APPROXIMATION ALGORITHMS FOR THE MINIMUM CONGESTION ROUTING PROBLEM VIA K-ROUTE FLOWS

BY

MARK IDLEMAN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:

Professor Chandra Chekuri

ABSTRACT

Given a directed network $G = (V, E)$ with source and target nodes s and t, respectively, and an integral capacity c_e on each edge $e \in E$, an elementary k-flow is defined as a flow of 1 unit along each of k edge-disjoint s-t paths. A k-route flow, first introduced as a concept by Kishimoto, is defined as a non-negative linear sum of elementary k-flows. In this thesis, the study of k-route flows is extended by presenting efficient algorithms to calculate exact and approximate decompositions of k-route flows into their constituent elementary k-flows.

In addition, such decomposition algorithms are shown to prove useful in developing approximation algorithms for the well-studied Minimum Congestion Routing Problem. Given a directed network $G = (V, E)$, a set of source-sink pairs $(s_1, t_1), ..., (s_l, t_l)$, and an integer k, the goal of the Minimum Congestion Routing Problem is to find k edge-disjoint paths between each pair (s_i, t_i) while minimizing the congestion over all chosen paths (defined as the maximum over all edges of the number of chosen paths that share a single edge). Early applications of randomized rounding introduced by Raghavan and Tompson provided a simple approximation algorithm for the case where $k = 1$, but attempts to achieve similar approximation bounds in the case where $k > 1$ have up until this point required the use of more advanced dependent rounding schemes. Utilizing the k-route flow decomposition algorithms presented in this thesis, we propose approximation algorithms for the Minimum Congestion Routing Problem for the case where $k > 1$ that mimic the straightforward approach of Raghavan and Tompson while achieving identical approximation guarantees. Finally, we implement two variants of the exact k-route flow decomposition algorithm proposed in this thesis, and experimentally compare their performance using flows generated from various graph structures.

Dedicated to my parents and to close friends near and far, all of whom have given me the support and stability to push through difficult times in order to achieve things I am truly proud of.

ACKNOWLEDGMENTS

The completion of this thesis would not have been possible without the massive amount of support that I received from friends, family, roommates, and professors, all of whom helped to provide me with the encouragement and knowledge required to continue my pursuit of research and ultimately to create this document. Since my interest in theoretical computer science first began during a semester abroad in Budapest, Hungary, I've been challenged to quickly learn new concepts spanning a wide variety of subject matter, and to push through projects and assignments that at times felt beyond my abilities. Through the support of mentors and advisors, both during my undergraduate years at Amherst College and during my time at the University of Illinois, I've been able to overcome these challenges, and have been fortunate enough to feel more confident in my abilities with each new goal attained.

In particular, I thank my advisor, Chandra Chekuri, for his consistent patience, commitment, and encouragement throughout the completion of this thesis. His emphasis on ensuring my understanding of material that was often challenging and new to me, along with his efforts to push me to independently work through difficult problems that arose throughout the course of the past year, have helped me immensely to develop as both a researcher and as an analytical thinker. I greatly appreciate the time he devoted to supporting me throughout the development of this thesis, and to ensuring that it could become something I am proud to have created.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION AND PRELIMINARIES

The contents of this thesis are laid out as follows. In this chapter, we introduce the concept of k-route flows along with prior work done on the subject, and describe variants of the Chernoff bounds that will be utilized in later chapters. In Chapter 2, we introduce exact and approximate algorithms for creating k-route flow decompositions of starting flows with particular properties. In Chapter 3, we review the history of the Minimum Congestion Routing Problem, and introduce a new approximation algorithm for the problem based on the decomposition algorithms of Chapter 2 that matches the current-best approximation bounds. In Chapter 4, we describe a simple implementation of the exact decomposition algorithm of Chapter 2 and experiementally test its performance. In Chapter 5, we give an overview of the conclusions made based on the work in this thesis, and suggest possible directions for future research into the subject.

1.1 Network Flows and Flow Decomposition

Throughout this thesis, we will be concerned mainly with directed graphs. Unless otherwise noted, we will use n and m to represent the number of nodes and edges of a graph in question, respectively. Given a directed graph $G = (V, E)$ with non-negative capacities c_e on each edge $e \in E$, a *network flow* is defined as a function $f : E \to \mathbb{R}^+$. Throughout this thesis, network flows will often be expressed as vectors to help differentiate them from scalar values; for a given flow $\overline{f}$ written in this way, f_e is the flow value on an edge e. A *feasible flow* of k units from a source node s to a target node t is a flow $\overline{f}$ such that

1. The flow value f_e on each edge $e \in E$ satisfies $0 \leq f_e \leq c_e$,

2. k units of flow leave vertex s and enter vertex t, i.e.,

$$\sum_{e=(s,u)} f_e - \sum_{e=(u,s)} f_e = \sum_{e=(u,t)} f_e - \sum_{e=(t,u)} f_e = k, \text{ and}$$

3. $\overline{f}$ satisfies the *flow conservation property*: for each vertex $v \in V \setminus \{s,t\}$,

$$\sum_{e=(u,v)} f_e = \sum_{e=(v,u)} f_e.$$

The above definition characterizes network flows based on the flow value assigned to each edge in the graph; we will refer to such formulations as *edge-based flows*. Oftentimes, it is beneficial to also consider an equivalent formulation of network flows, the *path-based flow*. In this characterization, an s-t flow in a graph G is viewed as a sum of individual flows, each consisting of a single s-t path or a single cycle in G. Given an edge-based flow $\overline{f}$, the process of *flow decomposition* allows us to decompose $\overline{f}$ into an equivalent path-based flow. Such decompositions can be computed easily in $O(nm)$ time (the details of the decomposition algorithm can be found in [1]), and are extremely useful in many applications.

