup q_3)$ and $Q_3' = U(Q_3, Q_3 \setminus q_3 \cup p_4)$. Let $\mathcal{P}' = \{P_1', P_2, P_4'\}$ and $\mathcal{Q}' = \{Q_1', Q_2, Q_3', Q_4\}$.

An argument similar to the proof of Lemma 5.13 shows that $\mathcal{P}'$ and $\mathcal{Q}'$ are each sets of pairwise disjoint walks; see Lemma 5.16 for details. The same argument as Lemma 5.14 implies that $\ell(P_4') + \ell(Q_3') \leq \ell(P_4) + \ell(Q_3)$. As usual, it follows that $\ell(\mathcal{P}') + \ell(\mathcal{Q}') \leq \ell(\mathcal{P}) + \ell(\mathcal{Q})$, contradicting the unique optimality of $\mathcal{P}$ and $\mathcal{Q}$. QED.

**Lemma 5.16.** In the proof of Lemma 5.15, $\mathcal{P}'$ and $\mathcal{Q}'$ are each sets of disjoint walks.

*Proof.* By assumption, $q_1$ is disjoint from $P_2$, so $P_1'$ is disjoint from $P_2$. The same argument as in the proof of Lemma 5.13 shows that $P_1'$ is disjoint from $P_4'$. Additionally, $P_4$ separates $P_2$ from $P_4'$. It follows that the walks in $\mathcal{P}'$ are pairwise vertex-disjoint.

$P_1[x, t_1] \cup Q_1[x, u] \cup P_4$ separates $p_1$ from $Q_2$, so $Q_1'$ is disjoint from $Q_2$. Suppose for the sake of argument that $Q_1'$ and $Q_3'$ meet at $c$. The definition of $y$ implies that $Q_3[s_3, b]$ does not meet $p_1$, while the definition of $x$ implies that $Q_3[b, t_3]$ does not meet $p_1$. Thus, $c \in Q_3'$ implies $c \in p_4$, and $c \in Q_1'$ implies $c \in Q_1 \setminus q_1$. But $Q_1[x, t_1]$ doesn't meet $p_4$ by the definition of $x$, and $Q_3[s_3, y] \cup P_1[y, w] \cup Q_1[w, t_1]$ separates $Q_1[s_1, w]$ from $p_4$, so we have a contradiction. By assumption, $p_4$ and $Q_2$ are disjoint, so $Q_2$ and $Q_3'$ are disjoint. $Q_3$ separates $Q_3'$ from $Q_4$. We have shown that the walks in $\mathcal{Q}'$ are pairwise vertex-disjoint. QED.

**Lemma 5.17.** If $Q_3$ crosses $P_4$ and $Q_1$ touches $P_4$, then $Q_2$ does not touch $P_4$.

*Proof.* Define a *far-reaching subpath* to be a component of $Q_1 \setminus R_1$ that touches both $P_4$ and $P_2$ or a component of $Q_2 \setminus R_2^\circ$ that touches both $P_4$ and $P_1$. Lemma 5.15 says that some component of $Q_1 \setminus R_1$ is a far-reaching subpath. Symmetrically, if $Q_2$ were to touch $P_4$, then some component of $Q_2 \setminus R_2$ would also be a far-reaching subpath, but the Jordan Curve Theorem implies that we cannot have both a far-reaching subpath of $Q_1 \setminus R_1$ and a far-reaching subpath of $Q_2 \setminus R_2$. It follows that $Q_2$ does not touch $P_4$. QED.

**Lemma 5.18.** If $Q_3$ crosses $P_4$ and $Q_1$ touches $P_4$, then $Q_2 \subset R_2$.

*Proof.* The proof is similar to that of Lemma 5.9. By Lemma 5.15 and 5.17, there exists a component of $Q_1 \setminus R_1^\circ$ that touches both $P_2$ and $P_4$, and $Q_2$ does not touch $P_4$. We will show that $Q_2 \subset R_2$. As in the proof of Lemma 5.9, define $S_1$ to be the closed region bounded by $Q_1 \cup C_1$, define $S_2$ to be the closed region bounded by $Q_2 \cup C_2$, call each component of $S_1 \setminus R_1^\circ$ intersecting both $P_1$ and $P_2$ a *left finger*, and call each component of $S_2 \setminus R_2^\circ$ a *right finger*. Let $f$ be the unique left finger that touches both $P_2$ and $P_4$.

The proof of Lemma 5.9 shows that no left fingers exist after the last right finger, and no right fingers exist after the last left finger. Repeatedly applying this observation shows that no fingers exist except for the first finger $f$. Since no right fingers exist, $Q_2$ does not touch $P_1$. Additionally, $Q_2$ does not touch $P_4$, so Lemma 5.10 implies that $Q_2 \subset R_2$. QED.

**Corollary 5.2.** If $Q_3$ crosses $P_4$, then either $Q_1 \subset R_1$ or $Q_2 \subset R_2$.

The proof of Theorem 5.1 is now complete.

## 5.3 SUBGRAPH SOLUTIONS

Our algorithm solves several parallel instances of $k$-min-sum inside certain subgraphs of $G$. To prove that our algorithm is correct, we need to argue that the subgraph solutions coincide exactly with portions of the desired global solution. As an intermediate step, we first show that the subgraph solutions interact with the global solution in a limited way. Unlike the structural results in the previous section, the following lemma applies to planar $k$-min-sum instances for *arbitrary* $k$.

**Lemma 5.19.** Let $(G, \{s_i, t_i \mid 1 \leq i \leq k\})$ be a planar instance of $k$-min-sum, with all terminals $s_i$ and $t_i$ on $\partial G$, whose unique solution is $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$. Let $S$ be a subset of $\{1, 2, \ldots, k\}$ such that the induced planar min-sum instance $(G, \{s_i, t_i \mid i \in S\})$ is parallel. Let $H$ be a subgraph of $G$ such that

(1) $Q_i \cap H \neq \varnothing$ if and only if $i \in S$, and

(2) for all distinct $i, j \in S$, no component of $Q_i \cap H$ separates components of $Q_j \cap H$ from each other in $H$.

For each index $i \in S$, let $u_i$ and $v_i$ be vertices of $Q_i \cap \partial H$ such that $Q_i[u_i, v_i] \subseteq H$. Finally, suppose $(H, \{u_i, v_i \mid i \in S\})$ is a parallel planar min-sum instance, whose unique solution is $\Pi = \{\pi_i \mid i \in S\}$. Then for all indices $i, j \in S$, if $i \neq j$, then $\pi_i$ does not cross $Q_j$.

*Proof.* First we establish some notation and terminology. Let $\kappa = |S|$, and re-index the terminals so that $S = \{1, 2, \ldots, \kappa\}$ and the counterclockwise order of terminals around the outer face of $H$ is $u_1, \ldots, u_\kappa, v_\kappa, \ldots, v_1$. Fix an index $i$ such that $1 \leq i < \kappa$, and consider the paths $Q_i$ and $\pi_{i+1}$.

Let $C$ ("ceiling") denote the path in $\partial G$ from $s_i$ to $t_i$ that does not contain $s_{i+1}$ or $t_{i+1}$, and let $A$ be the closed region bounded by $C$ and $Q_i$. A point in $G$ is *above* $Q_i$ if it lies in $A \setminus Q_i$ and *below* $Q_i$ if it does not lie in $A$.

Similarly, let $F$ ("floor") denote the path in $\partial H$ from $u_{i+1}$ to $v_{i+1}$ that does not contain $u_i$ or $v_i$, and let $B$ be the closed region bounded by $F$ and $\pi_{i+1}$. A point in $H$ is *below* $\pi_{i+1}$ if it lies in $B \setminus \pi_{i+1}$ and *above* $\pi_{i+1}$ if it does not lie in $B$.

Paths $Q_i$ and $\pi_{i+1}$ also divide the interior of $G$ into connected regions, exactly one of which has the entire path $C$ on its boundary; call this region $U$. Finally, let $Q'_i$ denote the unique path in $G$ from $s_i$ to $t_i$ such that $C \cup Q'_i$ is the boundary of $U$. Every point on $Q'_i$ lies on or above $Q_i$, and our assumption (2) implies that every point in $Q'_i \cap H$ lies on or above $\pi_{i+1}$. Thus, intuitively, $Q'_i$ is the "upper envelope" of $Q_i$ and $\pi_{i+1}$. In particular, $Q'_i = Q_i$ if and only if $Q_i$ and $\pi_{i+1}$ are disjoint.

Similarly, paths $Q_i$ and $\pi_{i+1}$ divide the interior of $H$ into closed connected regions, exactly one of which contains $F$ on its boundary; call this region $L$. Let $\pi'_{i+1}$ denote the unique path in $H$ from $u_{i+1}$ to $v_{i+1}$ such that $D \cup \pi'_{i+1}$ is the boundary of $L$. Assumption (2) implies that every point on $\pi'_{i+1}$ lies on or below both $\pi_{i+1}$ and $Q_i$. Thus, intuitively, $\pi'_{i+1}$ is the "lower envelope" of $Q_i$ and $\pi_{i+1}$. In particular, $\pi'_{i+1} = \pi_{i+1}$ if and only if $Q_i$ and $\pi_{i+1}$ are disjoint.

Each component of $Q'_i \setminus Q_i$ is an open subpath of $\pi_{i+1}$ that lies entirely above $Q_i$ and therefore is not contained in $\pi'_{i+1}$. Similarly, every component of $\pi'_{i+1} \setminus \pi_{i+1}$ is an open subpath of $Q_i \cap H$ that lies entirely below $\pi_{i+1}$ and therefore is not contained in $Q'_i$. It follows that $\ell(Q'_i) + \ell(\pi'_{i+1}) \le \ell(Q_i) + \ell(\pi_{i+1})$.

Finally, let $\mathcal{Q}' = \{Q'_1, \dots, Q'_{\kappa-1}, Q_\kappa, \dots, Q_k\}$ and $\Pi' = \{\pi_1, \pi'_2, \dots, \pi'_\kappa\}$; see Figure 5.1 for an example of our construction.

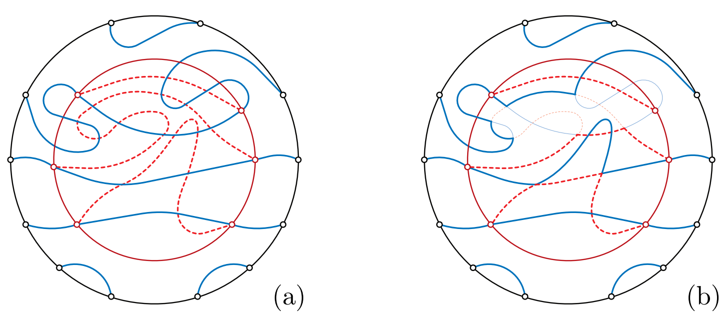


Figure 5.5: Proof of Lemma 5.19. The inner red circle is $\partial H$. (a) The original paths $\mathcal{Q}$ (solid blue) and $\Pi$ (dashed red). (b) The transformed paths $\mathcal{Q}'$ (solid blue) and $\Pi'$ (dashed red).

Now suppose for the sake of argument that $Q_i$ crosses $\pi_{i+1}$ for some index $i$, or equivalently, that $\mathcal{Q}' \neq \mathcal{Q}$ and $\Pi' \neq \Pi$. As usual, to derive a contradiction, we need to show that $\mathcal{Q}'$ and $\Pi'$ are sets of disjoint walks. The following case analysis implies that the walks in $\mathcal{Q}'$ are pairwise disjoint:

- None of the paths $Q_{\kappa+1}, \dots, Q_k$ intersect $H$. On the other hand, for all $i < \kappa$, $Q'_i \setminus Q_i$ is a subset of $\pi_{i+1}$ and therefore lies in $H$. Trivially, $Q_{\kappa+1}, \dots, Q_k$ are disjoint from

$Q_1, \ldots, Q_\kappa$. Thus, paths $Q'_1 \ldots, Q'_{\kappa-1}, Q_\kappa$ are disjoint from paths $Q_{\kappa+1}, \ldots, Q_k$.

- $Q_\kappa$ lies entirely below $Q_{\kappa-1}$ and therefore entirely below $Q'_{\kappa-1}$.
- Now consider any point $x \in Q'_i$, for any index $1 \le i < \kappa - 1$. Point $x$ lies on or above $Q_i$ (because every point in $Q'_i$ lies on or above $Q_i$), and therefore lies above $Q_{i+1}$. So we must have $x \in \pi_{i+2}$ and therefore $x \in H$. But because $x \in Q'_i \cap H$, $x$ lies either on or above $\pi_{i+1}$, and therefore lies above $\pi_{i+2}$. So $x$ cannot lie on $Q'_{i+1}$. We conclude that $Q'_i$ and $Q'_{i+1}$ are disjoint.

Similar case analysis implies that the walks in $\Pi'$ are pairwise disjoint:

- $\pi_1$ lies entirely above $\pi_2$ and therefore entirely above $\pi'_2$.
- Now consider any point $x \in \pi'_{i+1}$, for any index $1 < i < \kappa$. Point $x$ lies on or below $Q_i$, and therefore below $Q_{i-1}$. On the other hand, $x$ lies on or below $\pi_{i+1}$, and therefore lies below $\pi_i$. So $x$ cannot lie in $\pi'_i$. We conclude that $\pi'_i$ and $\pi'_{i+1}$ are disjoint.

The unique optimality of $\Pi$ and $\mathcal{Q}$ implies $\ell(\Pi) < \ell(\Pi')$ and $\ell(\mathcal{Q}) < \ell(\mathcal{Q}')$. On the other hand, we immediately have

$$\ell(\Pi) + \ell(\mathcal{Q}) = \ell(\pi_1) + \sum_{i=1}^{\kappa-1} \big(\ell(Q_i) + \ell(\pi_{i+1})\big) + \sum_{i=\kappa}^{k} \ell(Q_i) \tag{5.1}$$

$$\le \ell(\pi_1) + \sum_{i=1}^{\kappa-1} \big(\ell(Q'_i) + \ell(\pi'_{i+1})\big) + \sum_{i=\kappa}^{k} \ell(Q_i) \tag{5.2}$$

$$= \ell(\Pi') + \ell(\mathcal{Q}'), \tag{5.3}$$

giving us a contradiction.

We conclude that $\pi_i$ does not cross $Q_{i-1}$ for any index $i$. It follows immediately that $\pi_i$ does not cross (in fact, does not *touch*) any $Q_j$ such that $j < i - 1$. A symmetric argument implies that $\pi_i$ does not cross any $Q_j$ such that $j > i$. QED.

## 5.4 4-MIN-SUM ALGORITHM

Now we are finally ready to describe our algorithm for computing $\mathcal{Q}$ given $\mathcal{P}$. By Theorem 5.1, we can assume without loss of generality that $Q_2 \subset R_2$. We define five ***anchor vertices*** as follows; see Figure 5.6.

- If $Q_1$ meets $P_2$, then $a$ is the first vertex of $Q_1$ that is also on $P_2$, and $b$ is the first vertex in the suffix $P_2(a, t_2]$ that is also on $Q_2$; otherwise, $a = t_1$ and $b = s_2$.

- If $Q_3$ meets $P_2$, then $c$ is the first vertex in their intersection; otherwise, $c = t_3$.
- If $P_4$ meets the prefix $Q_3[s_3, c)$, then $d$ is the last vertex of $P_4$ in their intersection; otherwise, $d = s_4$.
- Finally, $e$ is the first vertex of the suffix $P_4(d, t_4]$ that is also on $Q_4$.

We also split each path $Q_i$ into a prefix $Q_i^s$ and a suffix $Q_i^t$ that meet at a single vertex. Specifically, we split $Q_1$ at $a$, we split $Q_2$ at $b$, we split $Q_3$ at $c$, and we split $Q_4$ at $e$. Thus, for example, $Q_1^s = Q_1[s_1, a]$ and $Q_1^t = Q_1[a, t_1]$.

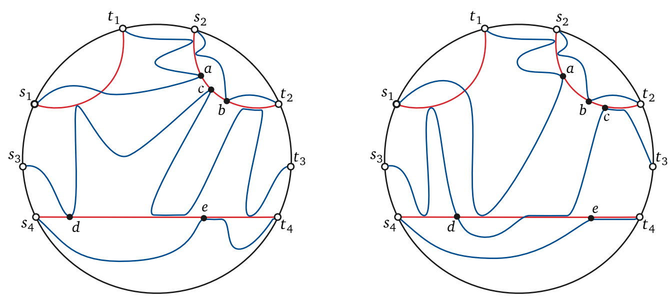


Figure 5.6: Anchor vertices $a, b, c, d, e$.

Now suppose we know the locations of the anchor vertices $a$, $b$, $c$, $d$, and $e$. (Our final $k$-min-sum algorithm actually enumerates all $O(n^5)$ possible locations for these vertices.) Our algorithm computes $\mathcal{Q}$ in three phases; each phase solves a parallel instance of the $k$-min-sum problem (with $k = 2$ or $k = 3$) in a subgraph of $G$ in $O(n)$ time, via minimum-cost flows. The subpaths of $\mathcal{Q}$ computed in each phase are shown in Figure 5.7.

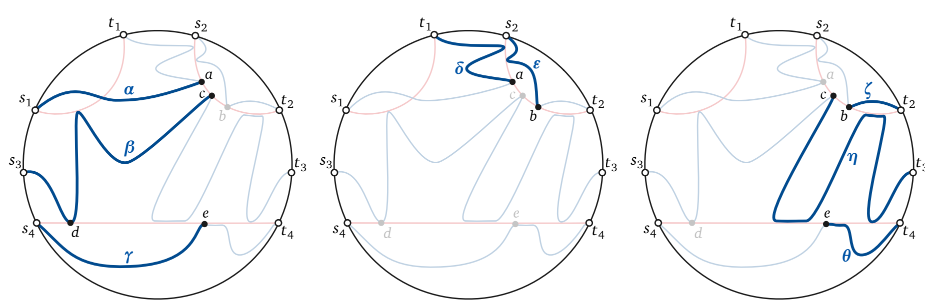


Figure 5.7: Subpaths of $\mathcal{Q}$ computed by the three phases of our algorithm.

- Let $H_1$ be the subgraph of $G$ obtained by deleting every vertex in $R_2$ except $a$ and $c$, every edge incident to $s_4$ or $e$ outside of $R_4$, and every vertex of $P_4(d, t_4]$ except $e$. The first phase of our algorithm computes the shortest set of vertex-disjoint paths in $H_1$ from $s_1$ to $a$, from $s_3$ to $c$, and from $s_4$ to $e$. Call these paths $\alpha$, $\beta$, and $\gamma$, respectively.