The well known Maximum Flow Problem is that of finding a flow $\overline{f}$ that routes as much flow as possible from a *source* node s to a *sink* node t, while obeying the capacity constraints on each edge and ensuring that the flow conservation property holds (i.e., the total flow entering any node is equal to the total flow exiting that node, aside from s and t).

1.2 k-route Flows

An *elementary k-flow* from s to t is defined as an s-t flow of k total units, consisting of 1 unit sent along each of k edge-disjoint s-t paths. A *k-route flow* from s to t is an s-t flow that can be expressed as a nonnegative linear sum of elementary k-flows. Given a k-route flow $\overline{f}$, a *k-route flow decomposition* of $\overline{f}$ is a set of elementary k-flows that together constitute $\overline{f}$; we will be especially interested in algorithms used to find k-route flow decompositions in Chapter 2.

Kishimoto and Takeuchi [2] first introduced the notion of 2-route flows, later presenting the more general k-route flows in [3]. In [3], Kishimoto developed an algorithm for the k-Route Flow Problem, the problem of finding a k-route flow from s to t of maximum value given flow capacities c_e for each edge e, that required computing at most k maximum flows.

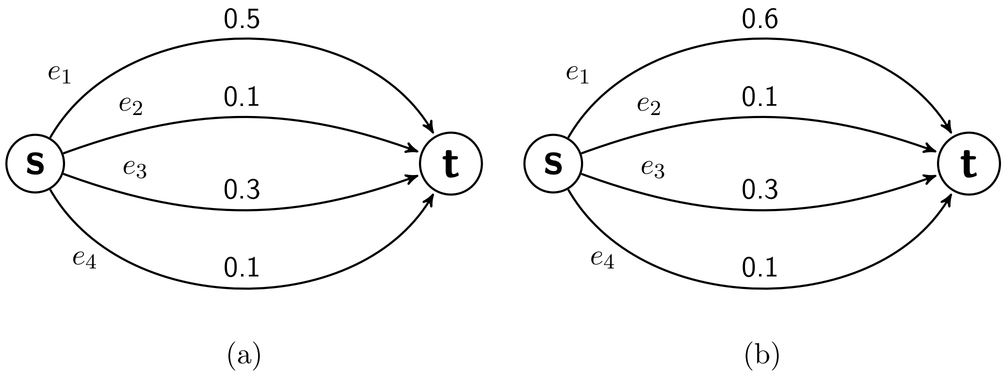

Figure 1.1: The flow in (a) is decomposable into a 2-route flow by sending 0.3 units of flow along e_1 and e_3, 0.1 units of flow along e_1 and e_2, and 0.1 units of flow along e_1 and e_4. The flow in (b) is not decomposable into a 2-route flow; the 0.6 units of flow on edge e_1 presents a bottleneck to obtaining a valid decomposition.

Aggarwal and Orlin described an improved algorithm for the k-Route Flow Problem using binary search that required only $\min\{k, \log(kU)\}$ maximum flow computations, where U is the maximum capacity of any edge in the graph [4]. Note that the authors of [4] assumed all flows to be acyclic without loss of generality; the following lemma shows that such an assumption can be made without any effect on the value of the flow in question.

Lemma 1. *Given a feasible s-t flow $\overline{x}$ in a directed graph G where $\overline{x}$ contains cycles carrying non-zero flow value, there exists a flow $\overline{x'}$ of equal value to $\overline{x}$ such that $\overline{x'}$ is acyclic.*

Proof. Given $\overline{x}$, the basic flow decomposition algorithm of [1] can be used to decompose $\overline{x}$ into a set of s-t paths and cycles in G. For each cycle C found in the resulting decomposition, the cycle can be removed from $\overline{x}$ by reducing the flow on all edges in C by the minimum flow value on any edge in the cycle. This necessarily drops the flow on such edges to 0 and disconnects the cycle, without affecting the value of the flow from s to t. Repeating the process for all cycles originally found in the decomposition of $\overline{x}$ therefore yeilds a new acyclic flow $\overline{x'}$ of equal value. $\square$

By Lemma 1, any s-t flow we consider that contains cycles can be converted into a flow of the same value without cycles. In addition to the algorithm they presented for the k-Route Flow Problem, the authors of [4] also presented a useful characterization of k-route flows, laid out in the lemmas below, by considering the flow going through each edge relative to the total flow sent from s to t.