- If $Q_1$ and $P_2$ are disjoint, let $\delta = t_1$ and $\varepsilon = s_2$. Otherwise, let $H_2$ be the subgraph of $G$ obtained by deleting every vertex of $P_2(a, t_2]$ except $b$, all edges incident to $b$ that leave $R_2$, and every vertex of $\alpha$ except $a$. The second phase of our algorithm computes the shortest vertex-disjoint paths in $H_2$ from $t_1$ to $a$ and from $s_2$ to $b$. Call these paths $\delta$ and $\varepsilon$, respectively.

- Finally, let $H_3$ be the subgraph of $G$ obtained by deleting all vertices in $\alpha \cdot rev(\delta)$, all vertices in $\beta[s_3, b)$, all vertices in $\gamma[s_4, e)$, and all vertices in $\varepsilon[s_2, b)$. The last phase of our algorithm computes the shortest vertex-disjoint paths in $H_3$ from $b$ to $t_2$, from $c$ to $t_3$, and from $e$ to $t_4$. Call these paths $\zeta$, $\eta$, and $\theta$, respectively.

**Lemma 5.20.** $Q_3^t$ does not cross $P_4$.

*Proof.* The proof is similar to that of Lemma 5.3. The lemma is obvious if $c = t_3$, so assume $Q_3$ touches $P_2$.

Suppose $Q_3^t$ crosses $P_4$. Let $q$ be any component of $Q_3^t \cap R_4$. The endpoints $x$ and $y$ of $q$ must lie on $P_4$; let $p$ denote the subpath $P_4[x, y]$. Define two new paths $Q_3' = Q_3 \setminus q \cap p$ and $P_4' = P_4 \setminus p \cup q$. Let $\mathcal{P}' = \{P_1, P_2, P_4'\}$ and $\mathcal{Q}' = \{Q_1, Q_2, Q_3', Q_4\}$.

$P_4$ separates $P_1$ and $P_2$ from $P_4'$, so the walks in $\mathcal{P}'$ are pairwise vertex-disjoint. On the other hand, $Q_3^s \cup P_2$ separates $Q_1$ from $p$, and $Q_2$ does not touch $P_4 \supseteq p$. Furthermore, subpath $p$ lies outside the disk enclosed by $P_4' \cup C_4$, so by Lemma 5.6, $Q_4$ does not meet $p$. It follows that the walks in $\mathcal{Q}'$ are also pairwise vertex-disjoint.

The unique optimality of $\mathcal{P}$ implies $\ell(\mathcal{P}) < \ell(\mathcal{P}')$, and the unique optimality of $\mathcal{Q}$ implies $\ell(\mathcal{Q}) < \ell(\mathcal{Q}')$. But $\ell(\mathcal{P}) + \ell(\mathcal{Q}) = \ell(\mathcal{P}') + \ell(\mathcal{Q}')$, so we have a contradiction. QED.

**Lemma 5.21.** $\alpha = Q_1^s$, $\beta = Q_3^s$, and $\gamma = Q_4^s$.

*Proof.* Suppose, for the sake of argument, that $(\alpha, \beta, \gamma) \neq (Q_1^s, Q_3^s, Q_4^s)$, and define a new set of walks $\mathcal{Q}' := \{\alpha \circ Q_1^t,\ Q_2,\ \beta \circ Q_3^t,\ \gamma \circ Q_4^t\}$. The following exhaustive case analysis shows that the paths of $\mathcal{Q}'$ are vertex-disjoint.

- Paths $\alpha$, $\beta$, and $\gamma$ are disjoint by definition.

- Similarly, $Q_1^t$, $Q_2$, $Q_3^t$, $Q_4^t$ are subpaths of paths in $\mathcal{Q}$ and thus are disjoint by definition.

- $P_2$ separates $Q_2$ from $\alpha$, $\beta$, and $\gamma$.

- Lemma 5.19 implies that $\beta$ and $\gamma$ do not cross $Q_1^s$, and therefore do not touch $Q_1^t$.

- Lemma 5.19 also implies that $\alpha$ does not cross $Q_3^s$, and therefore does not touch $Q_3^t$.

- Lemma 5.19 also implies that $\alpha$ and $\beta$ do not cross $Q_4^s$, and therefore do not touch $Q_4^t$.
- Finally, if $d = s_4$, then the definition of $H_1$ implies that $\gamma$ does not leave $R_4^\circ$ except at $s_4$ and $e$, so Lemma 5.20 implies that $\gamma$ is disjoint from $Q_3^t$. If $d \neq s_4$, then Lemma 5.19 implies that $\gamma$ does not cross $Q_3[s_3, d]$; on the other hand, $Q_3^t$ does not meet $Q_3[s_3, d]$. The definition of $H_1$ implies that $\gamma$ does not cross the path $P_4[d, t_4]$ and only meets it at $d$ or $e$; on the other hand, neither $d$ nor $e$ are on $Q_3^t$. Because $Q_3[s_3, d] \circ P_4[d, t_4]$ separates $\gamma$ from $Q_3^t$, we conclude that $Q_3^t$ and $\gamma$ are disjoint.

Because the walks in $\mathcal{Q}'$ are vertex-disjoint, the unique optimality of $\mathcal{Q}$ implies that $\ell(\mathcal{Q}) < \ell(\mathcal{Q}')$. On the other hand, the lemmas in Section 5.2.2 and the definitions of the anchor vertices imply that $Q_1^s$, $Q_3^s$, and $Q_4^s$ are indeed paths in $H_1$ between the appropriate terminals. Moreover, $Q_1^s$, $Q_3^s$, and $Q_4^s$ are vertex-disjoint, because they are subpaths of the disjoint paths in $\mathcal{Q}$. Thus, the unique optimality of $\{\alpha, \beta, \gamma\}$ implies that $\ell(\alpha) + \ell(\beta) + \ell(\gamma) < \ell(Q_1^s) + \ell(Q_3^s) + \ell(Q_4^s)$. It follows that $\ell(\mathcal{Q}') < \ell(\mathcal{Q})$, giving us the desired contradiction. QED.

**Lemma 5.22.** *$rev(\delta) = Q_1^t$ and $\varepsilon = Q_2^s$.*

*Proof of Lemma 5.22.* The lemma is obvious if $Q_1$ and $P_2$ are disjoint, so assume otherwise.

For the sake of argument, suppose $(rev(\delta), \varepsilon) \neq (Q_1^t, Q_2^s)$, and let $\mathcal{Q}' = \{Q_1^s \circ rev(\delta), \varepsilon \circ Q_2^t, Q_3, Q_4\}$. The following exhaustive case analysis implies that the walks in $\mathcal{Q}'$ are pairwise disjoint.

- $\delta$ and $\varepsilon$ are disjoint by definition.
- $Q_1^s$, $Q_2^t$, $Q_3$, and $Q_4$ are disjoint by definition of $\mathcal{Q}$.
- Lemma 5.19 implies that $\delta$ does not cross $Q_2^s$, and therefore does not touch $Q_2^t$.
- The path $\alpha \circ P_2[a, t_2]$ separates $\delta$ and $\varepsilon$ from $Q_3$ and therefore from $Q_4$.
- Lemma 5.21 implies that $Q_1^s \cap V(H_2) = \{a\}$. It follows that $\varepsilon$ does not touch $Q_1^s$.

The unique optimality of $\mathcal{Q}$ now implies that $\ell(\mathcal{Q}) < \ell(\mathcal{Q}')$.

On the other hand, the lemmas in Section 5.2.2 and the definitions of the anchor vertices imply that $Q_1^t$ and $Q_2^s$ are vertex-disjoint paths in $H_2$ between the appropriate terminals. Thus, the unique optimality of $\{\delta, \varepsilon\}$ implies that $\ell(Q_1^t) + \ell(Q_2^s) > \ell(\delta) + \ell(\varepsilon)$, and therefore $\ell(\mathcal{Q}) > \ell(\mathcal{Q}')$, giving us the desired contradiction. QED.

**Lemma 5.23.** *$\zeta = Q_2^t$, $\eta = Q_3^t$, and $\theta = Q_4^t$.*

*Proof of Lemma 5.23.* Suppose, for the sake of argument, that $(\zeta, \eta, \theta) \neq (Q_2^t, Q_3^t, Q_4^t)$, and let $\mathcal{Q}' := \{Q_1,\ Q_2^s \circ \zeta,\ Q_3^s \circ \eta,\ Q_4^s \circ \theta\}$. As usual, exhaustive case analysis implies that the walks in $\mathcal{Q}'$ are pairwise disjoint. Several cases rely on Lemmas 5.21 and 5.22, which imply that $\alpha \circ rev(\delta) = Q_1$, $\beta = Q_3^s$, $\gamma = Q_4^s$, and $\varepsilon = Q_2^s$.

- $\zeta$, $\eta$, and $\theta$ are disjoint by definition.
- $Q_1$, $Q_2^s$, $Q_2^s$, and $Q_2^s$ are disjoint by definition of $\mathcal{Q}$.
- $Q_1$ is disjoint from $H_3$ and thus disjoint from $\zeta$, $\eta$, and $\theta$.
- $Q_2^s \cap H_3 = \{b\}$, so $Q_2^s$ is disjoint from $\eta$ and $\theta$.
- $Q_3^s \cap H_3 = \{c\}$, so $Q_3^s$ is disjoint from $\zeta$ and $\theta$.
- $Q_4^s \cap H_3 = \{e\}$, so $Q_3^s$ is disjoint from $\zeta$ and $\eta$.

The unique optimality of $\mathcal{Q}$ now implies that $\ell(\mathcal{Q}) < \ell(\mathcal{Q}')$.

On the other hand, $Q_2^t$, $Q_3^t$, and $Q_4^t$ are paths between appropriate terminals in $H_3$. Thus, the unique optimality of $\{\zeta, \eta, \theta\}$ implies that $\ell(Q_2^t) + \ell(Q_3^t) + \ell(Q_3^t) > \ell(\zeta) + \ell(\eta) + \ell(\theta)$, and therefore $\ell(\mathcal{Q}) > \ell(\mathcal{Q})$, giving us the desired contradiction. QED.

Finally, we describe our overall 4-min-sum algorithm. First, in a preprocessing phase, we compute $\mathcal{P}$ using the algorithm of Kobayashi and Sommer [9]. Then for all possible choices for the anchor vertices $a, b, c, d, e$, we compute the paths $\alpha, \beta, \gamma, \delta, \varepsilon, \zeta, \eta, \theta$ as described above, first under the assumption that $Q_2 \subset R_2$, and then under the symmetric assumption that $Q_1 \subset R_1$ (mirroring the definitions of the anchor vertices and the paths). The previous lemmas imply that for the correct choice of anchor vertices, and the correct assumption $Q_1 \subset R_1$ or $Q_2 \subset R_2$, the resulting walks $Q_1 = \alpha \circ rev(\delta)$, $Q_2 = \varepsilon \circ \zeta$, $Q_3 = \beta \circ \eta$, and $Q_4 = \gamma \circ \theta$ comprise the optimal solution for the given instance of the 4-min-sum problem.

Altogether, our algorithm solves $O(n^5)$ parallel instances of 2-min-sum and 3-min-sum, each in $O(n)$ time, via minimum-cost flows. Thus, the overall running time of our algorithm is $O(n^6)$.

## 5.5 EXTENSION OF 4-MIN-SUM ALGORITHM

Here we briefly describe how to extend the algorithm to instances where the cyclic order of the terminals is $s_1, t_1, s_2, t_2, t_3, \ldots, t_k, s_k, \ldots, s_3$. In this case, Lemma 5.1 becomes

**Lemma 5.24.** Let $P_1, \ldots, P_\ell, P_{\ell+2}, \ldots, P_k$ be the solution to the $(k-1)$-min-sum instance where we omit the terminal pair $s_{\ell+1}t_{\ell+1}$ and $P_i$ connects $s_i$ to $t_i$. Let $Q_1, \ldots, Q_k$ be the paths in the desired $k$-min-sum solution, where $Q_i$ connects $s_i$ to $t_i$. For all $i \neq 3$, the path $P_i$ divides $G$ into two regions; let $R_i$ be the region containing neither $s_3$ nor $t_3$.

- If $Q_i$ crosses $P_j$, then either $i = j = 1$, or $i = j = 2$, or $i = 3$ and $j = 4$.
- Either $Q_1 \subset R_1$ or $Q_2 \subset R_2$ or both. Furthermore, $Q_i \subset R_i$ for $i \geq 4$.

Instead of defining five anchor vertices, we need to define $2k - 3$ anchor vertices. If we assume the anchor vertices are known, then solving the $k$-min-sum problem reduces to solving two parallel instances of the $(k-1)$-min-sum problem and one parallel instance of 2-min-sum. Each of these instances can be solved in $O(kn)$ time. Since we need to try all possible sets of anchor vertices, the resulting algorithm runs in $kn^{2k-2}$ time.

## 5.6 K-APPROXIMATION ALGORITHM

In this section we describe a $k$-approximation of the $k$-min-sum problem when all terminals are on a common face. That is, the algorithm computes a set of pairwise vertex-disjoint paths whose combined length is within a factor $k$ of optimal, assuming that a set of pairwise vertex-disjoint paths exists.

### 5.6.1 (2k-2)-approximation

First we describe a $(2k-2)$-approximation algorithm based on linear programming; later we will show how to modify the algorithm to get a $k$-approximation. We treat $G$ as a directed graph by replacing each edge $\{u, v\}$ with the arcs $(u, v)$ and $(v, u)$; we assume that $(u, v)$ and $(v, u)$ are embedded together. The $k$-min-sum problem is a special case of the minimum-cost multicommodity flow problem. Thus for each $i \in [k]$ and arc $(u, v) \in E(G)$, we construct variables $x_i(v, u)$ and $x_i(u, v)$, representing the flow for the $i$-th commodity through $(v, u)$ and $(u, v)$, respectively. We have the following integer program $\mathcal{I}$ for the $k$-min-sum problem:

$$\min \sum_{i \in [k], e \in E(G)} x_i(u, v)\ell(e) \tag{5.4}$$

$$\text{subject to} \sum_{i \in [k], (u,v) \in E(G)} x_i(u, v) \leq 1 \quad \forall v \in V(G) \tag{5.5}$$

$$\sum_{i\in[k],(v,w)\in E(G)} x_i(v,w) \le 1 \quad \forall v \in V(G) \tag{5.6}$$

$$\sum_{(u,v)\in E(G)} x_i(u,v) = \sum_{(v,w)\in E(G)} x_i(v,w) \quad \forall i \in [k], v \in V(G)\setminus\{s_i,t_i\} \tag{5.7}$$

$$1 + \sum_{(u,s_i)\in E(G)} x_i(u,s_i) = \sum_{(s_i,w)\in E(G)} x_i(s_i,w) \quad \forall i \in [k] \tag{5.8}$$

$$\sum_{(u,t_i)\in E(G)} x_i(u,t_i) = 1 + \sum_{(t_i,w)\in E(G)} x_i(t_i,w) \quad \forall i \in [k] \tag{5.9}$$

$$x_i(u,v) \in \{0,1\} \quad \forall i \in [k]; \forall u,v \in V(G) \tag{5.10}$$

Note that in any solution to $\mathcal{I}$, constraint (5.10) forces the left side of constraint (5.8) to be at least 1, so the right side of (5.4) is at least 1. Constraint (5.6) then implies that the right side of (5.8) is exactly 1; furthermore, for any $i \in [k]$, commodity $i$ has unit flow exiting $s_i$, and no other commodities have flow exiting $s_i$. Constraints (5.7) and (5.8) then imply that for any $i \in [k]$, no commodity has flow entering $s_i$. A symmetric argument shows that for any solution to $\mathcal{I}$ and for any $i \in [k]$, commodity $i$ is the only commodity with flow entering $t_i$, and no commodity has flow exiting $t_i$. In other words, the flow for any single commodity does not enter any of the terminals for any other commodity.

We define the linear program relaxation $\mathcal{L}$ by replacing the constraints $x_i(u,v) \in \{0,1\}$ in $\mathcal{I}$ with constraints $0 \le x_i(u,v) \le 1$. The program $\mathcal{L}$ has $O(kn^2)$ variables and $O(kn^2)$ constraints, so we can solve $\mathcal{L}$ in polynomial time using, say, the ellipsoid algorithm of Khachiyan [71]. If $\mathcal{L}$ is infeasible, then the original instance $G$ of the $k$-min-sum problem did not have a solution. Otherwise, let $\mathbf{x}$ be the assignment of values to the variables in the solution to $\mathcal{L}$, and let $\ell^*$ be the optimal value of the objective function of $\mathcal{L}$.

For each $i \in [k]$, the variables $x_i(u,v)$ form a flow $f_i$ of value 1 from $s_i$ to $t_i$. By the flow decomposition theorem, we can in polynomial time decompose this flow into a set $\mathcal{P}_i$ of flows such that the flows in $\mathcal{P}_i$ sum to $f_i$ and the support of each flow in $\mathcal{P}_i$ is either a cycle or a path from $s_i$ to $t_i$. In fact, we may assume without loss of generality that the support of each flow in $\mathcal{P}_i$ is a path and that all of these paths are pairwise noncrossing.

For each $i \in [k]$, we now have a set $\mathcal{P}_i$ of flow-paths whose values sum to 1. Furthermore, for each vertex $v$, the sum of the flow-values through $v$ is at most 1. The rest of the algorithm finds a way to round the flow-path values such that the resulting objective function has value at most $(2k-2)\ell^*$. This suffices for a $(2k-2)$-approximation because $\ell^*$ is at most the optimal value of the objective function of $\mathcal{I}$. To distinguish between the values of flow-paths and the value of the objective function, we will call the value of a flow-path $p$ its *weight* and denote it by $w(p)$.

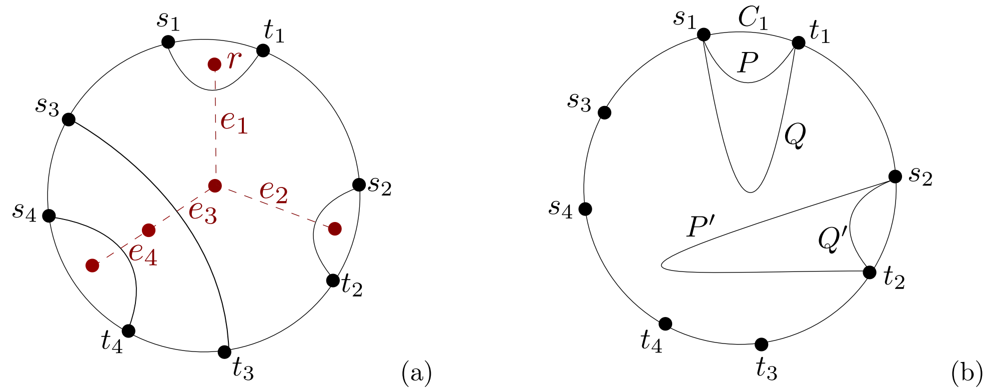


Figure 5.8: (a) A possible re-indexing of the terminal pairs after rooting the demand tree at $r$. The dashed red graph is $T$ and the solid black graph is $G_D \cup \partial G$. Edges $e_1, \ldots, e_4$ are in $T$. (b) An example where $P$ is above $Q$, and $P'$ is above $Q'$. Here $C_2$ is the portion of $\partial G$ from $s_2$ to $t_2$ containing all other terminals.