Lemma 2. *Let $\overline{x}$ be a feasible s-t flow in a graph $G = (V, E)$ with kv total units of flow entering t, for some integer k. If $0 \leq x_e \leq v$ for all edges $e \in E$, then there exists an elementary k-flow $\overline{y}$ in G such that $y_e = 0$ if $x_e = 0$ and $y_e = 1$ if $x_e = v$.*

Proof. Consider the Maximum Flow Problem in G where each edge $e \in E$ has an upper bound of $u_e = \lceil x_e/v \rceil$ and a lower bound of $l_e = \lfloor x_e/v \rfloor$. Define a flow $\overline{f}$ where $f_e = x_e/v$ for each edge $e \in E$. Note that $\overline{f}$ is a feasible flow for the aforementioned Maximum Flow Problem; in addition, $f_e = 0$ if $x_e = 0$ and $f_e = 1$ if $x_e = v$. $\overline{f}$ may not be an integral flow, but because the bounds on each edge are integral and the constraint matrix of the Maximum Flow Problem is totally unimodular, there exists an integral flow $\overline{y}$ of the same value satisfying the upper and lower bounds on each edge. Because $x_e \leq v$ for all edges $e \in E$, based on our choice of u_e and l_e, the capacity on each edge is either 0 or 1, meaning $\overline{y}$ must carry a flow of 0 or 1 on each edge. Because $\overline{y}$ has the same value as $\overline{f}$ and $\overline{f} = \overline{x}/v$, $\overline{y}$ must carry exactly k units of flow from s to t. Therefore, $\overline{y}$ is an elementary k-flow such that $y_e = 0$ if $x_e = 0$ and $y_e = 1$ if $x_e = v$. □

Lemma 1 leads to the proof of the following statement regarding the necessary conditions for a feasible flow to be expressed as a k-route flow.

Lemma 3. *Let $\overline{x}$ be a feasible s-t flow in a graph $G = (V, E)$ with kv total units of flow entering t, for some integer k. If $0 \leq x_e \leq v$ for all edges $e \in E$, then $\overline{x}$ is a k-route flow.*

Proof. Assume $v > 0$ (as the $v = 0$ case is trivial), and call an edge e *intermediate* if $0 < x_e < v$; the claim will be proven by induction on the number of intermediate edges. If the number of intermediate edges is 0 (i.e., all edges have a flow of 0 or v), then a flow $\overline{y}$ can be created by setting $y_e = 0$ if $x_e = 0$ and $y_e = 1$ if $x_e = v$. Because we have assumed $\overline{x}$ to be acyclic and the flow x_e on each edge e is 0 or v, $\overline{x}$ must be decomposable into k edge-disjoint s-t paths, each carrying v units of flow. It follows that $\overline{y}$ is a flow of k total units that can be decomposed into k edge-disjoint paths, each carrying 1 unit of flow; therefore, $\overline{y}$ is an elementary k-flow. It is clear that $\overline{x} = v \cdot \overline{y}$, and so $\overline{x}$ is a k-route flow.

Next consider the case where the number of intermediate edges is $i > 0$, and suppose that the lemma holds when less than i intermediate edges exist. Let $\overline{y}$ be an elementary k-flow as described in Lemma 1, with $y_e = 0$ if $x_e = 0$ and $y_e = 1$ if $x_e = v$. Define $\Delta_1 = \min\{v - x_e \mid y_e = 0\}$, $\Delta_2 = \min\{x_e \mid y_e = 1\}$, and $\Delta = \min\{\Delta_1, \Delta_2\}$. In addition, let $v' = v - \Delta$. Intuitively, Δ is a scaling factor for $\overline{y}$; we hope to create a new flow $\overline{x'} = \overline{x} - \Delta\overline{y}$ such that $\overline{x'}$ has at most $i - 1$ intermediate edges, where an edge e in $\overline{x'}$ is intermediate if $0 < x'_e < v'$. To guarantee that the number of intermediate edges is reduced in $\overline{x'}$, it is necessary to reduce the flow x_e on all edges e with $y_e = 1$ until either a single edge has had its flow reduced to 0 (and therefore is no longer intermediate), or an edge e with flow $x_e = v$ has its flow reduced enough to match that of another edge e' with $y_{e'} = 0$, whose flow was not reduced (thus making $x'_e = x'_{e'} = v'$ in the new flow $\overline{x'}$).

More formally, Δ_1 represents the maximum amount that any edge e with $x_e = v$ can have its flow reduced before it attains a flow equal to that of the edge e' with $y_{e'} = 0$ that maximizes $x_{e'}$. Δ_2 represents the amount that the flow on all edges e with $y_e = 1$ can be reduced before one such edge attains a flow value of 0. Therefore, reducing the flow on all edges with $y_e = 1$ by Δ ensures that the number of intermediate edges will drop by at least one, as at least one new edge will have a flow of 0 or v' in the resulting flow $\overline{x'}$. $\overline{x'}$ satisfies the necessary conditions for Lemma 2, so by the inductive hypothesis, $\overline{x'}$ is a k-route flow. It follows that $\overline{x} = \overline{x'} + \Delta\overline{y}$ is also a k-route flow. $\square$

The inductive proof of Lemma 3 implies that if a single elementary k-flow as described in the statement of the lemma can be found quickly, this process can be iteratively carried out to decompose a starting k-route flow into its constituent elementary k-flows. Such algorithms are the focus of Chapter 2.

1.3 Chernoff Bounds

At several points throughout the remainder of this thesis, we will utilize variants of the Chernoff bounds [5] to analyze the approximation guarantees of various algorithms. The Chernoff bounds provide strong and useful tail bounds on the sum of a set of independent random variables. Of particular interest to us will be the following versions of the bounds, stated without proof.