Pick an arbitrary leaf $r$ in the demand tree $T$ and root $T$ at $r$. Now re-index the terminal pairs such that if $(s_i, t_i)$ is an ancestor of $(s_j, t_j)$, then $i < j$. Note that $e_1$ is now the unique edge in $T$ that is incident to $r$. See Figure 5.8(a).

We let $C_1$ be the portion of the boundary between $s_1$ and $t_1$ and containing no other terminals, and for all $i > 1$, we let $C_i$ be the portion of the boundary between $s_i$ and $t_i$ containing $C_1$. Given two distinct paths $P, Q$, both with endpoints $s_i$ and $t_i$ on $\partial G$, we say that $P$ is *above* $Q$ if $P$ lies completely on or inside $C_i \circ Q$. See Figure 5.8(b).

For each pair $(s_i, t_i)$, we partition the set of flow-paths $\mathcal{P}_i$ into $2k-2$ parts $\mathcal{P}_{i,1}, \ldots, \mathcal{P}_{i,2k-2}$ of equal weight (i.e., each part is made up of flow-paths of total weight $1/(2k-2)$), such that for all $j \in [2k-3]$, the paths in $\mathcal{P}_{i,j}$ are all on or above the paths in $\mathcal{P}_{i,j+1}$. In order to do this, we may need to split a flow-path $p$ of weight $\alpha$ into two flow-paths with the same support as $p$ and combined weight $\alpha$. See Figure 5.9(a) for an example of a partition where no splitting of paths is required. Then, for each pair $(s_i, t_i)$, we pick the shortest path $Q_i$ in $\mathcal{P}_{i,k+i-2}$. Finally, return $\mathcal{Q}_a$, the set of picked paths. See Figure 5.9(b).

This completes the description of the algorithm. To show that the algorithm is correct, we need to show that the paths in $\mathcal{Q}_a$ have combined length at most $(2k-2)\ell^*$, and that the paths in $\mathcal{Q}_a$ are pairwise vertex-disjoint.

**Lemma 5.25.**

$$\sum_{i \in [k], e \in \mathcal{Q}_a} x_i(u, v)\ell(e) \leq (2k-2)\ell^* \tag{5.11}$$

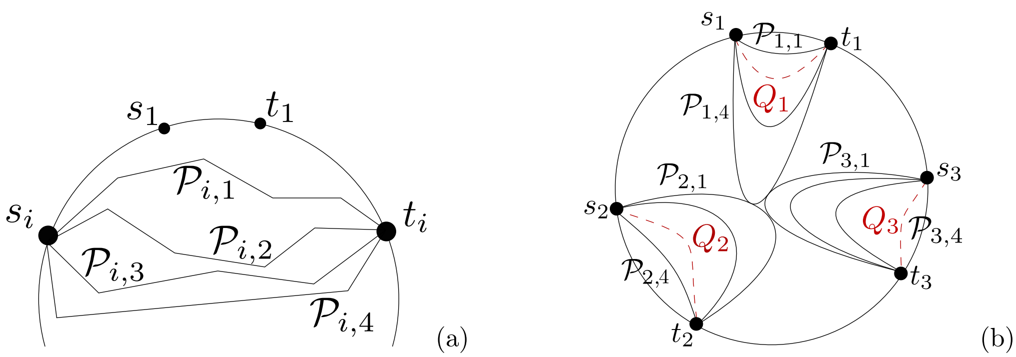


Figure 5.9: (a) Example of $\mathcal{P}_{i,1}, \ldots, \mathcal{P}_{i,4}$ if $k = 3$ and $\mathcal{P}_i$ is made up of four pairwise vertex-disjoint paths, each of weight $1/4$. (b) Example where $k = 3$ and $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ are each made up of four paths of equal weight. The algorithm returns the dashed red paths $Q_1, Q_2$, and $Q_3$. Note that $\mathcal{P}_{3,4} = \{Q_3\}$.

*Proof.* We have

$$\ell(Q_i) = (2k-2) \sum_{p \in \mathcal{P}_{i,k+i-2}} w(p)\ell(Q_i) \tag{5.12}$$

$$\leq (2k-2) \sum_{p \in \mathcal{P}_{i,k+i-2}} w(p)\ell(p) \tag{5.13}$$

$$\leq (2k-2) \sum_{p \in \mathcal{P}_i} w(p)\ell(p) \tag{5.14}$$

(Note that for the last inequality we use the fact that edges in $G$ have non-negative length.) This implies

$$\sum_{i \in [k], e \in \mathcal{Q}_a} x_i(u,v)\ell(e) = \sum_{i \in [k]} \ell(Q_i) \leq (2k-2) \sum_{i \in [k]} \sum_{p \in \mathcal{P}_i} w(p)\ell(p) = (2k-2)\ell^*. \tag{5.15}$$

QED.

This shows that the sum of the lengths of the paths of $\mathcal{Q}_a$ is within a factor $2k-2$ of optimal.

**Lemma 5.26.** The paths in $\mathcal{Q}_a$ are vertex-disjoint.

*Proof.* Suppose for the sake of argument that $i < j$ and picked paths $Q_i$ and $Q_j$ intersect at vertex $v$. It suffices to consider the cases where $(s_i, t_i)$ and $(s_j, t_j)$ are adjacent, because if paths connecting adjacent terminal pairs are vertex-disjoint, then all paths are pairwise vertex-disjoint. That is, it suffices to consider the cases where $(s_i, t_i)$ is a parent of $(s_j, t_j)$ and where $(s_i, t_i)$ and $(s_j, t_j)$ have a common parent. In Figure 5.9(b), $(s_1, t_1)$ is a parent of $(s_2, t_2)$, while $(s_2, t_2)$ and $(s_3, t_3)$ have a common parent. The need to balance these two cases is where the $2k-2$ comes from.

Suppose $(s_i, t_i)$ is a parent of $(s_j, t_j)$, so that $i < j$. In $\mathcal{P}_i$, all of the flow-paths below $Q_i$ also go through $v$. Similarly, in $\mathcal{P}_j$, all of the flow-paths above $Q_j$ also go through $v$. Thus we know that in $\mathcal{P}_i$, flow-paths of combined weight strictly more than $(k-i)/(2k-2)$ go through $v$, and in $\mathcal{P}_j$, flow-paths of combined weight strictly more than $(k+j-3)/(2k-2)$ go through $v$. Thus under $\mathcal{P}_i$ and $\mathcal{P}_j$, the sum of the weights of the flow-paths going through $v$ is strictly greater than $((k-i)+(k+j-3)/(2k-2) \geq 1$, which is impossible. See Figure 5.9(b), where the paths in $\mathcal{P}_{1,3}, \mathcal{P}_{1,4}, \mathcal{P}_{2,1}$, and $\mathcal{P}_{2,2}$ have combined weight 1, so red paths $Q_1$ and $Q_2$ must be vertex-disjoint.

Now suppose $(s_i, t_i)$ and $(s_j, t_j)$ have a common parent. Since $i, j > 1$, we have $i + j \geq 4$. In $\mathcal{P}_i$, all of the flow-paths above $Q_i$ also go through $v$. Similarly, in $\mathcal{P}_j$, all of the flow-paths above $Q_j$ also go through $v$. Thus we know that in $\mathcal{P}_i$, flow-paths of combined weight strictly more than $(k+i-3)/(2k-2)$ go through $v$, and in $\mathcal{P}_j$, flow-paths of combined weight strictly more than $(k+j-3)/(2k-2)$ go through $v$. Thus under $\mathcal{P}_i$ and $\mathcal{P}_j$, the sum of the weights of the flow-paths going through $v$ is strictly greater than $((k+i-3)+(k+j-3))/(2k-2) \geq 1$, which is impossible. See Figure 5.9(b), where the paths in $\mathcal{P}_{2,2}, \mathcal{P}_{2,1}, \mathcal{P}_{3,1}, \mathcal{P}_{3,2}$, and $\mathcal{P}_{3,2}$ have combined weight greater than 1, so the red paths $Q_2$ and $Q_3$ must be vertex-disjoint. QED.

This completes the proof of correctness for the $(2k-2)$-approximation.

### 5.6.2 k-approximation

To turn the $(2k-2)$-approximation algorithm into a $k$-approximation algorithm, note that we can modify the $(2k-2)$-approximation to give us a $(2d-2)$-approximation, where $d$ is the height of the demand tree $T$. Solve the linear program relaxation and arbitrarily root $T$. For each pair $(s_i, t_i)$, let $h(i)$ be the number of edges in the path from $e_i$ to the root in the $T$. That is, $h(1) = 1$, and for each child $e_j$ of $e_i$ we have $h(j) = h(i) + 1$. Note that $h(i) \leq d$ for all $i \in [k]$.

Now instead of partitioning the set of flow-paths $\mathcal{P}_i$ into $2k-2$ parts $\mathcal{P}_{i,1}, \ldots, \mathcal{P}_{i,2k-2}$ of equal weight, we partition it into $2d-2$ parts of equal weight. Likewise, for each pair $(s_i, t_i)$, instead of picking the shortest path in $\mathcal{P}_{i,k+i-2}$, we pick the shortest path in $\mathcal{P}_{i,d+h(i)-2}$. These are the only changes to the algorithm. The result is a $(2d-2)$-approximation algorithm. The proof of correctness is almost the same as that of the $(2k-2)$-approximation and is omitted.

Now we use the $(2d-2)$-approximation to get a $k$-approximation as follows. The demand

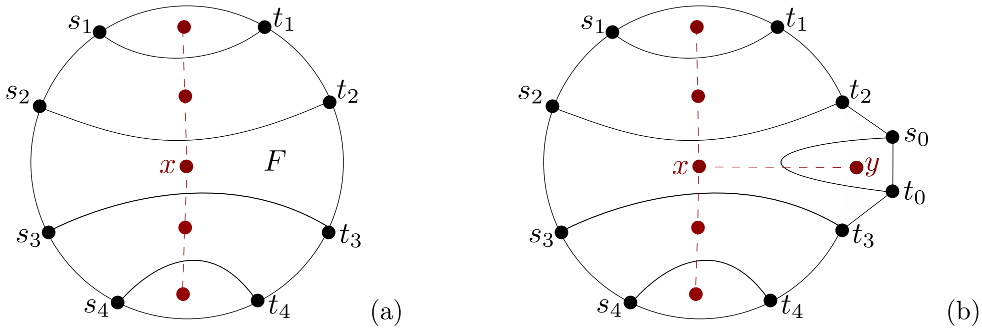


Figure 5.10: An example where $x$ is not a leaf. (a) The solid black graph is $G_D \cup \partial G$, and the dashed red graph is $T$ (b) The solid black graph is $H_D \cup \partial H$, and the dashed red graph is $T_H$. Here $u = t_2$ and $v = t_3$. Edges $\{t_2, s_0\}$ and $\{t_0, t_3\}$ have infinite length, and edge $\{s_0, t_0\}$ has zero length.

tree $T$ consists of $k$ edges. If $T$ is a path, then $G$ is a parallel instance and can be solved exactly using min-cost flow, as described in Section 5.1. Otherwise, there is some vertex $x$ in $T$ that is within $\lfloor k/2 \rfloor$ hops of every other vertex in $T$.

If $x$ is a leaf in $T$, then we can root $T$ at $x$ and apply the $(2d-2)$-approximation. Since $d \leq k/2$, the $(2d-2)$-approximation is a $(k-2)$-approximation.

If $x$ is not a leaf, then we first construct a graph $H$ by adding vertices and edges to $G$ as well as a terminal pair, as follows. The tree vertex $x$ corresponds to a face $F$ in $G_D \cup \partial G$, where $G_D$ is the demand graph. Since $x$ is not a leaf, $F$ is incident to at least four terminals. See Figure 5.10(a). Let $u$ and $v$ be two terminals that are incident to $F$, appear consecutively in the cyclic order of the terminals on $\partial G$, and are not part of the same terminal pair. We add a new pair of terminals $(s_0, t_0)$, embedding both terminals in the infinite face of $G$. Furthermore, we add an edge $\{u, s_0\}$ of infinite length, an edge $\{s_0, t_0\}$ of length 0, and an edge $\{t_0, v\}$ of infinite length. The resulting graph $H$ is a planar instance of the $(k+1)$-min-sum problem where all terminals are on $\partial H$. See Figure 5.10(b). (Strictly speaking, the construction of $H$ is not necessary, but it allows us to directly apply the $(2d-2)$-approximation.)

Let $T_H$ be the demand tree corresponding to $H$, and let $y$ be the unique leaf of $T_H$ incident to the edge of $T_H$ that corresponds to $(s_0, t_0)$. If we root $T_H$ at $y$, then $T_H$ has height $\lfloor k/2 \rfloor + 1$. Applying the $(2d-2)$-approximation algorithm to $H$ then gives us a $k$-approximation for $H$. Note that both the optimal solution to $H$ and the solution given by the $k$-approximation for $H$ use the zero-length edge $\{s_0, t_0\}$ to connect $s_0$ to $t_0$, and this edge is vertex-disjoint from $G$. Thus the $k$-approximation for $H$ also gives us a $k$-approximation for $G$.

## 5.7 OPEN PROBLEMS

Obviously we would like to extend our 4-min-sum algorithm to more terminal pairs and more general arrangements of the terminals on the outer face. However, this seems to be difficult. A potentially more promising idea is to improve the approximation ratio of our approximation algorithm. We have described a $k$-approximation algorithm for the $k$-min-sum vertex-disjoint paths problem when all terminals are on a common face. For the most part, the algorithm is based on solving a linear program relaxation and then rounding the solution. We suspect that LP-based algorithms may actually lead to constant-factor approximations, perhaps even a 2-approximation. In this subsection, we give several pieces of circumstantial evidence for this.

**Parallel instances.** Assume that the optimal solution in the $k$-min-sum instance is unique. In this case, the problem reduces to a min-cost flow problem, as shown in Section 5.1. Solving this min-cost flow problem is in fact equivalent to solving the linear program relaxation that our approximation algorithm solves. Thus by omitting the rounding step our approximation algorithm solves parallel instances exactly. This is not a new result, since parallel instances can be solved faster using min-cost flow algorithms, but is an indication that LP-based algorithms can be useful for this problem.

**Serial instances.** We have already described a $(2d-2)$-approximation for $k$-min-sum; this is a 2-approximation for serial instances. Again, this is not a new result, since serial instances can be solved faster using the algorithm of Borradaile, Nayyeri, and Zafarani [62], but is an indication that LP-based algorithms can be useful for this problem.

**Integrality gap.** The integrality gap of a minimization problem is defined to be the ratio of the minimum of the integer program to the minimum of the linear program relaxation. Intuitively, a low integrality gap (i.e., a gap close to 1) means that the relaxation captures the original integer program well, so solving the relaxation gives a solution close to that of the original problem. We do not know what the integrality gap of $\mathcal{I}$ is, but Figure 5.11 shows that it is at least 4/3. On the other hand, we have been unable to come up with instances with higher gaps. Note that our $k$-approximation is a 3/2-approximation for this instance.

**Conditional 2-approximation.** Consider the following algorithm. First, we solve the linear program relaxation. For each $i \in [k]$, we now have a set $\mathcal{P}_i$ of flow-paths whose values sum to 1. Each flow-path has a weight, and we round every weight to the nearest integer.

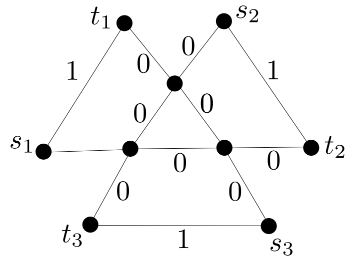


Figure 5.11: Our integrality gap instance. The optimal integer solution has value 2, while the optimal fractional solution has value 3/2 (every edge has weight 1/2)

Paths with weight exactly 1/2 do not get their weights rounded. Call this half-integral solution $\mathcal{H}$. Note that for each vertex $v$, at most one path going through $v$ gets its weight rounded up; furthermore, if a path going through $v$ gets its weight rounded up, then all other paths going through $v$ get their weights rounded down. It follows that at the end of the rounding procedure, the total weights of the paths going through $v$ is still at most 1, and so there are at most two paths going through $v$. Furthermore, each pair $i \in [k]$ has at most two paths connecting $s_i$ to $t_i$.

Pick an arbitrary leaf in the demand tree $T$ and root $T$ at this leaf. Re-index the terminal pairs such that $(s_i, t_i)$ is the root in $T$. We let $C_1$ be the portion of $\partial G$ between $s_1$ and $t_1$ and containing no other terminals, and for all $i > 1$, we let $C_i$ be the portion of $\partial G$ between $s_i$ and $t_i$ containing $C_1$. Given two paths $P$ and $Q$ with endpoints $s_i$ and $t_i$, we say that $P$ is *lower than* $Q$ if $Q$ is inside $C_i \circ Q$. Recall that each pair $i \in [k]$ has at most two paths connecting $s_i$ to $t_i$. For each $i$, we simply pick the lower of the two paths connecting $s_i$ to $t_i$; let $Q_i$ be the picked path. If there is only one path connecting $s_i$ to $t_i$, then we pick that path.

We claim that the picked paths are vertex-disjoint. Suppose $Q_i$ and $Q_j$ share a vertex $v$, where $e_j$ is a parent of $e_i$ in the demand tree. Then in the half-integral solution, there must have been paths of total weight 3/2 going through $v$, which is impossible.

We also claim that if the algorithm computes a feasible solution, then the solution is a 2-approximation. In the first rounding step (where we obtain a half-integral solution), we at most double the weight on every path, and weight of every path with weight 1/2 remains the same. In the second step (where we pick lower paths), we at most double the weight of paths with weight 1/2, but all other path weights remain the same or decrease. As a result, throughout the algorithm, the weight of any path at most doubles. This shows that the value of the solution the algorithm computes is at most twice the minimum of the linear program relaxation.

The only potential problem with this algorithm is that one of the sets of paths $\mathcal{P}_i$ may contain more than two paths. Then in the rounding step, the algorithm may round the weights of all the paths in $\mathcal{P}_i$ down. As a result, the solution returned by the algorithm will not have any paths connecting $s_i$ to $t_i$. Curiously, though, we have been unable to construct instances in which for some $i$, $\mathcal{P}_i$ must contain at least three paths, each with weight strictly less than 1/2. We conjecture that no such instances exist. If we assume this conjecture holds and $k$-min-sum instances have unique solutions, then we have a 2-approximation for the $k$-min-sum problem when all terminals are on the outer face.

# CHAPTER 6: ORIENTATION PROBLEMS

In this chapter, $G$ is an undirected graph, and $(s_1, t_1), \ldots, (s_k, t_k)$ are $k$ pairs of vertices (terminals) in $G$. We wish to find an orientation, an ideal orientation, or a $k$-min-sum orientation for $G$.

Ito et al. [72] suggest the following application of the orientation problem. Suppose we have to assign one-way restrictions to aisles in, say, an industrial factory, while maintaining reachability between several sites. This corresponds to the orientation problem. We may also want to maintain the distances of routes between the sites in order to keep transit time low and productivity high; this corresponds to the ideal orientation problem.

The orientation problem was first studied by Hassin and Megiddo [19], and they gave the following algorithm that works in general graphs. Without loss of generality, assume that $G$ is connected. First, compute the bridges of $G$. (A bridge is an edge whose removal would disconnect $G$). For each $i$, pick an arbitrary path from $s_i$ to $t_i$, and orient the bridges on this path in the direction that they appear on this path. If a bridge is forced to be oriented in both directions, then no orientation preserving reachability exists. Otherwise, such an orientation does exist: in the rest of $G$ each component is a 2-connected component and can be oriented to be strongly connected, by Robbins' theorem [73].

By contrast, much less is known about the ideal orientation problem and generalizations like the $k$-min-sum orientation problem. Hassin and Megiddo showed that the ideal orientation problem is polynomially-time solvable when $k = 2$ but is NP-hard for general $k$. Eilam-Tzoreff [12] extended Hassin and Megiddo's algorithm when $k = 2$ to find an ideal orientation minimizing the number of shared arcs in the paths realizing the distances in $H$. She also solved the generalization when $k = 2$ and we only require the shorter distance in $H$ to be a distance in $G$. The complexity of the ideal orientation problem for fixed $k > 2$ remains open.

Fenner, Lachish, and Popa [11] considered the min-sum orientation problem in general graphs when $k = 2$. They give a PTAS and reduce the 2-min-sum orientation problem to the 2-min-sum edge-disjoint paths problem. (In the 2-min-sum edge-disjoint paths problem, we need to find edge-disjoint paths from $s_1$ to $t_1$ and from $s_2$ to $t_2$ of minimum total length.) It remains unknown whether the 2-min-sum orientation problem or the 2-min-sum edge-disjoint paths problem can be solved in polynomial time. However, for unweighted graphs, Bjorklund and Husfeldt showed that the 2-min-sum edge-disjoint paths problem can be solved in polynomial time [74].

Ito et al. [72] considered the $k$-min-sum and $k$-min-max orientation problems. They proved

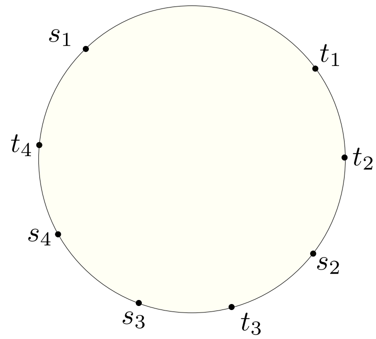


Figure 6.1: a serial instance where $k = 4$

that both problems are NP-hard in planar graphs, and that the $k$-min-sum orientation problem is solvable in $O(nk^2)$ time if $G$ is a cactus graph and $O(n + k^2)$ time if $G$ is a cycle. They showed that the $k$-min-max orientation problem is NP-hard in cacti, even when $k = 2$, but solvable in cycles in $O(n+k^2)$ time. For the $k$-min-max orientation problem, they also give a 2-approximation in cacti and a fully polynomial-time approximation scheme for fixed $k$ in cacti. It remains an open question whether $k$-min-sum or $k$-min-max orientation problems can be solved or approximated in classes of graphs more general than cacti.

In this chapter we present four results, three of which deal with the ideal orientation problem and one of which deals with the $k$-min-sum problem. First, we solve the ideal orientation problem for *serial* instances, even if $k$ is part of the input. An instance of any orientation problem is *serial* if the terminals are all on a single face in cyclic order $u_1, v_1, \ldots, u_k, v_k$, where for each $i$ we have either $(u_i, v_i) = (s_i, t_i)$ or $(u_i, v_i) = (t_i, s_i)$. See Figure 6.1. The algorithm is simple and relies on the fact that we can assume that the paths realizing the $s_i$-to-$t_i$ distances are pairwise noncrossing.

**Theorem 6.1.** We can solve any serial instance of the ideal orientation problem in $O(n \log n)$ time.

The algorithm uses Klein's algorithm for finding multiple-source shortest paths [75], which computes an implicit representation of the solution. If an explicit orientation is desired, then a solution takes $O(n^2)$ time to compute.

Second, we solve the ideal orientation problem in planar graphs for a fixed number of terminals when all terminals are on a single face and no terminal pairs cross. Two pairs of terminals $(s_i, t_i)$ and $(s_j, t_j)$ cross if all four terminals are on a common face and the cyclic order of the terminals is $s_i, s_j, t_i, t_j$. The algorithm relies on an algorithm of Schrijver that finds partially vertex-disjoint paths in directed planar graphs [34].

**Theorem 6.2.** If $k$ is fixed and all terminals are on the outer face and no terminals cross, then we can solve the ideal orientation problem in polynomial time.

The restriction that the terminals be noncrossing may seem arbitrary, but can be motivated in the following way. Recall that the demand graph $G_D$ is the graph with the same vertices as $G$ but with an edge $\{s_i, t_i\}$ for each $i$. Define $G + G_D$ to be the graph with the same vertices as $G$ (or $G_D$) and whose edge set is $E(G) \cup E(G_D)$. The case of noncrossing terminals is then exactly the case where $G + G_D$ is planar.

Third, we show that the ideal orientation problem is NP-hard in planar graphs. The reduction is from planar 3-SAT and is inspired by reductions by Middendorf and Pfeiffer [8] and by Eilam-Tzoreff [12], who showed that finding disjoint paths and disjoint shortest paths are NP-hard in planar graphs. Since the min-sum, min-max, and min-min orientation problem are all generalizations of the ideal orientation problem, this reduction shows that the min-sum, min-max, and min-min problems are also NP-hard. This is stronger than Ito et al.'s result because the ideal orientation problem is a special case of the $k$-min-sum orientation problem.

**Theorem 6.3.** If $k$ is part of the input, then the ideal orientation problem is NP-hard in unweighted planar graphs.

Fourth, we solve the $k$-min-sum orientation problem for serial instances. To do this, we classify each terminal pair as clockwise or counterclockwise, and we break up the instance into two sub-instances, one of which consists only of clockwise pairs and the other of which consists only of counterclockwise pairs. It turns out that solving each sub-instance reduces to solving serial instances of a shortest vertex-disjoint paths problem, which can be done using an algorithm of Borradaile, Nayyeri, and Zafarani [62]. Finally, after solving the two sub-instances independently, we show that the two sub-solutions can be easily combined to solve the original instance.

**Theorem 6.4.** Any serial instance of the $k$-min-sum orientation problem can be solved in $O(kn^5)$ time.

Our algorithms search for pairwise nonconflicting directed walks that are shortest paths connecting corresponding terminals, rather than explicitly seeking simple paths. Because all edge lengths are positive, the set of shortest walks will end up consisting of simple paths. Note that given a directed walk $P$ that conflicts with itself, we can repeatedly remove directed cycles from $P$ to obtain a simple directed path $P'$ such that $P'$ has the same starting and ending vertices as $P$, $P'$ is no longer than $P$, and $P'$ does not conflict with itself. Thus we do not have to worry about directed walks conflicting with themselves.

We assume without loss of generality that the paths in any solution do not use edges on the outer face. If necessary to enforce this assumption, we can connect the terminals using an outer cycle of $2k$ infinite-weight edges. We also assume that the $2k$ terminals are all distinct. (If two terminals, say $s_i$ and $s_j$, are not distinct, then we add new terminals $s'_i$ and $s'_j$ that will be terminals instead of $s_i$ and $s_j$, respectively, and we add new arcs $s'_i s_i$ and $s'_j s_j$. If $s_i$ and $s_j$ were on a common face then we can ensure $s'_i$ and $s'_j$ still are.)

This chapter is organized as follows. In section 6.1, we prove various structural results. In section 6.2 we prove Theorem 6.1, in section 6.3 we prove Theorem 6.2, in section 6.4 we prove Theorem 6.3, and in section 6.5 we prove Theorem 6.4.

## 6.1 STRUCTURE

Let $a, b, c$, and $d$ be four vertices on the outer face of $G$. Let $P$ be a directed walk from $a$ to $b$ and let $Q$ be a directed walk from $c$ to $d$. Walks $P$ and $Q$ are *opposite* if the cyclic order of their four endpoints around $\partial G$ is $a, b, c, d$. $P$ and $Q$ are *parallel* if the order is $a, b, d, c$, and we denote this by $P \sim Q$. We define each path to be parallel to itself. Note that if $P$ is parallel to $Q$, then $Q$ is parallel to $P$. We have the following two lemmas.

**Lemma 6.1.** Suppose $G$ has positive edge weights, and suppose $P$ and $Q$ are opposite nonconflicting shortest paths. If a vertex $x$ precedes a vertex $y$ on $P$, then $x$ does not precede $y$ in $Q$. In particular, $P$ and $Q$ are edge-disjoint.

*Proof.* Suppose for the sake of argument that $P$ and $Q$ are opposite nonconflicting shortest paths, and vertex $x$ precedes vertex $y$ on both $P$ and $Q$. By the Jordan curve theorem, there exists a vertex $z$ on $P \cap Q$ such that either $z$ precedes $x$ on $Q$ and $y$ precedes $z$ on $P$, or $y$ precedes $z$ on $Q$ and $z$ precedes $x$ on $P$. Suppose the first case holds. See Figure 5.1a. Since $P$ and $Q$ are shortest paths, we have

$$\ell(P[z,x]) = \ell(Q[x,z]) = \ell(Q[x,y]) + \ell(Q[y,z]) \tag{6.1}$$

$$\text{and } \ell(P[z,x]) + \ell(P[x,y]) = \ell(P[z,y]) = \ell(Q[y,z]). \tag{6.2}$$

This is impossible because $\ell(P[x,y]) = \ell(Q[x,y]) > 0$.

Now suppose the second case holds. See Figure 5.1b. Similar to the first case, we have

$$\ell(P[y,z]) = \ell(Q[z,y]) = \ell(Q[z,x]) + \ell(Q[x,y]) \tag{6.3}$$

$$\text{and } \ell(P[x,y] + \ell(P[y,z]) = \ell(P[x,z]) = \ell(Q[z,x]), \tag{6.4}$$

which is impossible because $\ell(Q[x,y]) = \ell(P[x,y]) > 0$.

QED.

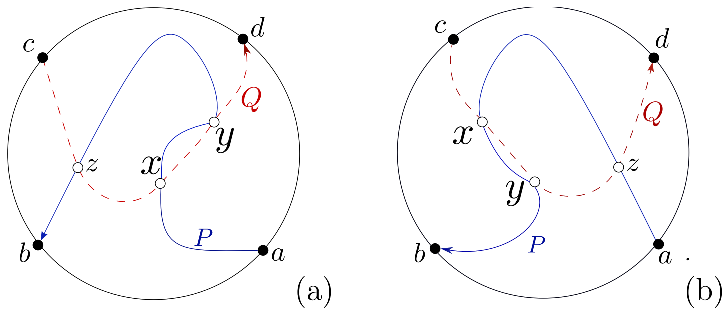


Figure 6.2: Impossible configurations in the proof of Lemma 6.1. The solid blue path is $P$ and the dashed red path is $Q$ (a) $z \prec_Q x$ and $y \prec_P z$ (b) $z \prec_P x$ and $y \prec_Q z$

**Lemma 6.2.** Suppose $G$ has positive edge weights, and suppose $P$ and $Q$ are parallel shortest paths. If a vertex $x$ precedes a vertex $y$ in $P$, then $y$ does not precede $x$ in $Q$. In particular, $P$ and $Q$ do not conflict.

*Proof.* This is just Lemma 6.1 with $Q$ replaced by $\text{rev}(Q)$. QED.

This lemma immediately suggests an algorithm for a special case of the ideal orientation problem. Suppose $G$ is an instance where the terminals all appear on the outer face in clockwise order $s_1, \ldots, s_k, t_k, \ldots, t_1$. Lemma 6.2 implies that the shortest paths from $s_i$ to $t_i$ are nonconflicting, so we just need to find a shortest path from $s_i$ to $t_i$ for all $i$. Steiger [76] showed how to find a representation of these paths in $O(n \log\log k)$ time.

The following two lemmas are trivial when shortest paths are unique. In the ideal orientation problem, we cannot assume that shortest paths are unique because then the problem becomes trivial: just find the (unique) shortest paths and check if they conflict.

**Lemma 6.3.** Let $G$ be any planar instance of the ideal orientation problem with terminal pairs $(s_1, t_1)$, …, $(s_k, t_k)$. If a solution $\mathcal{P}$ exists, then a solution $\mathcal{P}'$ exists in which for every pair of parallel paths $P_i$ and $P_j$ in $\mathcal{P}$, $P_i$ and $P_j$ are noncrossing.

*Proof.* This was proved by Liang and Lu [77]. Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a solution to the ideal orientation problem.

Suppose $P_i$ and $P_j$ are parallel paths that cross each other. Let $x$ be the first vertex of $P_i$ on $P_j$ and let $y$ be the last. see Figure 6.3.

Now we construct two alternate solutions $\mathcal{P}'$ and $\mathcal{P}''$ to the ideal orientation problem. To construct $\mathcal{P}'$, we exchange $P_i[x, y]$ for $P_j[x, y]$. In other words, let $P_i' = P_i[s_i, x] \circ P_j[x, y] \circ P_i[y, t_i]$, and let $\mathcal{P}' = \mathcal{P} \setminus \{P_i\} \cup \{P_i'\}$. Since $P_i[x, y]$ and $P_j[x, y]$ are shortest paths, $P_i'$ is still a shortest path connecting $s_i$ to $t_i$. Since $P_i'$ only uses arcs in $P_i$ and $P_j$, $P_i'$ does not conflict with any other path in $\mathcal{P}'$. Thus $\mathcal{P}'$ is another set of $k$ nonconflicting shortest paths. Furthermore, paths $P_j'$ and $P_i'$ are noncrossing.

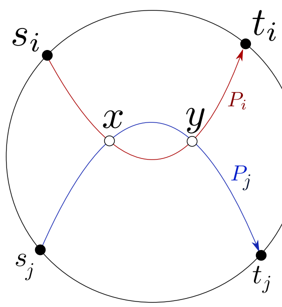


Figure 6.3: Uncrossing $P_i$ and $P_j$: replace $P_i$ with $P_i[s_i, x] \circ P_j[x, y] \circ P_i[y, t_i]$ or replace $P_j$ with $P_j[s_j, x] \circ P_i[x, y] \circ P_j[y, t_j$

To construct $\mathcal{P}''$, we exchange $P_j[x, y]$ for $P_i[x, y]$. In other words, let $P_j'' = P_j[s_j, x] \circ P_i[x, y] \circ P_j[y, t_i]$, and let $\mathcal{P}' = \mathcal{P} \setminus \{P_j\} \cup \{P_j''\}$. Since $P_i[x, y]$ and $P_j[x, y]$ are shortest paths, $P_j''$ is still a shortest path connecting $s_i$ to $t_i$. Since $P_i''$ only uses arcs in $P_i$ and $P_j$, $P_i''$ does not conflict with any other path in $\mathcal{P}''$. Thus $\mathcal{P}''$ is another set of $k$ nonconflicting shortest paths. Furthermore, paths $P_j''$ and $P_i''$ are noncrossing.

Let $\text{cr}(\mathcal{P})$ be the number of pairs of paths in $\mathcal{P}$ that cross each other. We have

$$2\text{cr}(\mathcal{P}) \geq (\text{cr}(\mathcal{P}') + 1) + (\text{cr}(\mathcal{P}'') + 1) \tag{6.5}$$

where the two instances of "+1" on the right side come from the fact that $P_j'$ and $P_i'$ do not cross, and $P_j''$ and $P_i''$ do not cross. Thus at least one of $\mathcal{P}'$ and $\mathcal{P}''$ have strictly fewer pairs of crossing paths than $\mathcal{P}$. We have thus found a way to reduce the number of pairs of crossing paths while maintaining optimality.

The exchange procedure strictly reduces the number of parallel paths that cross each other, so we can keep repeating the procedure until no two parallel paths cross each other. QED.

**Lemma 6.4.** Let $G$ be any planar instance of the ideal orientation problem with noncrossing terminal pairs $(s_1, t_1)$, ..., $(s_k, t_k)$. If there exists a solution $\mathcal{P}$ in which parallel paths do not cross, then there exists a solution $\mathcal{P}'$ such that (1) parallel paths do not cross; (2) for every pair of parallel paths $P_i$ and $P_j$ in $\mathcal{P}'$, $P_i \cap P_j$ is connected; and (3) the number of crossings between paths of $\mathcal{P}'$ is no more than the number of crossings between paths of $\mathcal{P}$.

*Proof.* Suppose there exists a solution $\mathcal{P} = \{P_1, \ldots, P_k\}$ in which parallel paths do not cross. We show how to make property (2) hold while ensuring that parallel paths remain noncrossing and the total number of crossings between opposite paths does not increase.