Theorem 1. *Let $X_1, \ldots, X_n$ be n independent random variables (not necessarily distributed identically), with each variable X_i taking a value of 0 or v_i for some value $0 < v_i \leq 1$. Then for $X = \sum_{i=1}^{n} X_i$, $E[X] \leq \mu$, and $\delta > 0$,*

$$\Pr[X \geq (1+\delta)\mu] < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}.$$

Lemma 4. *For $0 \leq \delta \leq 1$, we have*

$$\left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} \leq e^{-\mu\delta^2/3}.$$

In addition, we will prove the following corollary of Theorem 1 and Lemma 4.

Corollary 1. *Let $X = \sum_{i=1}^{n} X_i$ be the sum of n independent random variables $X_1, \ldots, X_n$, with each $X_i \in [0, \alpha]$ for some $\alpha > 0$. Let $\theta = E[X]$, with $\theta \leq \mu$. For $0 \leq \delta < 1$, we have*

$$\Pr[X \geq (1+\delta)\mu] < e^{-\mu\delta^2/(3\alpha)}.$$

Proof. Consider the scaled sum $X/\alpha = \sum_{i=1}^{n} X_i/\alpha$. Note that each X_i/α lies in the range $[0,1]$, and $E[X/\alpha] = \theta/\alpha \leq \mu/\alpha$. Therefore, by Theorem 1 and Lemma 4, we have

$$Pr[X \geq (1+\delta)\mu] = \Pr[X/\alpha \geq (1+\delta)\mu/\alpha] < e^{-(\mu/\alpha)\delta^2/3} = e^{-\mu\delta^2/(3\alpha)}.$$

□

CHAPTER 2

K-ROUTE FLOW DECOMPOSITION ALGORITHMS

In this chapter, we present both exact and approximate algorithms used to decompose a k-route flow into a corresponding set of elementary k-flows. As noted in Chapter 1, Lemma 1 allows us to assume without loss of generality that all flows dealt with in the chapters that follow are acyclic.

2.1 Exact Algorithm

Recall from Lemma 3 that k-route flows can be characterized as flows carrying kv units of flow from the source node to the target node, with the flow on any edge not exceeding v units. Over the course of this section, we will propose an efficient algorithm for computing an exact k-route flow decomposition, and will establish a proof of the following theorem.

Theorem 2. *Given an acyclic s-t flow $\overline{x}$ in a directed network $G = (V, E)$ sending kv units of flow from s to t such that $0 \leq x_e \leq v$ for all edges $e \in E$, $\overline{x}$ can be decomposed into constituent elementary k-flows in $O(T(n, m) + m^2)$ time, where $n = |V|$, $m = |E|$, and $T(n, m)$ is the time required to compute a maximum s-t flow in a unit capacity graph with n vertices and m edges.*

2.1.1 Finding an Elementary k-flow

In order to decompose a k-route flow into a weighted sum of elementary k-flows, we must first have the ability to compute the specific elementary k-flows described in Lemma 2 algorithmically, so that the iterative decomposition algorithm implied by the inductive proof of Lemma 3 may be carried out.

Given an s-t flow $\overline{x}$ of kv total units in a graph $G = (V, E)$ with $0 \leq x_e \leq v$ for all edges $e \in E$, we hope to find an elementary k-flow $\overline{y}$ from s to t such that $y_e = 0$ if $x_e = 0$ and $y_e = 1$ if $x_e = v$. Note that without this constraint on $\overline{y}$, finding an elementary k-flow is equivalent to finding k edge-disjoint paths from s to t, which can be done easily using a unit capacity maximum flow algorithm. By a recent result in [6], a unit capacity maximum flow in a directed graph with n vertices and m edges can be calculated in $\tilde{O}(m^{10/7})$ time, giving such an algorithm an equivalent runtime.

To find an elementary k-flow that obeys these constraints, we reduce the problem to that of finding a feasible lower-bounded circulation in a slightly modified graph. Given a flow network $H = (V', E')$ with upper and lower bounds u_e and l_e on each edge e and demands d_v on each vertex v, a *feasible circulation* $\overline{f}$ in H must satisfy the following conditions:

1. For each edge $e \in E'$, $l_e \leq f_e \leq u_e$.

2. For each vertex $v \in V'$, $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$.

We first create a new flow network by modifying G as follows. We add to G a single directed edge e from t to s with upper and lower bounds $u_e = l_e = k$, and for any edge e with $x_e = v$, we give e lower and upper bounds of $l_e = u_e = 1$; we will refer to such edges as *special*. Non-special edges will be assumed to have a lower bound $l_e = 0$ and an upper bound $u_e = 1$. Finally, we simplify the graph by removing any edges e from consideration with $x_e = 0$ (note that this does not affect the correctness of the resulting elementary k-flow we find). Each vertex v is given a demand of $d_v = 0$. Note that any feasible circulation $\overline{f}$ in this network must send exactly k units of flow from s to t along k edge-disjoint paths (given the assumption that x is acyclic), and must ensure that all special edges are saturated; therefore, removing from $\overline{f}$ the t-s back edge would yield the desired elementary k-flow $\overline{y}$, where $y_e = 0$ if $x_e = 0$ and $y_e = 1$ if $x_e = v$.

To find a feasible circulation satisfying these lower bounds, we use a simple reduction to the standard Max-Flow Problem on a new graph G' (as outlined in [7]), constructed from G as follows.

1. Add "super-source" and "super-sink" nodes s^* and t^*.

2. Consider each edge $e = (u, v)$ with lower bound $l_e > 0$. Remove e, and add to the graph the edges (s^*, v) and (u, t^*), each with capacity l_e.