Suppose $P_i$ and $P_j$ are parallel paths such that $P_i \cap P_{i+1}$ consists of at least two paths. There exist vertices $x$ and $y$ on $P_i \cap P_j$ such that $P_i[x, y]$ and $P_j[x, y]$ intersect only at $x$ and $y$. Let $\mathcal{I}$ be the set of paths in $\mathcal{P}$ that contain $P_i[x, y]$ as a subpath, and let $\mathcal{J}$ be the set of paths in $\mathcal{P}$ that contain $P_j[x,y]$. Note that $P_i \in \mathcal{I}$ and $P_j \in \mathcal{J}$. The region $B_{ij}[x,y]$, bounded by $P_i[x,y]$ and $P_j[x,y]$, is a bigon. We will assume without loss of generality that the bigon $B_{ij}[x,y]$ is minimal, in the sense that no other bigon is contained in $B_{ij}[x,y]$. Note that this ensures that no path parallel to any of the paths in $\mathcal{I}$ or $\mathcal{J}$ enters $B_{ij}[x,y]$, by Lemma 6.1 and our assumption that parallel paths in $\mathcal{P}$ do not cross. This implies that any path that crosses $P_i[x,y]$ or $P_j[x,y]$ must be opposite to all paths in $\mathcal{I}$ and $\mathcal{J}$. There are two cases:

1. Suppose $P_i[x,y]$ crosses at least as many paths as $P_j[x,y]$ does. Then we exchange $P_i[x,y]$ for $P_j[x,y]$, as follows. For each path $P_p \in \mathcal{I}$, let $P'_p = P_p[s_p,x] \circ P_j[x,y] \circ P_p[y,t_p]$, and let $\mathcal{P}' = \mathcal{P} \setminus \mathcal{I} \cup \{P'_p | P_p \in \mathcal{I}\}$. Since $P_j[x,y]$ is a shortest path, $P'_p$ is a shortest path from $s_p$ to $t_p$ for any $P_p \in \mathcal{I}$. Since no paths parallel to $P_p$ enter the interior of $B_{ij}[x,y]$, $P'_p$ does not cross any paths parallel to it, for any $P_p \in \mathcal{I}$. Finally, for any $P_p \in \mathcal{P}$, the number of paths opposite to $P_p$ or $P_j$ that $P'_p$ crosses is at most the number of paths opposite to $P_p$ or $P_j$ that $P_p$ crosses, since $P_j[x,y]$ crosses at most as many paths as $P_i[x,y]$ does.

2. Suppose $P_i[x,y]$ crosses fewer paths than $P_j[x,y]$. Then we can exchange $P_j[x,y]$ for $P_i[x,y]$. That is, for any path $P_p \in \mathcal{J}$, define $P'_p = P_p[s_p,x] \circ P_i[x,y] \circ P_p[y,t_p]$, and let $\mathcal{P}' = \mathcal{P} \setminus \mathcal{J} \cup \{P'_p | P_p \in \mathcal{J}\}$. Analogous to the previous case, one can show that the number of crossings does not increase and that parallel paths still do not cross. Furthermore, the resulting solution is still made up of shortest paths and is thus still a solution.

As long as there exist two parallel paths whose intersection is not a single subpath, we can perform the exchange. Each time we perform the exchange, the number of crossings does not increase, parallel paths are still pairwise noncrossing, and the sum of the number of components of $P_i \cap P_j$ decreases, where the sum is taken over all pairs of parallel paths $P_i$ and $P_j$. Thus the procedure will terminate. QED.

## 6.2 SERIAL CASE FOR IDEAL ORIENTATIONS

Recall that an instance of the ideal orientation problem is *serial* if the terminals all appear on the outer face in clockwise order $u_1, v_1, \ldots, u_k, v_k$, where for each $i \in [k]$ we have $(u_i, v_i) = (s_i, t_i)$ or $(u_i, v_i) = (t_i, s_i)$. For all $i$, if $(u_i, v_i) = (s_i, t_i)$, then we say that $(s_i, t_i)$ and any path from $s_i$ to $t_i$ are *clockwise*; otherwise $(s_i, t_i)$ and any path from $s_i$ to $t_i$ are *counterclockwise*. Note that a clockwise and a counterclockwise path are parallel, while two clockwise paths

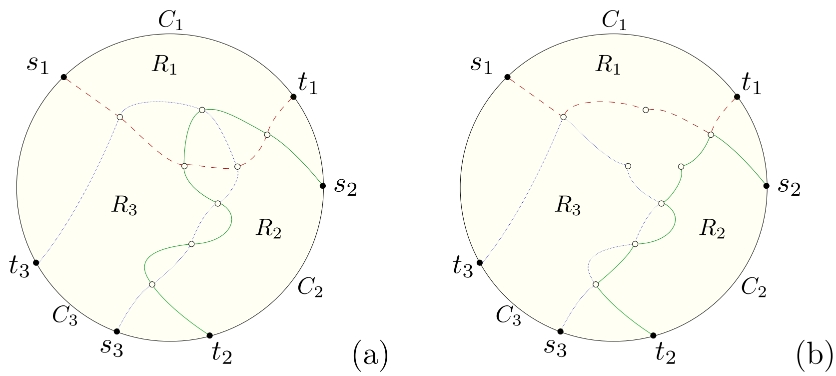


Figure 6.4: All paths are directed from $s_i$ to $t_i$. (a) We have $\Pi = \{\pi_1, \pi_2, \pi_3\}$, where the dashed red path is $\pi_1$, solid green path is $\pi_2$, and dotted blue path is $\pi_3$ (b) The dashed red path is $L(1, \Pi)$, solid green path is $L(2, \Pi)$, and dotted blue path is $L(3, \Pi)$

(or two counterclockwise paths) are opposite. In this section, we describe an algorithm that solves serial instances of the ideal orientation problem in $O(n^2)$ time even when $k$ is part of the input. First we prove the following lemmas:

### 6.2.1 Envelopes

Suppose $G$ is a serial instance with terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$. Suppose we have a set $\Pi$ of arbitrary directed paths $\pi_1, \ldots, \pi_k$ such that $\pi_i$ connects $s_i$ to $t_i$ and no path touches $\partial G$; the paths may intersect arbitrarily. Let $C_i$ be the portion of $\partial G$ that connects $s_i$ to $t_i$ without containing any other terminals. The paths in $\pi_1, \ldots, \pi_k$ divide the interior of $G$ into connected regions. Let $R_i$ be the unique region with $C_i$ on its boundary. Finally, we define $L(i, \Pi)$ to be the directed path from $s_i$ to $t_i$ whose set of edges is $\partial R_i \setminus C_i$. Intuitively, $L(i, \Pi)$ is the "lower envelope" of $\pi_1, \ldots, \pi_k$ if we draw $G$ such that $s_i$ and $t_i$ are on the bottom. See Figure 6.4

For any arc $e$ in $L_i$, either $e$ or $\text{rev}(e)$ must be an arc in one of $\pi_1, \ldots, \pi_k$. Thus $L(i, \Pi)$ does not contain any edges on $\partial G$. Also, the walks $L(1, \Pi), \ldots, L(k, \Pi)$ are pairwise noncrossing.

**Lemma 6.5.** Suppose $G$ is serial and $\Pi = \{\pi_1, \ldots, \pi_k\}$ is a set of paths such that $\pi_i$ connects terminal $s_i$ to terminal $t_i$ for all $i$. If $\pi_1, \ldots, \pi_k$ are pairwise nonconflicting, then $L(1, \Pi), \ldots, L(k, \Pi)$ are also pairwise nonconflicting.

*Proof.* We prove the contrapositive. Suppose $L(i, \Pi)$ and $L(j, \Pi)$ conflict at an edge $e$. Then the regions $R_i$ and $R_j$ touch each other at $e$. Since $\pi_i$ and $\pi_j$ do not use any boundary arcs, the Jordan Curve Theorem implies that $\pi_i$ and $\pi_j$ also conflict at $e$. QED.

**Lemma 6.6.** Suppose $G$ is serial and $\Pi = \{\pi_1, \ldots, \pi_k\}$ is a set of pairwise nonconflicting paths such that $\pi_i$ connects terminal $s_i$ to terminal $t_i$ for all $i$. Then we have

$$\sum_{i=1}^{k} \ell(\pi_i) \geq \sum_{i=1}^{k} \ell(L(i, \Pi)). \tag{6.6}$$

In particular, if $\pi_1, \ldots, \pi_k$ are shortest paths, then so are $L(1, \Pi), \ldots, L(k, \Pi)$.

*Proof.* Because the walks $L(1, \Pi), \ldots, L(k, \Pi)$ are pairwise noncrossing, the Jordan Curve Theorem implies that each arc $e$ can be used by at most one of the walks $L(1, \Pi), \ldots, L(k, \Pi)$. By Lemma 6.5, arc $\text{rev}(e)$ can only be used by one of the walks $L(1, \Pi), \ldots, L(k, \Pi)$ if $e$ is not used by any of those walks. It follows that each edge of $G$ is used by at most one of the walks $L(1, \Pi), \ldots, L(k, \Pi)$. In addition, every edge used by one of $L(1, \Pi), \ldots, L(k, \Pi)$ must be used by at least one of $\pi_1, \ldots, \pi_k$. The lemma follows. QED.

### 6.2.2 Algorithm

**Lemma 6.7.** Let $G$ be a serial instance of the ideal orientation problem with terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$. If a solution exists, then a solution exists in which the paths $P_1, \ldots, P_k$ are pairwise noncrossing.

*Proof.* This is straightforward using envelopes. Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a solution to the serial instance $G$ of the ideal orientation problem, where $P_i$ connects $s_i$ to $t_i$. Then the walks $L(1, \mathcal{P}), \ldots, L(k, \mathcal{P})$ are pairwise noncrossing. By Lemma 6.5, the walks are pairwise nonconflicting, and by Lemma 5.2 they are shortest paths, so they constitute a solution.
QED.

A path from $s_i$ to $t_i$ is the *outermost* shortest path from $s_i$ to $t_i$ if it is outside all other shortest paths from $s_i$ to $t_i$. The following lemma states that finding outermost shortest paths is sufficient:

**Lemma 6.8.** Let $G$ be a serial instance of the ideal orientation problem with terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$. If a solution $\mathcal{P} = \{P_1, \ldots, P_k\}$ exists, then a solution exists in which $P_i$ is the outermost shortest path from $s_i$ to $t_i$ for all $i$.

*Proof.* Suppose we have a solution $\mathcal{P} = \{P_1, \ldots, P_k\}$ where the paths in $\mathcal{P}$ are pairwise noncrossing and some path $P_i$ is not the outermost shortest path from $s_i$ to $t_i$. Then we can exchange $P_i$ for the outermost shortest path. That is, let $P_i'$ be the outermost shortest path from $s_i$ to $t_i$, and let $\mathcal{P}' = \mathcal{P} \setminus \{P_i\} \cup \{P_i'\}$. The new path $P_i'$ is in $R_i$ so it does not cross with any other path in $\mathcal{P}'$; in particular, $P_i'$ does not conflict with any other path in $\mathcal{P}'$, so $\mathcal{P}'$ is a solution. We can keep doing this exchange until we get a solution where every path is the outermost shortest path. QED.

So the algorithm is to find all outermost shortest paths. If the outermost paths conflict, then there is no solution. Computing the outermost paths explicitly takes $O(n)$ time per path and thus $O(n^2)$ time [31]. Alternatively, Klein's algorithm computes an implicit representation of the paths in $O(n \log n)$ time [75].

## 6.3 FIXED NUMBER OF TERMINALS AND NONCROSSING PAIRS

In this section we describe an algorithm to solve the ideal orientation problem in planar graphs where all terminals are on a single face, the number of terminals is fixed, and none of the terminal pairs cross. (Recall that two terminal pairs $(s_i, t_i)$ and $(s_j, t_j)$ cross if the four terminals are on a common face and their cyclic order on that face is $s_i, s_j, t_i, t_j$.) For simplicity, call such instances one-face noncrossing instances. We will first show that the number of crossings between the paths in a solution is a function bounded only by $k$. This allows us to guess all crossing points and then reduce the problem to the partially noncrossing edge-disjoint paths problem (PNEPP). As described in section 2.3.3, this reduces to the partially vertex-disjoint paths problem, which has been solved by Schrijver [34]. The algorithm is inspired by a result of Bérczi and Kobayashi [78].

We saw in the previous section that in the serial instances of the ideal orientation problem we can assume that the paths in the solution are noncrossing. This is unfortunately not true for general one-face noncrossing instances. See Figure 6.5a for an example. On the other hand, we can prove that the number of crossings is small when $k$ is small. Specifically, we have the following lemma:

**Lemma 6.9.** Suppose $G$ is a one-face noncrossing instance of the ideal orientation problem with terminal pairs $(s_1, t_1)$, …, $(s_k, t_k)$. If a solution exists, then a solution $\{P_1, \ldots, P_k\}$ exists in which for all $i$ and $j$, path $P_i$ crosses $P_j$ a total of $O(k)$ times.

We will prove this lemma at the end of the section. For now, we will just assume the lemma is true and describe the algorithm for one-face noncrossing instances. Let $G$ be a one-face noncrossing instance of the ideal orientation problem with terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$. Suppose $\mathcal{P} = \{P_1, \ldots, P_k\}$ is a solution with the fewest crossings. Our algorithm first guesses all the crossing points; by Lemma 6.9, there are $O(k^3)$ such points.

Next, inspired by Erickson and Nayyeri [10], we define the overlay graph $H_{\mathcal{P}}$, whose vertices are the crossing points and terminals, and whose edges are the subwalks between

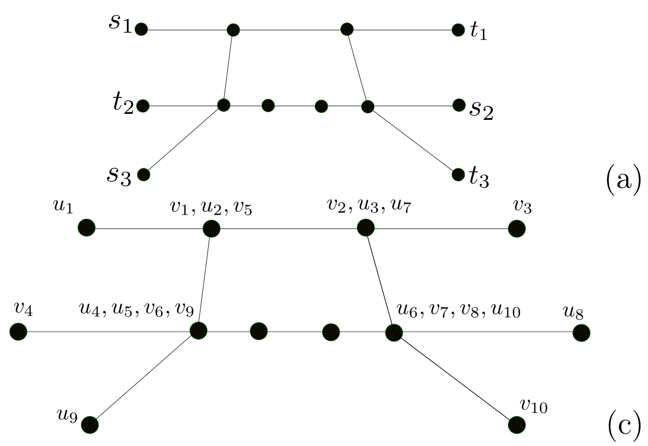


Figure 6.5: (a) An instance of the ideal orientation problem where the unique solution has the path from $s_2$ to $t_2$ crossing the path from $s_3$ to $t_3$. All edges have unit weight. (b) overlay graph corresponding to the unique solution (c) instance of PNEPP corresponding to the overlay graph. Here $S = \{\{i, j\} | i \in \{1, 2, 3, 5, 7, 9, 10\}, j \in \{4, 6, 8\}\}$

consecutive crossing points and terminals. The graph $H_{\mathcal{P}}$ has a natural embedding. Our algorithm guesses the overlay graph. Given the set of $O(k^3)$ crossing points, there are $2^{O(k^6)}$ possible such graphs. See Figure 6.5b.

The $O(k^3)$ crossing points split up $P_1, \ldots, P_k$ into pairwise noncrossing directed subpaths. By Lemma 6.1, subpaths of opposite paths are edge-disjoint (there is no analogous restriction for parallel paths). Furthermore, every directed subpath is a shortest path between its endpoints. Let $\{p_1, \ldots, p_\beta\}$ be the set of these directed subpaths.

To compute these subpaths, we construct an instance of PNEPP as follows. The directed graph $H$ is just $G$ where every undirected edge $\{u, v\}$ is replaced with two arcs $uv$ and $vu$. The terminal pairs in $G$ are no longer terminal pairs in $H$. For each subpath $p_i$ in $\mathcal{P}$, we construct a pair of terminals $u_i, v_i$ that are just the endpoints of $p_i$. Thus the constructed terminals will not necessarily be distinct. The set $S$ consists of all pairs $\{i, j\}$ such that $p_i$ and $p_j$ are subpaths of opposite paths in $G$. For all $i \in \{1, \ldots, \beta\}$, the subgraph $H_i$ is the union of all shortest paths from $u_i$ to $v_i$ in $H$. Each $H_i$ is a directed acyclic graph.

We then solve the instance $H$ of PNEPP to find subpaths $Q_i$ connecting the $u_i$ to the $v_i$, and we check if the concatenations of the appropriate found subpaths are indeed shortest paths connecting corresponding terminals in $G$. (In Figure 6.5c, we would need to check, for example, that the concatenation of $Q_1$, $Q_2$, and $Q_3$ is indeed a shortest path from $s_1 = u_1$ to $t_1 = v_3$.) If the concatenations are indeed shortest paths in $G$, then they form a solution to the instance $G$ of the ideal orientation problem. Clearly, if we take any solution of $G$, the subpaths formed by the crossing points are noncrossing and nonconflicting. The following lemma implies that the algorithm is correct.

**Lemma 6.10.** Let $G$ be a one-face noncrossing instance of the ideal orientation problem. The following statements are equivalent

- There exists a solution to $G$ with crossing points $v_1, \ldots, v_h$ and overlay graph $H_{\mathcal{P}}$.
- There exist crossing points $v_1, \ldots, v_h$, an overlay graph whose vertices are the crossing points and the terminals of $G$, and a set of shortest noncrossing partially edge-disjoint subpaths connecting the crossing points in accordance with the overlay graph, such that the paths formed from concatenating the appropriate subpaths are shortest paths. (Here when we say that two sub-paths are partially edge-disjoint we mean that they are edge-disjoint if they correspond to subpaths of opposite paths in the overlay graph.)

*Proof.* $\Rightarrow$: Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a solution to the instance $G$ of the ideal orientation problem. Split the paths in $\mathcal{P}$ into subpaths using the crossing points. The subpaths are noncrossing by construction. We just need to show that the subpaths are partially disjoint. In fact we will show that the paths in $\mathcal{P}$ are partially disjoint. If $P_i$ and $P_j$ are parallel, then there is nothing to prove. If $P_i$ and $P_j$ are opposite, then they are edge-disjoint by Lemma 6.1.

$\Leftarrow$: Concatenate the subpaths and assume the concatenations are shortest paths. Since the subpaths are noncrossing, We just need to show that they are nonconflicting. Suppose $P_1, \ldots, P_k$ are the resulting paths after concatenation. If $P_i$ and $P_j$ are parallel, then by Lemma 6.2 they are nonconflicting. If $P_i$ and $P_j$ are opposite, then by construction each subpath of $P_i$ is edge-disjoint from each subpath of $P_j$. This means that $P_i$ and $P_j$ are edge-disjoint, so they don't conflict. QED.

To summarize, we first guess the crossing points, then guess an overlay graph on these crossing points and the original terminals, and finally use the overlay graph to construct and solve an instance of PNEPP. The number of possible sets of crossing points and overlay graphs depends only on $k$, while PNEPP can be solved in polynomial time for fixed $k$ (equivalently, fixed $\beta$) by reducing to PVPP and using Schrijver's algorithm [34]. Furthermore, constructing the instance of PNEPP takes polynomial time. Thus, our algorithm runs in polynomial time for fixed $k$.