Note that all edges in G' leaving s^* or entering t^* that are added to replace a special edge in G have unit capacity, as each special edge has a lower bound of 1. In addition, the reduction replaces the t-s back edge e first added to G (with $l_e = u_e = k$) with edges (s^*, s) and (t, t^*), each with capacity k. Note that these edges are the only edges in G' with capacities greater than 1, but we can still think of G' as a unit capacity graph by replacing the edges (s^*, s) and (t, t^*) each with k parallel unit capactiy edges.

Then, to find an elementary k-flow that saturates all special edges in G, we calculate an s^*-t^* max-flow $\overline{h}$ in G' using a unit capacity max-flow algorithm. For each edge e in G with lower bound $l_e > 0$ (including the added t-s back edge), we add exactly 2 new edges to form G', meaning $|E(G')| = O(|E(G)|)$; therefore, we can calculate $\overline{h}$ in $O(T(m, n))$ time, where $n = |V(G)|$, $m = |E(G)|$, and $T(m, n)$ is the time required to compute a maximum flow in a unit capacity graph with n vertices and m edges.

If a feasible circulation $\overline{f}$ exists in G, then any s^*-t^* max-flow $\overline{h}$ calculated in G' must be *saturating* - all edges leaving s^* and all edges entering t^* must be fully saturated with flow in $\overline{h}$ (see the proof of correctness for this reduction provided in Section 2.1.2). For each edge $e = (u, v)$ in the original flow network with $0 < l_e = u_e = 1$, we add edges (s^*, v) and (u, t^*) after removing e, each with capacity $l_e = u_e = 1$, and such edges must be saturated in $\overline{h}$ (if a corresponding feasible circulation exists).

Therefore, we can obtain a feasible circulation $\overline{f}$ in the original flow network by removing s^* and t^* from G' (along with all adjacent edges), adding back all edges e with $l_e > 0$ from the original flow network G that were removed in G' and assigning them each a flow of $f_e = l_e = u_e = 1$, and assigning a flow of $f_e = h_e$ to all other edges from G'. Then, as previously described, the desired elementary k-flow $\overline{y}$ from s to t can be obtained by simply removing the t-s back edge originally added to G to form an instance of the Circulation Problem; it is guaranteed that the resulting flow sends k total units from s to t and saturates all special edges from G.

2.1.2 Proof of Correctness

Define $\overline{x}$ to be an s-t flow of kv total units in a graph $G = (V, E)$, where for all edges $e \in E$, $0 \leq x_e \leq v$. Recall from Lemma 2 that under these conditions, there must exist an elementary k-flow $\overline{y}$ in G such that $y_e = 0$ if $x_e = 0$ and $y_e = 1$ if $x_e = v$. Therefore, after modifying G to create an instance of the Lower-Bounded Circulation Problem as described in Section 2.1.1, there must exist a feasible circulation in the resulting graph that satisfies the upper and lower bounds on the capacities of each edge, as such a circulation could be created by starting with $\overline{y}$ and adding back the back edge from t to s. In turn, given the following proof of correctness for the reduction given in Section 2.1.1 from the Lower-Bounded Circulation Problem to the Maximum Flow Problem, it follows that we are able to find an elementary k-flow that saturates all special edges in G by running any general unit capacity maximum flow algorithm once.

Let G be an instance of the Lower-Bounded Circulation Problem as described in Section 2.1.1, and define $G' = (V', E')$ to be the s^*-t^* max-flow instance built from G by following the reduction outlined there.

Lemma 5. *There exists a feasible circulation $\overline{f}$ in G if and only if there exists a saturating max-flow $\overline{f'}$ in G'.*

Proof. First, suppose there exists a feasible circulation $\overline{f}$ in G obeying all upper and lower bounds on each edge, along with $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v = 0$ for all vertices $v \in V$. Note that for each edge e in G with $l_e > 0$, we have $l_e = f_e = u_e$. In addition, because $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v = 0$ for all vertices $v \in V$, $\overline{f}$ satisfies the flow conservation property.

We construct a saturating max-flow $\overline{f'}$ in G' as follows. For each edge $e \in E$ with $l_e = 0$, e is unaffected by the reduction used to form G'; therefore, we set $f'_e = f_e$, which sends the same f_e units of flow along each such edge in G'. In addition, for all edges $e = (u, v)$ in G for which we add edges $e_1 = (s^*, v)$ and $e_2 = (u, t^*)$ to G', we set $f'_{e_1} = f'_{e_2} = f_e$. Because f_e units of flow are sent along e in G and $\overline{f}$ is a feasible circulation satisfying $f^{\text{in}}(v) - f^{\text{out}}(v) = 0$ for all $v \in V$, there must be f_e units of flow entering u and f_e units of flow leaving v in G. Therefore, given that $f'_e = f_e$ for all edges $e \in E$ with $l_e = 0$, $\overline{f'}$ can send f_e units of flow along each of the s^*-v and u-t^* edges in G', saturating them. By construction, $\overline{f'}$ is clearly a saturating s^*-t^* flow in G', and it must obey the flow conservation property, as for each vertex v, we maintain $f'^{\text{in}}(v) - f'^{\text{out}}(v) = 0$. Therefore, $\overline{f'}$ is a valid saturating max-flow in G'.