### 6.3.1 The crossing bound

In this subsection we prove Lemma 6.9. Suppose $\mathcal{P} = \{P_1, \ldots, P_k\}$ is a solution to a one-face noncrossing instance $G$ of the ideal orientation problem, where $P_i$ connects $s_i$ to $t_i$. By Lemmas 6.3 and 6.4, we may assume that for every pair of parallel paths in $\mathcal{P}$, the paths

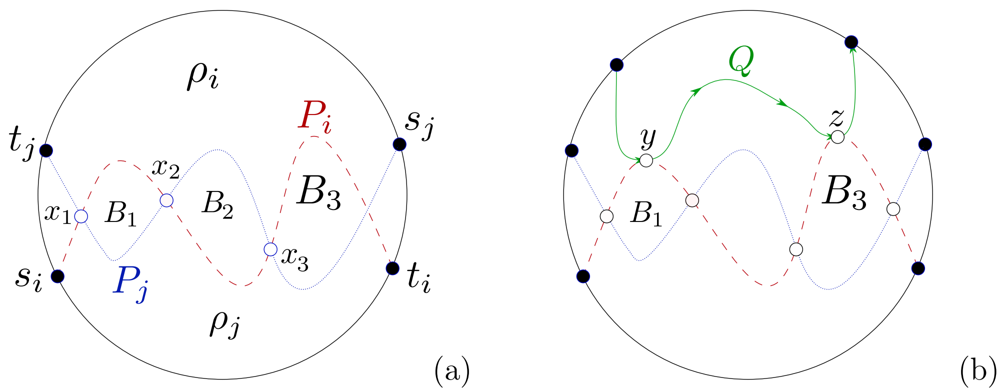


Figure 6.6: The dashed red path is $P_i$ and the dotted blue path is $P_j$. (a) Three bigons $B_1$, $B_2$, and $B_3$ formed by $P_i$ and $P_j$. (b) An impossible configuration in the proof of Lemma 6.11. Here $Q$ is the solid green path, $p = 1$, and $q = 3$

do not cross and their intersection consists of at most one subpath; of all such solutions, we may assume without loss of generality that $\mathcal{P}$ is the solution with the fewest crossings. It suffices to show that for all $i, j$, path $P_i$ crosses path $P_j$ at most $2k$ times.

Let $h$ be the number of times $P_i$ and $P_j$ cross; we want to show that $h \leq 2k$. Since terminal pairs $(s_i, t_i)$ and $(s_j, t_j)$ are noncrossing, $h$ is even. Parallel paths in $\mathcal{P}$ do not cross, so assume that $P_i$ and $P_j$ are opposite. The path $P_i$ divides the interior of $G$ into two regions; let $\rho_i$ be the region containing $s_j$ and $t_j$, and define a path to be *above* $P_i$ if it lies in $\rho_i$. Likewise, the path $P_j$ divides the interior of $G$ into two regions; let $\rho_j$ be the region containing $s_i$ and $t_i$, and define a path to be *below* $P_j$ if it lies in $\rho_j$. Let $x_1, \ldots, x_h$ be the vertices at which $P_i$ and $P_j$ cross, in order along $P_i$; by Lemma 6.1, this is exactly the reverse of their order along $P_j$. Split the region $\rho_i \cap \rho_j$ into $h - 1$ pairwise internally disjoint *bigons*, denoted by $B_1, \ldots, B_{h-1}$; the bigon $B_p$ consists of the region bounded by the two subpaths $P_i[x_p, x_{p+1}]$ and $P_j[x_{p+1}, x_p]$. Note that under our definition, $P_i[x_p, x_{p+1}]$ and $P_j[x_{p+1}, x_p]$ may touch but they may not cross. A bigon $B_p$ is *odd* if $p$ is odd and *even* if $p$ is even. Note that any odd bigon is below $P_i$ and above $P_j$, and any even bigon is below $P_j$ and above $P_i$. See Figure 6.6a.

For any vertex $x$, let $pred_i(x)$ denote the predecessor of $x$ on $P_i$, $succ_i(x)$ denote the successor of $x$ on $P_i$, $pred_j(x)$ denote the predecessor of $x$ on $P_j$, and $succ_j(x)$ denote the successor of $x$ on $P_j$. Suppose a path $Q$ is parallel to $P_i$. Path $Q$ and a bigon $B_p$ *partially overlap* each other if $Q$ shares edges with $P_i[x_p, x_{p+1}]$ and $Q$ does not contain $P_i[pred_i(x_p), succ_i(x_{p+1})]$. Likewise, suppose a path $Q'$ is parallel to $P_j$. Path $Q'$ and a bigon $B_p$ *partially overlap* each other if $Q$ shares edges with $P_j[x_{p+1}, x_p]$ but $Q$ does not contain $P_j[pred_j(x_{p+1}), succ_j(x_p)]$.

For the rest of this subsection we will say "overlap" when we mean "partially overlap."

Lemma 6.9 follows if we can prove the following two lemmas:

**Lemma 6.11.** Each path in $\mathcal{P}$ overlaps at most two different bigons.

**Lemma 6.12.** Each bigon overlaps some path (different bigons could overlap different paths).

Lemmas 6.11 and 6.12 together imply that there must be at most $2(k-1)$ bigons, and thus at most $2k$ crossings between $P_i$ and $P_j$.

*Proof of Lemma 6.11.* There are three cases. For the first case, suppose $Q$ is a path parallel to $P_i$ that overlaps some bigons formed by $P_i$ and $P_j$. We will show that $Q$ overlaps at most two bigons. See Figure 6.6b. Let $B_p$ be the first bigon along $P_i$ that $Q$ overlaps and let $B_q$ be the last, so that $Q$ contains some vertex $y$ on $P_i[x_p, x_{p+1}]$ and some vertex $z$ on $P_i[x_q, x_{q+1}]$. We have assumed that the intersection of any two parallel paths in $\mathcal{P}$ consists of exactly one subpath. Since $P_i$ and $Q$ are parallel, this implies that $Q$ contains the subpath $P_i[y, z]$. In particular, $Q$ contains each of $P_i[x_{p+1}, x_{p+2}], \ldots, P_i[x_{q-1}, x_q]$, so $Q$ does not overlap any of the bigons $B_{p+1}, \ldots, B_{q-1}$. It follows that $Q$ overlaps at most two bigons.

For the second case, a symmetric argument shows that if $Q$ is parallel to $P_j$ then $Q$ overlaps at most two bigons. For the third case, if $Q$ is opposite to both $P_i$ and $P_j$, then by Lemma 6.1, $Q$ is edge-disjoint from both $P_i$ and $P_j$, and so does not overlap any bigons. QED.

*Proof of Lemma 6.12.* Suppose for the sake of argument that $B_p$ is an odd bigon that does not overlap any path. The bigon $B_p$ is below $P_i$ and above $P_j$. Specifically, it is bounded by $P_i[x_p, x_{p+1}] \cup P_j[x_{p+1}, x_p]$. To lighten notation, let $A = P_i[x_p, x_{p+1}]$ and let $B = P_j[x_{p+1}, x_p]$. Our goal is to reduce the number of crossings in $\mathcal{P}$ via an exchange procedure similar to those used in previous lemmas. Roughly speaking, we will do this by reversing the orientations of $A$ and $B$ and by modifying the paths that enter $B_p$ so that they no longer do so.

First we describe how to reverse the orientations of $A$ and $B$. By assumption, all paths in $\mathcal{P}$ that use edges in $A$ must contain $A$, and all paths in $\mathcal{P}$ that use edges in $B$ must contain $B$. Let $\mathcal{Q}_L$ be the set of paths that contain $A$ and let $\mathcal{Q}_R$ be the set of paths that contain $B$. By Lemma 6.1, all paths in $\mathcal{Q}_L$ are parallel to $P_i$ and all paths in $\mathcal{Q}_R$ are parallel to $P_j$. Now we simply let

$$P_l' = P_l[s_l, x_p] \circ \mathrm{rev}(B) \circ P_l[x_{p+1}, t_l] \tag{6.7}$$

for any path $P_l \in \mathcal{Q}_l$, and let

$$P_r' = P_r[s_r, x_{p+1}] \circ \mathrm{rev}(A) \circ P_r[x_p, t_r] \tag{6.8}$$

for any path $P_r \in \mathcal{Q}_R$. Let $\mathcal{Q}'_L = \{P'_l | P_l \in \mathcal{Q}_L\}$ and $\mathcal{Q}'_R = \{P'_r | P_r \in \mathcal{Q}_R\}$. Note that $P_i \in \mathcal{Q}_L$ and $P_j \in \mathcal{Q}_R$, so we have described how to modify $P_i$ and $P_j$.

Now we describe how to modify the paths that enter $B_p$. This is necessary so that the paths do not cross with the paths $P'_l$ and $P'_r$ described in the previous paragraph. By Lemma 6.1, $A$ only crosses paths parallel to $P_j$, and $B$ only crosses paths parallel to $P_i$. Let $\mathcal{Q}_A$ be the set of paths that cross $A$, and let $\mathcal{Q}_B$ be the set of paths that cross $B$. For each path $P_a \in \mathcal{Q}_A$, let $u_a$ be the first vertex (of $P_a$) at which $P_a$ touches $A$ and let $v_a$ be the last. We define

$$P'_a = P_a[s_a, u_a] \circ A[u_a, v_a] \circ P_a[v_a, t_a]. \tag{6.9}$$

Note that $P'_a$ conflicts with $A$ but does not conflict with rev$(A)$. Furthermore, $P'_a$ no longer crosses $A$. Similarly, for each path $P_b \in \mathcal{Q}_B$, let $u_b$ be the first vertex (of $P_b$) at which $P_b$ crosses $B$, let $v_b$ be the last vertex at which $P_b$ crosses $B$, and let

$$P'_b = P_b[s_b, u_b] \circ B[u_b, v_b] \circ P_b[v_b, t_b]. \tag{6.10}$$

Let $\mathcal{Q}'_A = \{P'_a | P_a \in \mathcal{Q}_A\}$ and $\mathcal{Q}'_B = \{P'_b | P_b \in \mathcal{Q}_B\}$. This finishes the description of how to modify $\mathcal{P}$ to reduce the number of crossings. That is, let

$$\mathcal{P}' = \mathcal{P} \setminus (\mathcal{Q}_L \cup \mathcal{Q}_R \cup \mathcal{Q}_A \cup \mathcal{Q}_B) \cup (\mathcal{Q}'_L \cup \mathcal{Q}'_R \cup \mathcal{Q}'_A \cup \mathcal{Q}'_B). \tag{6.11}$$

We need to show that $\mathcal{P}'$ is a solution with fewer crossings than $\mathcal{P}$. Paths $A$ and $B$ are shortest paths, so all subpaths of $A$ and $B$ are shortest paths and all paths in $\mathcal{P}'$ are shortest paths. All paths in $\mathcal{P}'$ use the edges of $A$ in the reverse direction (i.e., from $x_{p+1}$ to $x_p$), if at all. Similarly, all paths in $\mathcal{P}'$ use the edges of $B$ in the reverse direction (i.e., from $x_p$ to $x_{p+1}$), if at all. All arcs used by $\mathcal{P}'$ that are not used by $\mathcal{P}$ are in $A$ or $B$, so this implies that the paths in $\mathcal{P}'$ are nonconflicting. During the exchange procedure, we replace subpaths in or on $B_p$ with subpaths of the boundary of $B_p$, such that no paths in $\mathcal{P}'$ enter $B_p$. Tedious casework implies that no crossings are added when we go from $\mathcal{P}$ to $\mathcal{P}'$; for details, see the proof of Lemma 6.13. Without increasing the number of crossings in $\mathcal{P}'$, we can also use the procedure in the proof of Lemma 6.4 to modify the paths in $\mathcal{P}'$ so that the intersection of any pair of parallel paths consists of a single subpath. On the other hand, given any pair of paths $P_l \in \mathcal{Q}_L$ and $P_r \in \mathcal{Q}_R$, $P'_l$ and $P'_r$ have strictly fewer crossings than $P_l$ and $P_r$; for details, see the proof of Lemma 6.13 again. This contradicts the fact that $\mathcal{P}$ has the fewest crossings out of all solutions that satisfy Lemma 6.4. We have thus proved the lemma for odd bigons. A symmetric argument proves the lemma for even bigons. QED.

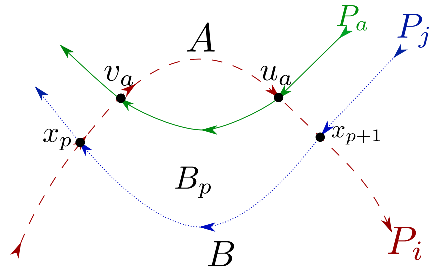

Figure 6.7: Paths before the exchange procedure, in the proof of Lemma 6.12. The dashed red path is $P_i$, the dotted blue path is $P_j$, and the solid green path is $P_a$. $B_p$ is bounded by $A$ and $B$

**Lemma 6.13.** In the proof of Lemma 6.12, the paths in $\mathcal{P}'$ have no more crossings than the paths in $\mathcal{P}$.

*Proof.* We need to extend the definitions of "below" and "above" introduced in subsection 6.3.1. Suppose $P$ and $Q$ are paths in $G$ whose endpoints are on $\partial G$. Suppose further that the endpoints of any two of $P, Q$, and $P_i$ do not cross. Let $C_i$ be the portion of $\partial G$ from $s_i$ to $t_i$ that does not contain $s_j$ or $t_j$. There are two cases.

1. Suppose the endpoints of $Q$ are not in $C_i$. The path $Q$ divides the interior of $G$ into two regions. If $P$ lies entirely in the region whose closure contains $s_i$ and $t_i$, then $P$ is below $Q$.

2. Suppose the endpoints of $Q$ are in $C_i$. The path $Q$ divides the interior of $G$ into two regions. If $P$ lies entirely in the region whose closure does not contain $s_j$ and $t_j$, then $P$ is below $Q$.

Now let $P$ and $Q$ be paths in $\mathcal{P}$. To simplify notation, let $P' = P$ if $P \notin \mathcal{Q}_L \cup \mathcal{Q}_R \cup \mathcal{Q}_A \cup \mathcal{Q}_B$, so that $\mathcal{P}' = \{p' | p \in \mathcal{P}\}$.

First we will show that $P'$ and $Q'$ do not cross more times than $P$ and $Q$ cross. There are eight different cases (not counting symmetric cases). For the first four cases, suppose $P$ and $Q$ are both parallel to $P_j$, so that $P$ and $Q$ do not cross by Lemma 6.3:

1. Suppose $P \notin \mathcal{Q}_L \cup \mathcal{Q}_R \cup \mathcal{Q}_A \cup \mathcal{Q}_B$. We have $P' = P$. By Lemma 6.3, none of the edges of $P'$ are in $B_p$ or on $A$. By Lemma 6.1, none of the edges of $P'$ are in $B$, so none of the edges of $P$ are on the boundary of $B_p$. On the other hand, $Q' \bigoplus Q$ consists only of edges in $B_p$ or on its boundary. It follows that if $P$ is below $Q$, then $P'$ is below $Q'$. Similarly, if $P$ is above $Q$, then $P'$ is above $Q'$. In both cases, $P'$ and $Q'$ do not cross.

2. Suppose $P, Q \in \mathcal{Q}_L$. In both $P$ and $Q$ we replace $A$ with $\text{rev}(B)$ to get $P'$ and $Q'$. It follows that if $P$ is below $Q$, then $P'$ is below $Q'$. Similarly, if $P$ is above $Q$, then $P'$ is above $Q'$. In both cases $P'$ and $Q'$ do not cross.

3. Suppose $P \in \mathcal{Q}_L, Q \in \mathcal{Q}_B$. In $P$, we replace $A$ with $\text{rev}(B)$ to get $P'$. On the other hand, $Q$ must be below $P$. Since $B$ is below $P$ and $Q' \setminus Q$ consists of edges in $B$, we see that $Q'$ is below $P$ as well. By construction, $Q'$ is also on or below $B$, so $Q'$ is below $P'$ and does not cross it.

4. Suppose $P, Q \in \mathcal{Q}_B$, and suppose without loss of generality that $P$ is below $Q$. Let $u$ be the first vertex at which $P$ crosses $B$ and let $v$ be the last. Let $s_P$ and $t_P$ be the endpoints of $P$. Then $P[s_P, u]$ and $P[v, t_P]$ are below $Q'$, so $P'$ is below $Q'$.

For the remaining four cases, suppose $P$ is left-to-right but $Q$ is right-to-left. We need to show that $P'$ and $Q'$ do not cross each other more than $P$ and $Q$ cross each other:

5. Suppose $P \notin \mathcal{Q}_L \cup \mathcal{Q}_R \cup \mathcal{Q}_A \cup \mathcal{Q}_B$. We have $P' = P$. As in case 1, none of the edges of $P$ are in $B_p$ or on the boundary of $B_p$. On the other hand, $Q' \bigoplus Q$ consists only of edges in $B_p$ or on its boundary. It follows that $P'$ and $Q'$ do not cross each other more than $P$ and $Q$ do.

6. Suppose $P \in \mathcal{Q}_L$ and $Q \in \mathcal{Q}_R$. Path $P$ replaces $A$ with $\text{rev}(B)$ to get $P'$, and $Q$ replaces $B$ with $\text{rev}(A)$ to get $Q'$. Path $P$ contains $A$, so Lemma 6.1 implies that $P$ does not cross $B$ and so does not enter the interior of $B_p$. Similarly, $Q$ contains $B$ but does not enter the interior of $B_p$. This means that when we replace $P$ and $Q$ with $P'$ and $Q'$, the only vertices that could become crossing points or stop being crossing points are $x_p$ and $x_{p+1}$. But in fact $P$ contains $P_i[pred_i(x_p), succ_i(x_{p+1})] \supsetneq A$ and $Q$ contains $P_j[pred_j(x_{p+1}), succ_j(x_p)] \supsetneq B$, so both $x_p$ and $x_{p+1}$ are points at which $P$ and $Q$ cross and $P'$ and $Q'$ do not cross. Furthermore, no new crossings are added when we replace $P$ and $Q$ with $P'$ and $Q'$.

7. Suppose $P \in \mathcal{Q}_L, Q \in \mathcal{Q}_A$. Note that $Q$ is above $P_j$, and so is $A \supset Q' \setminus Q$, so $Q'$ is above $P_j$. On the other hand, $P' \setminus P$ consists of edges in $\text{rev}(B)$, which is a subpath of $P_j$. Thus no new crossings are added when we replace $P$ and $Q$ with $P'$ and $Q'$.