Now, suppose that there exists a saturating max-flow $\overline{f'}$ in G'. We will build a feasible circulation $\overline{f}$ in G from $\overline{f'}$ as follows. First, we remove s^* and t^* from G' (along with all adjacent edges), and we add back all edges e with $l_e > 0$ in G that were removed when forming G', assigning them each a flow of $f_e = l_e = u_e$. For all other edges e in $E \cap E'$, we assign a flow of $f_e = f'_e$ to e in G. Because $\overline{f'}$ is a saturating flow in G', for all edges $e = (u, v)$ in G with upper and lower bounds $0 < l_e = u_e$, $\overline{f'}$ sent u_e units of flow out of v and u_e units of flow into u in G'. Therefore, adding back such edges with a flow of $f_e = l_e = u_e$ in G ensures that $l_e \leq f_e \leq u_e$ for all edges e in G, and $f^{\text{in}}(v) - f^{\text{out}}(v) = 0 = d_v$ for all vertices v (as we have ensured that the flow entering and exiting each vertex remains the same as it did in $\overline{f'}$, and $\overline{f'}$ met flow conservation constraints as it was a valid max-flow). Therefore, $\overline{f}$ is a feasible circulation in G. □

2.1.3 Decomposition Algorithm

Once an elementary k-flow $\overline{y}$ has been obtained, we complete one iteration of the decomposition algorithm by calculating an appropriate scaling factor Δ (as outlined in the proof of Lemma 2) and obtaining a new flow $\overline{x'} = \overline{x} - \Delta\overline{y}$ with fewer intermediate edges. As the number of intermediate edges is guaranteed to decrease by at least one over each iteration of the algorithm, by simply recalculating each elementary k-flow from scratch, we obtain the following lemma.

Lemma 6. *Given an acyclic s-t flow $\overline{x}$ in a directed network $G = (V, E)$ sending kv units of flow from s to t such that $0 \leq x_e \leq v$ for all edges $e \in E$, $\overline{x}$ can be decomposed into constituent elementary k-flows in $O(m \cdot T(n, m))$ time, where $n = |V|$, $m = |E|$, and $T(n, m)$ is the time required to compute a maximum s-t flow in a unit capacity graph with n vertices and m edges.*

However, rather than recalculating a new elementary k-flow each iteration, one can find such a flow by simply modifying the graph G' and augmenting the s^*-t^* flow found during the last iteration. We will now complete the proof of Theorem 2.

Consider any intermediate edge $e = (u, v)$ in $\overline{x}$ that becomes no longer intermediate in $\overline{x'}$. There are two cases: either e becomes a *dead* edge in $\overline{x'}$ (i.e., $x'_e = 0$), or e becomes *tight* in $\overline{x'}$ (i.e., $x'_e = v'$).

Removing Dead Edges: In the first case, the new flow x'_e on e becomes 0 (note that in this case, we must have $y_e = 1$, as e's flow must have been reduced from what it was in $\overline{x}$). Because $y_e = 1$, there was 1 unit of flow going through e in $\overline{f}$, the s^*-t^* max-flow found in G' that was used to create $\overline{y}$ in the previous iteration of the algorithm. However, because the flow on e in $\overline{x'}$ is now 0, the new elementary k-flow $\overline{y'}$ we calculate in the current iteration of the decomposition algorithm must have $y'_e = 0$. To calculate $\overline{y'}$, we simply modify the graph G', along with the s^*-t^* flow $\overline{f}$ found in the last iteration, as follows.

Lemma 7. *A new elementary k-flow $\overline{y'}$ satisfying $y'_e = 0$ if $x'_e = 0$ and $y'_e = 1$ if $x'_e = v$ can be calculated in $O(m+n)$ time, given a saturating flow $\overline{f}$ in G' from the previous iteration of the decomposition, and one new dead edge e.*

Proof. To begin, a breadth-first search is run over the edges of G' carrying a non-zero flow in $\overline{f}$ between the endpoints of e, to detect if e is part of a cycle C carrying 1 unit of flow in $\overline{f}$. If this is the case, the flow on e can be reduced by simply reducing the flow on all edges in C by 1 unit, without affecting the overall flow value of $\overline{f}$. If e is not contained in such a cycle, we proceed by finding an s^*-t^* path p in G' that contains e, such that p sends 1 unit of flow from s^* to t^* in $\overline{f}$; this can be done in $O(m+n)$ time by running a breadth first search over the edges of G' that have non-zero flow in $\overline{f}$. We then remove e from G', reduce the flow in $\overline{f}$ on all edges in p by 1, and find one new augmenting s^*-t^* path in $O(m+n)$ time to obtain a new max-flow $\overline{f'}$ in G' from which $\overline{y'}$ can be extracted. In the worst case, breadth-first search is run a constant number of times to modify $\overline{f}$ to account for the newly dead edge e, giving an overall running time of $O(m+n)$.

To see why such an augmenting path must exist, note that after the flow on p is reduced by 1, the edges leaving s^* (and those entering t^*) are no longer fully saturated. However, structurally, G' contains the same edges as it would have if we had reconstructed a new max-flow instance from scratch; we have simply removed an edge whose flow had dropped to 0. Because $\overline{x'}$ must still satisfy the inductive hypothesis used to prove Lemma 2, there must exist a new elementary k-flow $\overline{y'}$ satisfying $y'_e = 0$ if $x'_e = 0$ and $y'_e = 1$ if $x'_e = v$, and therefore, by the proof of correctness in Section 2.1.2, there must exist a saturating s^*-t^* max-flow in G' from which $\overline{y'}$ can be extracted. Because only a single edge leaving s^* is left unsaturated after reducing the flow on p, finding a single augmenting path yields a new saturating s^*-t^* flow. $\square$

Mending Tight Edges: The second way in which an edge $e = (u, v)$ can become no longer intermediate is if after reducing the flow $\overline{x}$ by $\Delta\overline{y}$ to obtain a new flow $\overline{x'}$, e becomes tight (i.e., $x_e < v$ but $x'_e = v'$). Note that in this case, $y_e = 0$, as otherwise we'd have $x'_e = x_e - \Delta < v - \Delta = v'$. Because e becomes tight in $\overline{x'}$, in the elementary k-flow $\overline{y'}$ we find using $\overline{x'}$, y'_e must be 1.

Lemma 8. *A new elementary k-flow $\overline{y'}$ satisfying $y'_e = 0$ if $x'_e = 0$ and $y'_e = 1$ if $x'_e = v$ can be calculated in $O(m + n)$ time, given a saturating flow $\overline{f}$ in G' from the previous iteration of the decomposition, and one new tight edge e.*

Proof. We begin by modifying G' by removing e and adding edges (s^*, v) and (u, t^*), each with capacity 1. Note that once again, G' is now identical to the max-flow instance we'd construct if starting from scratch using $\overline{x'}$, as we have simply removed e and added the appropriate edges leaving from s^* and entering t^*. Because $\overline{x'}$ must satisfy the inductive hypothesis used in the proof of Lemma 2, there must exist an elementary k-flow $\overline{y'}$ that includes all tight edges in $\overline{x'}$; it follows from the correctness of the reduction in Section 2.1.2 that after modifying G', there must exist some s^*-t^* max-flow that saturates all edges leaving s^* and all edges entering t^*. Only one unit capacity edge was added leaving s^* (along with one entering t^*), so finding a single augmenting path in G' in $O(m + n)$ time gives a new s^*-t^* max-flow from which $\overline{y'}$ can be extracted. $\square$

Mending Multiple Edges: If, after calculating a new flow $\overline{x'}$, multiple edges in G become no longer intermediate, the above techniques can simply be applied multiple times, as follows.

Lemma 9. *Given i edges that have become either dead or tight in the new flow $\overline{x'}$, a new elementary k-flow $\overline{y'}$ satisfying $y'_e = 0$ if $x'_e = 0$ and $y'_e = 1$ if $x'_e = v$ can be calculated by finding at most i augmenting paths in G', given a saturating flow $\overline{f}$ in G' from the previous iteration of the decomposition.*

Proof. If all i new non-intermediate edges in $\overline{x'}$ have had their flow drop to 0, such edges can be dealt with one at a time; for each dead edge e, we find an s^*-t^* path p containing e, reduce the flow on p by 1, remove e from the graph, and find one new augmenting path from s^* to t^* before repeating this process for the remaining edges. Because after each modification we're left with one fewer edge to "fix" and the flow remains a saturating s^*-t^* max-flow after each modification is carried out, we can be sure that modifying edges one-by-one in this fashion results in a feasible saturating s^*-t^* flow with all edges whose flow had dropped to 0 in $\overline{x'}$ removed. Note that reducing the flow along some path p may cause the flow on a different dead edge to drop to 0; if at any point we consider a dead edge whose flow has already been decreased in this way, we can simply remove it from the graph with no further changes, as the flow that remains is unaffected. Therefore, to remove i dead edges from G', at most i augmenting paths must be calculated.

If all i new non-intermediate edges in $\overline{x'}$ have become tight, we can modify G' as previously described for all such newly-tight edges at once, and then repeatedly find augmenting paths from s^* to t^* until a new saturating s^*-t^* flow is obtained (and in turn, a new elementary k-flow $\overline{y'}$). Because newly-tight edges must have previously had no flow going through them in G', there is no need to reduce the flow along any paths in G' before modifying the graph and augmenting; we simply add one unit capacity edge leaving s^* and one unit capacity edge entering t^* for each newly-tight edge in G. Therefore, if i edges become tight after calculating $\overline{x'}$, i new unit capacity edges are added to G' that leave s^*, so exactly i augmentations are necessary to find a new saturating s^*-t^* flow.

In the case where several edges become tight and several edges have their flow drop to 0 in $\overline{x'}$, we first remove the tight edges in G', add all appropriate edges leaving s^* and entering t^*, and calculate a new s^*-t^* max-flow as described above. If, while augmenting the old flow $\overline{f}$ in G' after modifying the graph for each tight edge, an edge (u, v) that had it's flow in $\overline{x'}$ drop to 0 in the last iteration is de-augmented (i.e., an augmenting path is found that takes the reverse edge (v, u) in the residual graph), then (u, v) has already had it's new flow reduced to 0 in $\overline{f}$, and therefore it can simply be removed from the graph. All remaining edges that have had their flow in $\overline{x'}$ drop to 0 but still carry 1 unit of flow in $\overline{f}$ after fixing each tight edge can then be corrected one-by-one, as described above. The flow $\overline{f'}$ resulting from modifying $\overline{f}$ in these ways can then be used to find the next elementary k-flow $\overline{y'}$, as it is a valid saturating s^*-t^* flow in G' that accounts for all edges that became tight or dead during the last iteration. Therefore, if i total edges either become dead or tight in $\overline{x'}$, at most i augmenting paths must be calculated in G'. $\square$