8. Suppose $P \in \mathcal{Q}_B, Q \in \mathcal{Q}_A$. As in the previous case, $Q$ and $Q'$ are above $P_j$, On the other hand, $P' \setminus P$ consists of edges in $\text{rev}(B)$, which is a subpath of $P_j$. Thus no new crossings are added when we replace $P$ and $Q$ with $P'$ and $Q'$.

All other cases are symmetric to these eight cases. Note that in case 6, $P'$ and $Q'$ cross each other fewer times than $P$ and $Q$, which is part of what we wanted to show. QED.

## 6.4 NP-HARDNESS OF IDEAL ORIENTATIONS

In this section we show that the ideal orientation problem is NP-hard in unweighted planar graphs when $k$ is part of the input. The reduction is from planar 3-SAT and is similar to reductions by Middendorf and Pfeiffer [8] and by Eilam-Tzoreff [12]. Planar 3-SAT is the special case of 3-SAT where a certain bipartite graph $G(y)$ is planar, defined as follows. Given an instance $y$ of 3-SAT, each variable of $y$ is a vertex, and each clause of $y$ is also a vertex. For every variable $x_i$ and every clause $c_j$, we add an edge between $x_i$ and $c_j$ if either $x_i$ or $\overline{x_i}$ appears in $c_j$. The resulting graph $G(y)$ is bipartite; if it is planar, then $y$ is an instance of planar 3-SAT. Lichtenstein showed that planar 3-SAT is still NP-hard [79].

Suppose we are given an instance $y$ of planar 3-SAT. As noted by Middendorf and Pfeiffer [8], we may assume that each variable appears in three clauses. To see this, fix a planar embedding of $G(y)$, and let $vC_1, \ldots, vC_k$ be the edges incident to a variable $v$ in clockwise order. Introduce new variables $v_1, \ldots, v_k$ and clauses $v_k \vee \neg v_1$ and $v_i \vee \neg v_{i+1}$ for all $i \in \{1, \ldots, k-1\}$. In addition, replace the occurrence of $v$ in $C_i$ with $v_i$. If we do this for all variables $v$, we get an instance $y'$ of planar 3-SAT that is satisfiable if and only if $y$ is satisfiable, and every variable in $y'$ appears in exactly three clauses.

We use $y$ to construct an instance of the ideal orientation problem. We will construct a clause gadget for each clauses and a variable gadget for each variable. The clause gadget for a clause $C$ is shown in Figure 6.8. There are three terminals pairs $(s_C, t_C), (s'_C, t'_C)$, and $(s''_C, t''_C)$. Let us note some key properties of $G_C$. We have $d(s_C, t_C) = d(s''_C, t''_C) = 3$ and $d(s'_C, t'_C) = 4$. There are two shortest paths from $s_C$ to $t_C$, three shortest paths from $s'_C$ to $t'_C$, and two shortest paths from $s''_C$ to $t''_C$. There exist pairwise nonconflicting shortest paths connecting $(s_C, t_C), (s'_C, t'_C)$, and $(s''_C, t''_C)$ in $G_C$. These paths must use at least one of the edges $ab = e_{vC}$, $cd = e_{wC}$, and $ef = e_{xC}$. Furthermore, three such nonconflicting shortest paths exist even when two of the three edges are not to be used.

The edges $e_{vC}$, $e_{wC}$, and $exC$ are part of the clause gadget associated with $C$ and will also each be in variable gadgets associated with $v$, $w$, and $x$, respectively. Before defining the variable gadgets, we need to fix some terminology regarding the orientations of the three edges. Each of the three edges can be oriented *forward* or *backward* as follows. The forward orientation of $e_{vC}$ is from $a$ to $b$, the forward orientation of $e_{wC}$ is from $c$ to $d$, and the forward orientation of $e_{xC}$ is from $e$ to $f$. The backward orientation of an edge is simply the reverse of the forward orientation. Intuitively, an edge must be oriented forward in order to be used in some shortest path connecting a pair of terminals; furthermore, orienting an edge $e_{vC}$ forward means that the literal $v$ or $\neg v$ (whichever one appears in $C$) is set to True.

We also give each of the three edges a *true* and a *false* orientation depending on whether

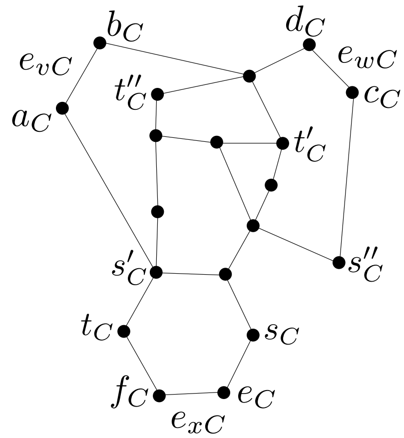


Figure 6.8: Clause gadget $G_C$ for a clause $C$ containing variables $v$, $w$, $x$. All edges are unweighted.

the literals in $C$ are positive or negative. If $v$ is a literal in the clause $C$, then the true orientation of $e_{vC}$ is the forward orientation of $e_{vC}$ and the false orientation of $e_{vC}$ is the backward orientation. If $\neg v$ is a literal in $C$, then the true orientation of $e_{vC}$ is the backward orientation and the false orientation is the forward orientation. True and false orientations for $e_{wC}$ and $e_{xC}$ are defined analogously. Intuitively, the true orientation of an edge $e_{vC}$ is the direction that it would be oriented in if the variable $v$ were assigned to true.

Finally, each of the three edges has a *clockwise* orientation and a *counterclockwise* orientation. The clockwise orientation of $e_{vC}$ is its forward orientation, the clockwise orientation of $e_{xC}$ is its forward orientation, and the clockwise orientation of $e_{wC}$ is its *backward* orientation. The counterclockwise orientation of an edge is the reverse of its clockwise orientation. Intuitively, an edge oriented clockwise goes clockwise around its clause gadget, and an edge oriented counterclockwise goes counterclockwise around its clause gadget. However, somewhat confusingly, we will construct the variable gadgets such that a clockwise-oriented edge goes *counterclockwise* around its *variable* gadget and a counterclockwise-oriented edge goes *clockwise* around its *variable* gadget.

For each variable $v$, we construct a variable gadget $G_v$ as follows. Suppose $v$ appears in clauses $C, D$, and $E$; suppose further that $vC, vD$, and $vE$ are the edges incident to $v$ in clockwise order in $G(y)$. For each of the three edges $e_{vC}$, $e_{vD}$, and $e_{vE}$ (in the clause gadgets), we check whether or not the true orientation of the edge is the counterclockwise orientation of that edge. There are four cases:

- If for each of the three edges the true orientation is the counterclockwise orientation, then we construct the variable gadget in Figure 6.9a.

- If for exactly two of the three edges (without loss of generality, $(v, C)$ and $(v, D)$) the

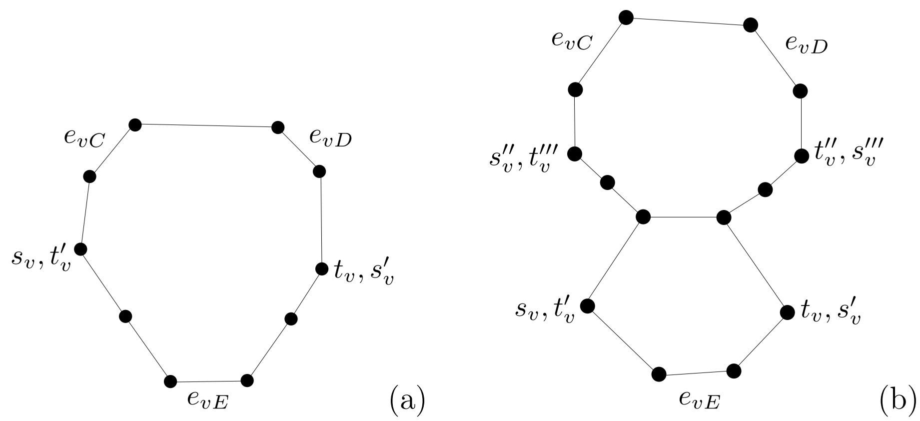


Figure 6.9: possible variable gadgets $G_v$ for a variable $v$ appearing in three clauses $C, D$, and $E$

true orientation is the counterclockwise orientation, then we construct the variable gadget in Figure 6.9b.

- If for exactly one of the three edges (without loss of generality, $(v, E)$) the true orientation is the counterclockwise orientation, then again we still construct the variable gadget in Figure 6.9b.

- If for each of the three edges the true orientation is the clockwise orientation, then we still construct the variable gadget in Figure 6.9a.

Finally, for every variable $v$ and every clause $C$ we identify the edge $e_{vC}$ in both $G_v$ and $G_C$. The resulting graph is still planar and is $G_1(y)$.

$G_v$ is constructed so that there are only two ways to orient the edges. In one orientation, all edges $vC, vD$, and $vE$ are oriented in the true direction, and in the other orientation, the three edges are oriented in the false direction. Orienting the three edges in the true direction corresponds to setting the variable to True, and orienting them in the false direction corresponds to setting them to False. The reduction clearly takes polynomial time, and the following lemma implies its correctness.

**Lemma 6.14.** A planar 3-SAT formula $y$ is satisfiable if and only if there exists an ideal orientation in $G_1(y)$.

*Proof.* $\underline{\Rightarrow:}$ Suppose $y$ is satisfiable, and fix a satisfying assignment. For each clause $C$, we orient the edges in $G_C$ as follows. For each of the three literals, we do the following. Let $v$ or $\neg v$ be some literal in $C$. Orient the edge $e_{vC}$ forwards if $v$ or $\neg v$ is in $C$ and set to True; otherwise, the edge is oriented backwards. We know that exactly one of the three edges

$e_{vC}, e_{wC}, e_{xC}$ is oriented forwards. It is possible to orient the rest of the edges in $G_C$ such that distances between the terminal pairs $(s_C, t_C), (s'_C, t'_C)$, and $(s''_C, t''_C)$ are preserved.

In each variable gadgets $G_v$, we orient the edges as follows. If $v$ is set to True, then each of $e_{vC}, e_{vD}, e_{vE}$ are oriented in the true direction; otherwise, the three edges are oriented in the false direction. It is possible to orient the rest of the edges in $G_v$ such that the distances between the terminal pairs $(s_v, t_v), (s'_v, t'_v$, $(s''_v, t''_v)$, and $(s'''_v, t'''_v)$ (if they exist) are preserved. To show that orientations are consistent, recall that in the clause gadgets, there are four cases:

- $v$ appears in $C$ and is set to True. Then $e_{vC}$ is oriented forward and so is oriented in the true direction.

- $\neg v$ appears in $C$ and $v$ is set to False. Then $e_{vC}$ is oriented forward and in the false direction.

- $v$ appears in $C$ and is set to False. Then $e_{vC}$ is oriented backward and so is- oriented in the false direction.

- $\neg v$ appears in $C$ and $v$ is set to True. Then $e_{vC}$ is oriented backward and so is oriented in the true direction.

In all cases we see that the orientation in the clause gadget is consistent with the orientation in the variable gadget. $\Leftarrow$: Suppose an ideal orientation exists. If $e_{vC}, e_{vD}$, and $e_{vE}$ are all oriented in the true direction, then set $v$ to True; otherwise they are all oriented in the false direction and we set $v$ to False. We need to show that this is a satisfying assignment. Consider a clause $C$. Since an ideal orientation exists, at least one of the edges $e_{vC}$, $e_{wC}$, and $e_{xC}$ must be oriented forward. Say $e_{vC}$ is oriented forward. This means that either $e_{vC}$ is oriented in the true direction with $v$ appearing positively, or $e_{vC}$ is oriented in the false direction with $v$ appearing negatively. In the first case, $v$ is set to True, so $C$ is satisfied. In the second case, $v$ is set to False, so $C$ is satisfied. QED.

## 6.5 SERIAL CASE FOR K-MIN-SUM ORIENTATIONS

In this section, we describe an algorithm to solve serial instances of the $k$-min-sum orientation problem. Recall that every terminal pair $(s_i, t_i)$ is either clockwise (i.e., a clockwise traversal of the outer face will visit $s_i$ and then immediately visit $t_i$) or counterclockwise. Given a set $\Pi$ of arbitrary directed paths $\pi_1, \ldots, \pi_k$ such that $\pi_i$ connects $s_i$ to $t_i$, we define "lower envelopes" $L(1, \Pi), \ldots, L(k, \Pi)$ in the same way as in section 6.2. To simplify our presentation, we assume that our given instance has a unique solution, if it exists. If necessary, this uniqueness assumptions can be enforced with high probability using the isolation lemma of Mulmuley, Vazirani, and Vazirani [33].

Before we describe the algorithm, we prove an analog of Lemma 6.7.

**Lemma 6.15.** Let $G$ be a serial instance of the $k$-min-sum orientation problem with terminal pairs $(s_1, t_1)$, ..., $(s_k, t_k)$. If a solution exists, then the paths in the solution are pairwise noncrossing.

*Proof.* The proof is similar to that of Lemma 6.7. Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be the unique solution to the serial instance $G$ of the $k$-min-sum orientation problem, where $P_i$ connects $s_i$ to $t_i$. The walks $L(1, \mathcal{P}), \ldots, L(k, \mathcal{P})$ are pairwise noncrossing. By Lemma 6.5 they are pairwise nonconflicting. By Lemma 5.2 and our uniqueness assumption, their total length is strictly less than that of the paths in $\mathcal{P}$. This contradicts the fact that $\mathcal{P}$ was the optimal solution to $G$. QED.

Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be the unique solution to the instance $G$ of the $k$-min-sum solution. By the Jordan Curve Theorem, noncrossing opposite paths must be edge-disjoint. This suggests the following algorithm, which occurs in two phases.

1. In the first phase, we re-index the terminals so that $(s_1, t_1), \ldots, (s_\alpha, t_\alpha)$ are clockwise and $(s_{\alpha+1}, t_{\alpha+1}), \ldots, (s_k, t_k)$ are counterclockwise. We split the instance of the $k$-min-sum problem into two sub-instances. One of the sub-instances consists of the original graph $G$ with the clockwise terminal pairs, while the other sub-instance consists of $G$ with the counterclockwise terminal pairs. We solve each sub-instance separately. In the clockwise sub-instance, we are finding $\alpha$ noncrossing edge-disjoint directed paths of minimum total length such that the $i$-th path connects $s_i$ to $t_i$ for $i \in \{1, \ldots, \alpha\}$ (We will describe later how to find such paths). Likewise, in the counterclockwise sub-instance we are finding $k - \alpha$ noncrossing edge-disjoint directed paths of minimum total length such that the $(i - \alpha)$-th path connects $s_i$ to $t_i$ for $i \in \{\alpha + 1, \ldots, k\}$. We then let $\Pi = \{\pi_1, \ldots, \pi_k\}$ be the set of all $k$ paths, where $\pi_i$ connects $s_i$ to $t_i$. By Lemma 6.15, the sum of the lengths of $\pi_1, \ldots, \pi_k$ is at most the sum of the lengths of the paths in $\mathcal{P}$.

2. Any two opposite paths in $\Pi$ are edge-disjoint and so are nonconflicting. However, parallel paths (i.e., a clockwise path and a counterclockwise path) found by the first phase may conflict with each other; the purpose of phase 2 is to remove these conflicts. In phase 2, we simply output $L(1, \Pi), \ldots, L(k, \Pi)$. By Lemma 6.15, the sum of the

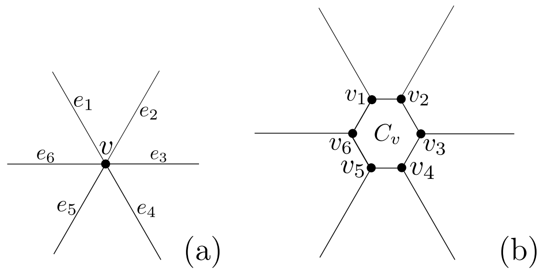


Figure 6.10: (a) vertex $v \in V(G)$ with incident edges $e_1, \ldots, e_6$ (b) corresponding cycle $C_v$ in $G^\circ$; each edge in $C_v$ has zero length

lengths of the output paths is no greater than the sum of the lengths of the paths in $\mathcal{P}$. Since the output paths are envelopes, they noncrossing; the Jordan Curve Theorem then implies that two output paths can conflict only if they are opposite. On the other hand, Lemma 6.5 implies that opposite paths are nonconflicting. Thus the output paths are indeed nonconflicting paths of minimum total length that connect the terminals.

To finish the description of the algorithm we just need to show how to find the noncrossing edge-disjoint directed paths in Phase 1. Before doing this, we define the *$k$-min-sum noncrossing edge-disjoint paths problem ($k$-NEPP)* and the *$k$-min-sum vertex-disjoint paths problem ($k$-VPP)* as follows. In $k$-NEPP we are given a plane graph $G$ with $k$ pairs of terminals $(s_1, t_1), \ldots, (s_k, t_k)$, and we wish to find $k$ paths $P_1, \ldots, P_k$ such that $P_i$ connects $s_i$ to $t_i$ and the $k$ paths are pairwise noncrossing and edge-disjoint. (Note that under our definition of "edge-disjoint," finding edge-disjoint directed paths in undirected graphs is the same as finding edge-disjoint undirected paths in undirected graphs. Thus for the rest of this section all paths will be undirected.) $k$-VPP is similar except that the paths $P_1, \ldots, P_k$ are to be vertex-disjoint instead of noncrossing edge-disjoint. It is known that $k$-VPP can be solved in serial instances in $O(kn^5)$ time when edge lengths are non-negative [62].

In order to find the paths in Phase 1 we need to solve serial instances of $k$-NEPP. We will solve such instances by reducing to serial instances of $k$-VPP; this will finish the description of the algorithm for serial instances of the $k$-min-sum orientation problem.

The reduction is as follows. Starting with $G$, we replace each vertex $v$ in $G$ with an undirected cycle $C_v$ of $\deg(v)$ vertices $v_1, \ldots, v_{\deg(v)}$. Each edge in the cycle has length zero. We make every edge that was incident to $v$ incident to some vertex $v_i$ instead, such that each edge is connected to a different vertex $v_i$, the clockwise order of the edges is preserved, and the graph remains planar. The resulting graph $G^\circ$ has $O(n)$ vertices and arcs. See Figure 6.10. Furthermore, if $G$ has all terminals on the outer face, then so does $G^\circ$.

**Lemma 6.16.** Suppose $G$ is serial instance with terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$. The following statements are equivalent:

1. There exist pairwise noncrossing edge-disjoint paths $P_1, \ldots, P_k$ of total length $L$ in $G$ such that $P_i$ connects $s_i$ and $t_i$ for all $i$.

2. There exist pairwise vertex-disjoint paths $Q_1, \ldots, Q_k$ of total length $L$ in $G^\circ$ such that $Q_i$ connects $s_i$ and $t_i$ for all $i$.

*Proof.* $\Rightarrow$: Suppose there exist pairwise noncrossing edge-disjoint paths $P_1, \ldots, P_k$ of total length $L$ in $G$ such that $P_i$ connects $s_i$ and $t_i$. We construct the paths $Q_1, \ldots, Q_k$ as follows. For any edge $e$ in $P_i$, we add $e$ to $Q_i$. This defines the portions of the paths $Q_1, \ldots, Q_k$ outside the cycles $C_v$; these portions are vertex-disjoint because by construction the endpoints of edges of $G$ are all distinct in $G^\circ$.

We route the portions of $Q_1, \ldots, Q_k$ inside the cycles $C_v$ in $G^\circ$ as follows. Let $v$ be a vertex of $G$, and suppose $P_i$ go through $v$. Suppose the cyclic order of the edges around $v$ is $e_1, \ldots, e_d$, where $d = \deg(v)$. Say $P_i$ goes into $v$ through $e_x$ and leaves through $e_y$, where $x < y$. By the Jordan Curve Theorem, either no other path uses $e_{x+1}, \ldots, e_{y-1}$ or no other path uses $e_{y+1}, \ldots, e_d, e_1, \ldots, e_{x-1}$. Suppose the first case holds (the second case is symmetric). Route the path $Q_i$ through vertices $v_x, \ldots, v_y$. The resulting paths $Q_1, \ldots, Q_k$ are vertex-disjoint from because none of the paths $P_1, \ldots, P_k$ use $e_{x+1}, \ldots, e_{y-1}$ (except possibly $P_i$). Clearly $Q_1, \ldots, Q_k$ have the same length as $P_1, \ldots, P_k$.

$\Leftarrow$: Suppose there exist pairwise vertex-disjoint paths $Q_1, \ldots, Q_k$ of total length $L$ in $G^\circ$. Trivially, the paths $Q_1, \ldots, Q_k$ are pairwise noncrossing edge-disjoint too. Each path $P_i$ can be defined by "projecting" $Q_i$ into $G$ in the obvious way: an edge of $G$ is in $P_i$ if and only if $e$ was in the original path $Q_i$. The resulting paths $P_1, \ldots, P_k$ are pairwise noncrossing because the original paths $Q_1, \ldots, Q_k$ were pairwise noncrossing. By similar reasoning as in the second half of the proof of Lemma 2.8, we can see that the paths $P_1, \ldots, P_k$ are pairwise noncrossing and edge-disjoint. Clearly $P_1, \ldots, P_k$ are the same length as $Q_1, \ldots, Q_k$. QED.

We can use the algorithm of Borradaile, Nayyeri, and Zafarani [62] to solve serial instances of $k$-min-sum vertex-disjoint paths. Since $G^\circ$ has $k$ pairs of terminals and $O(n)$ vertices and edges, the algorithm of Borradaile, Nayyeri, and Zafarani still takes $O(kn^5)$ time to compute $\Pi$. Given $\Pi$, computing the envelopes $L(1, \Pi), \ldots, L(k, \Pi)$ takes $O(n)$ time, so our entire algorithm still takes $O(kn^5)$ time.

## 6.6 OPEN PROBLEMS

Very little is known about the ideal orientation problem or the $k$-min-sum orientation problem, and so there are many open problems. Here we list three that we find particularly interesting. First, we do not know whether or not the ideal orientation problem can be solved in planar graphs when all terminals are on a single face (and we allow some pairs to cross). In fact, this problem is open even if $k = 3$. Second, we do not know if the ideal orientation problem in planar graphs can be solved when for each $i \in \{1, \ldots, k\}$, $s_i$ and $t_i$ are on the same face (different pairs could be on different faces), and no two pairs on the same face cross each other. Again, this is open even if $k = 3$. If $k$ is fixed, then we may be able to solve this problem using an algorithm similar to the one in Section 6.3. Finally, we do not know if the $k$-min-sum problem can be solved in outerplanar graphs, even if $k = 2$.

## CHAPTER 7: REFERENCES


[1] K. Jain, I. Măndoiu, V. V. Vazirani, and D. P. Williamson, "A primal-dual schema based approximation algorithm for the element connectivity problem," in *Proc. 10th Ann. ACM-SIAM Symp. Discrete Algorithms - SODA '99*, 1999, pp. 484–489.

[2] C. Chekuri and N. Korula, "A graph reduction step preserving element-connectivity and packing Steiner trees and forests," *SIAM Journal on Discrete Mathematics*, vol. 28(2), pp. 577–597, 2014.

[3] C. Chekuri, T. Rukkanchanunt, and C. Xu, "On element-connectivity preserving graph simplification," in *Algorithms ESA 2015*, 2015, pp. 313–324.

[4] K. Menger, "Zur allgemeinen kurventheorie," *Fund. Math.*, vol. 10:96-115, 1927.

[5] L. Ford and D. Fulkerson, *Flows in Networks.* Princeton, NJ: Princeton University Press, 1962.

[6] T. E. Harris and F. S. Ross, "Fundamentals of a method for evaluating rail net capacities," *Research Memorandum*, 1955.

[7] R. K. Ahuja, T. L. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[8] M. Middendorf and F. Pfeiffer, "On the complexity of the disjoint paths problem," *Combinatorica*, vol. 13(1), pp. 97–107, 1993.

[9] Y. Kobayashi and C. Sommer, "On shortest disjoint paths in planar graphs," *Discrete Optimization*, vol. 7(4), pp. 234–245, 2010.

[10] J. Erickson and A. Nayyeri, "Shortest non-crossing walks in the plane," in *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, 2011, pp. 279–308.

[11] T. Fenner, O. Lachish, and A. Popa, "Min-sum 2-paths problems," in *Approximation and Online Algorithms: 11th International Workshop*, 2013, pp. 1–11.

[12] T. Eilam-Tzoreff, "The disjoint shortest paths problem," *Discrete Applied Mathematics*, vol. 85(2), pp. 113–138, 1998.

[13] E. Colin de Verdière and A. Schrijver, "Shortest vertex-disjoint two-face paths in planar graphs," *ACM Trans. Algorithms*, vol. 7, no. 2, pp. 19:1–19:12, 2011.

[14] J. Chuzhoy, D. H. K. Kim, and R. Nimavat, "Improved Approximation for Node-Disjoint Paths in Grids with Sources on the Boundary," in *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, Eds., vol. 107. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/9042 pp. 38:1–38:14.

[15] A. Frank, "Packing paths, cuts and circuits - a survey," in *Paths, Flows, and VLSI-Layout.* Heidelberg: Springer, 1990, pp. 49–100.

[16] N. Robertson and P. D. Seymour, "An outline of a disjoint paths algorithm," in *Paths, Flows, and VLSI-Layout.* Heidelberg: Springer, 1990, pp. 267–292.

[17] R. G. Ogier, V. Rutenburg, and N. Shacham, "Distributed algorithms for computing shortest pairs of disjoint paths," *IEEE Trans. Inf. Theory*, vol. 39, no. 2, pp. 443–455, 1993.

[18] A. Srinivas and E. Modiano, "Finding minimum energy disjoint paths in wireless ad-hoc networks," *Wireless Networks*, vol. 11(4), pp. 401–417, 2005.

[19] R. Hassin and N. Megiddo, "On orientations and shortest paths," *Linear Algebra Appl.*, pp. 114–115, 589–602, 1989.

[20] J. B. Orlin, "Max flows in $O(nm)$ time, or better," in *Proceedings of the 45th Annual ACM Symposium on the Theory of Computing*, 2013, pp. 765–774.

[21] S. Fortune, J. Hopcroft, and J. Wyllie, "The directed subgraph homeomorphism problem," *Theoret. Comput. Sci.*, vol. 10, no. 2, pp. 111–121, 1980.

[22] H. Kaplan and Y. Nussbaum, "Maximum flow in directed planar graphs with vertex capacities," *Algorithmica*, vol. 61(1), pp. 174–189, 2011.

[23] G. Borradaile, P. Klein, S. Mozes, Y. Nussbaum, and C. Wulff-Nilsen, "Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time," *SIAM Journal on Computing (SICOMP)*, vol. 46(4), pp. 1280–1303, 2017.

[24] S. Khuller and J. S. Naor, "Flow in planar graphs with vertex capacities," *Algorithmica*, vol. 11(3), pp. 200–225, 1994.

[25] X. Zhang, W. Liang, and H. Jiang, "Flow equivalent trees in node-edge-capacitated undirected planar graphs," *Inf. Process. Lett.*, vol. 100, pp. 100–115, 2006.

[26] X. Zhang, W. Liang, and G. Chen, "Computing maximum flows in undirected planar networks with both edge and vertex capacities," in *Proceedings of the 24th International Computing and Combinatorics Conference*, 2008, pp. 577–586.

[27] Y. T. Lee, S. Rao, and N. Srivastava, "A new approach to computing maximum flows using electrical flows," in *Proceedings of the 45th symposium on Theory of computing - STOC '13*, 2013, pp. 755–764.

[28] D. Kang and J. Payor, "Flow Rounding," `arXiv:1507.08139`, July 2015.

[29] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *J. Comput. Syst. Sci.*, vol. 26, pp. 362–391, 1983.

[30] S. Khuller, J. Naor, and P. Klein, "The lattice structure of flow in planar graphs," *SIAM J. Discrete Math*, vol. 63, pp. 477–490, 1993.

[31] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian, "Faster shortest-path algorithms for planar graphs," *J. Comput. Syst. Sci.*, vol. 55, pp. 3–23, 1997.

[32] R. Karp, "On the computational complexity of combinatorial problems," *Networks*, vol. 5, pp. 45–68, 1974.

[33] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani, "Matching is as easy as matrix inversion," *Combinatorica*, vol. 7(1), pp. 105–113, 1987.

[34] A. Schrijver, "Finding $k$ partially disjoint paths in a directed planar graph," arXiv:1504.00185, Apr. 2015.

[35] Y. Wang, "Maximum integer flows in directed planar graphs with vertex capacities and multiple sources and sinks," in *Proc. 30th Ann. ACM-SIAM Symp. Discrete Algorithms - SODA '19*, 2019, pp. 554–568.

[36] A. Mądry, "Navigating central path with electrical flows: from flows to matchings, and back," in *FOCS'16: Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, 2013, pp. 253–262.

[37] A. V. Goldberg and S. Rao, "Beyond the flow decomposition barrier," *Journal of the ACM*, vol. 45, pp. 783–797, 1998.

[38] U. Brandes and D. Wagner, "A linear time algorithm for the arc disjoint menger problem in planar directed graphs," *Algorithmica*, vol. 28, pp. 16–36, 2000.

[39] D. Eisenstat and P. N. Klein, "Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graph," in *STOC '13*, 2013.

[40] G. F. Italiano, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen, "Improved algorithms for min cut and max flow in undirected planar graphs," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, 2011, pp. 313–322.

[41] G. Borradaile and P. N. Klein, "An $O(n \log n)$ algorithm for maximum $st$-flow in a directed planar graph," *Journal of the ACM*, vol. 56(2), 2009.

[42] E. W. Chambers, J. Erickson, and A. Nayyeri, "Homology flows, cohomology cuts," *SIAM Journal on Computing*, vol. 41(6), pp. 1605–1634, 2012.

[43] J. M. Hochstein and K. Weihe, "Maximum $st$-flow with $k$ crossings in $O(k^3 n \log n)$ time," in *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms*, 2007, pp. 843–847.

[44] G. L. Miller and J. Naor, "Flow in planar graphs with multiple sources and sinks," *SIAM J. Comput.*, vol. 24(5), pp. 1002–1017, 1995.

[45] W. D. W. K. Ripphausen-Lipa, H., "The vertex-disjoint menger problem in planar graphs." *SIAM J. Comput.*, vol. 26, pp. 331–349, 1997.

[46] Y. P. Liu and A. Sidford, "Faster divergence maximization for faster maximum flow," 2020.

[47] S. Guattery and G. L. Miller, "A contraction procedure for planar directed graphs," in *Proc. 4th annual ACM Symposium on Parallel Algorithms and Architectures*, 1992, pp. 431–441.

[48] S. Even and R. Tarjan, "Network flow and testing graph connectivity," *SIAM Journal on Computing*, vol. 4(4), pp. 507–518, 1975.

[49] G. Borradaile, D. Eppstein, A. Nayyeri, and C. Wulff-Nilsen, "All-Pairs Minimum Cuts in Near-Linear Time for Surface-Embedded Graphs," arXiv:1411.7055, Nov. 2014.

[50] G. Borradaile, "Exploiting planarity for network flow and connectivity problems," Ph.D. dissertation, Brown University, 2008.

[51] M. T. Goodrich, "Planar separators and parallel polygon triangulation," *J. Comput. Syst. Sci.*, vol. 51(3), pp. 374–389, 1995.

[52] P. N. Klein, S. Mozes, and C. Sommer, "Structured recursive separator decompositions for planar graphs in linear time," in *STOC '13*, 2013.

[53] A. Abboud and S. Dahlgaard, "Popular conjectures as a barrier for dynamic planar graph algorithms," in *57th Annual Symposium on Foundations of Computer Science (FOCS)*, 2016.

[54] J. Erickson and Y. Wang, "Four shortest disjoint paths in planar graphs," preprint, 2018.

[55] M. Middendorf and F. Pfeiffer, "On the complexity of the disjoint paths problem," *Combinatorica*, vol. 13, no. 1, pp. 97–107, 1993.

[56] N. Robertson and P. D. Seymour, "Graph minors. XIII. The disjoint paths problem," *J. Combin. Theory, Ser. B*, vol. 63, no. 1, pp. 65–110, 1995.

[57] K.-i. Kawarabayashi and P. Wollan, "A shorter proof of the graph minor algorithm: The unique linkage theorem," in *Proc. 42nd Ann. ACM Symp. Theory Comput.*, New York, NY, USA, 2010, pp. 687–694.

[58] A. Schrijver, "Finding $k$ disjoint paths in a directed planar graph," *SIAM J. Comput.*, vol. 23, no. 4, pp. 780–788, 1994.

[59] M. Cygan, D. Marx, M. Pilipczuk, and M. Pilipczuk, "The planar directed $k$-vertex-disjoint paths problem is fixed-parameter tractable," in *Proc. IEEE 54th Ann. Symp. Found. Comput. Sci.*, Washington, DC, USA, 2013, pp. 197–206.

[60] G. Naves and A. Sebő, "Multiflow feasibility: an annotated tableau," *Research Trends in Combinatorial Optimization*, pp. 261–283, 2008.

[61] H. van der Holst and J. C. de Pina, "Length-bounded disjoint paths in planar graphs," *Discrete Appl. Math.*, vol. 120, no. 1–3, pp. 251–261, 2002.

[62] G. Borradaile, A. Nayyeri, and F. Zafarani, “Towards single face shortest vertex-disjoint paths in undirected planar graphs,” in *Proc. 23rd Ann. Europ. Symp. Algorithms*, ser. Lecture Notes Comput. Sci., no. 9294. Springer, 2015, pp. 227–238.

[63] F. Zafarani, “A structural theorem for shortest vertex-disjoint paths computation in planar graphs,” M.S. thesis, Oregon State University, May 2016. [Online]. Available: https://ir.library.oregonstate.edu/xmlui/handle/1957/59656

[64] S. Datta, S. Iyer, R. Kulkarni, and A. Mukherjee, “Shortest $k$-disjoint paths via determinants,” arXiv:1802.01338, Feb 2018.

[65] B. Yang, S.-Q. Zheng, and E. Lu, “Finding two disjoint paths in a network with normalized $\alpha^{-}$-min-sum objective function,” in *IASTED International Conference on Parallel and Distributed Computing and Systems*, 2005, pp. 342–348.

[66] B. Yang, S.-Q. Zheng, and S. Katukam, “Finding two disjoint paths in a network with min-min objective function,” in *IASTED International Conference on Parallel and Distributed Computing and Systems*, 2003, pp. 75–80.

[67] H. Kaplan and Y. Nussbaum, “Maximum flow in directed planar graphs with vertex capacities,” *Algorithmica*, vol. 61(1), pp. 174–189, 2011.

[68] W. S. Jewell, “Optimal flow through networks,” Oper. Rec. Center, MIT, Interim Tech. Rep. 8, 1958, cited in [7].

[69] R. G. Busacker and P. J. Gowen, “A procedure for determining minimal-cost network flow patterns,” Oper. Res. Office, Johns Hopkins U., Technical Paper ORO-TP-15, November 1960. [Online]. Available: http://www.dtic.mil/docs/citations/AD0249662

[70] M. Iri, “A new method of solving transportation-network problems,” *J. Oper. Res. Soc. Japan*, vol. 3, pp. 27–87, 1960. [Online]. Available: http://www.orsj.or.jp/~archive/menu/01_03.html

[71] L. G. Khachiyan, “A polynomial algorithm in linear programming,” *Doklady Akademii Nauk SSSR*, vol. 244, pp. 1093–1096, 1979.

[72] T. Ito, Y. Miyamoto, H. Ono, H. Tamaki, and R. Uehara, “Route-enabling graph orientation problems,” in *ISAAC 2009: Algorithms and Computation*, 2009, pp. 403–412.

[73] H. Robbins, “A theorem on graphs, with an application to a problem on traffic control,” *Amer. Math. Monthly*, vol. 46(5), pp. 281–283, 1939.

[74] A. Björklund and T. Husfeldt, “Shortest two disjoint paths in polynomial time,” in *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, 2014, pp. 211–222.

[75] P. Klein, “Multiple-source shortest paths in planar graphs,” in *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms - SODA ’05*, 2005, pp. 146–155.

[76] A. Steiger, "Single-face non-crossing shortest paths in planar graphs," M.S. thesis, University of Illinois at Urbana-Champaign, 2017. [Online]. Available: https://www.ideals.illinois.edu/bitstream/handle/2142/98345/STEIGER-THESIS-2017.pdf?sequence=1

[77] H.-C. Liang and H.-I. Lu, "Minimum cuts and shortest cycles in directed planar graphs via noncrossing shortest paths," *SIAM J. Discrete Math*, vol. 31(1), pp. 454–476, 2017.

[78] K. Bérczi and Y. Kobayashi, "The Directed Disjoint Shortest Paths Problem," in *25th Annual European Symposium on Algorithms (ESA 2017)*, vol. 87, 2017, pp. 13:1–13:13.

[79] D. Lichtenstein, "Planar formulae and their uses," *SIAM J. Comput.*, vol. 11(2), pp. 329–343, 1982.