2.1.4 Running Time Analysis

To analyze the overall running time of the k-route flow decomposition algorithm outlined in this section (therefore proving Theorem 2), we consider an s-t flow $\overline{x}$ of kv total units in a graph $G = (V, E)$ with $0 \leq x_e \leq v$ for all edges $e \in E$. Let $m = |E|$ and $n = |V|$. To begin decomposing $\overline{x}$, we must first find an initial elementary k-flow by solving an instance of the Lower-Bounded Circulation Problem, as discussed in Section 2.1.1. To do so, we form an auxiliary graph $G' = (V', E')$ in $O(m + n)$ time, with new source and sink nodes s^* and t^*, respectively. Based on the construction of G', we have $|E'| = O(|E|)$ and $|V'| = O(|V|)$.

As outlined in Section 2.1.3, once the first elementary k-flow has been computed, subsequent elementary flows can be found by computing individual augmenting paths in the modified flow network G', each in $O(m + n)$ time, rather than calculating a new maximum flow from scratch. By Lemma 9, if i edges become either dead or tight during some iteration of the algorithm, a new flow can be caluculated with at most i augmenting path computations. Therefore, because G' has $O(m)$ edges, after the initial unit capacity max-flow is found, calculating all subsequent flows requires $O(m(m+n)) = O(m^2)$ time. Accounting for the time required to construct G' and to find an intial s^*-t^* unit capacity max-flow, this gives the k-route flow decomposition algorithm presented in this chapter an overall running time of $T(n, m) + O(m^2)$, (where $T(n, m)$ is the time required to compute a maximum flow in a unit capacity graph with n vertices and m edges), thus proving Theorem 2.

As mentioned before, the results of [6] show that a unit capacity maximum flow in a directed graph with n vertices and m edges (allowing parallel edges) can be calculated in $\tilde{O}(m^{10/7})$ time, giving the exact algorithm presented in this section an overall runtime of $O(m^2)$ by Theorem 2.

2.2 Approximation Algorithm

If we are willing to lose a small fraction of the value of the starting flow $\overline{x}$ when decomposing it into elementary k-flows, we can significantly improve the performance guarantee of the algorithm described in the previous section with only a slight loss in the quality of the resulting solution. A simple preprocessing step on $\overline{x}$ is the basis of this approximate decomposition algorithm. The basic idea behind the preprocessing step is to modify $\overline{x}$ to create a new flow $\overline{x'}$ where the flow on each edge in $\overline{x'}$ is a multiple of some constant γ. The following lemma establishes the benefits of creating $\overline{x'}$ prior to decomposing it into a k-route flow.

Lemma 10. *Let $\overline{x}$ be a feasible acyclic s-t flow in a graph $G = (V, E)$ with kv total units of flow entering t, for some integer k, such that $0 \leq x_e \leq v$ for all edges $e \in E$. If there exists some constant γ with $0 \leq \gamma \leq v$ such that for all $e \in E$, $x_e = \gamma \cdot i$ for some integer i, and $v = \gamma \cdot h$ for some integer $h \geq i$, then $\overline{x}$ can be decomposed into a k-route flow by calculating at most h unit capacity maximum flow computations in a graph with $O(|V|)$ vertices and $O(|E|)$ edges.*

Proof. We prove the claim by induction on h, the integer such that $v = \gamma \cdot h$. In the case where $h = 1$, each edge e has a flow x_e of either 0 or γ, and $v = \gamma$. Because in this case no intermediate edges exist, the elementary k-flow $\overline{y}$ we find in this iteration must include all edges e with $x_e = \gamma$. Recall that the decomposition algorithm chooses $\Delta = \min\{\Delta_1, \Delta_2\}$, where $\Delta_1 = \min\{v - x_e \mid y_e = 0\}$ and $\Delta_2 = \min\{x_e \mid y_e = 1\}$. Here, we set $\Delta = \gamma$, so the new flow $\overline{x'} = \overline{x} - \Delta\overline{y}$ has flow $x'_e = 0$ on all edges e, and the algorithm terminates. Therefore, only a single maximum flow calculation is needed in this case.

When $h > 1$, suppose for all integers $h' < h$ that if some s-t flow $\overline{x}$ has for each edge $e \in E$ a flow of $x_e = \gamma \cdot i$ for some integer $1 \leq i \leq h'$, and $v = \gamma \cdot h'$, then $\overline{x}$ can be decomposed into a k-route flow by calculating at most h' unit capacity maximum flows. The decomposition algorithm finds an elementary k-flow $\overline{y}$ such that $y_e = 1$ for all edges e with $x_e = v = \gamma \cdot h$, and then $\overline{x}$ is reduced by $\Delta\overline{y}$. Because x_e is a multiple of γ for all edges e and $v = \gamma \cdot h$, note that Δ must also be a multiple of γ. Therefore, in the flow $\overline{x'} = \overline{x} - \Delta\overline{y}$, x'_e is a multiple of γ for all edges $e \in E$; similarly, $v' = v - \Delta$ must also be a multiple of γ. We also have $v' < v = \gamma \cdot h$ in the new flow $\overline{x'}$. Therefore, there exists some integer $h' < h$ such that for all edges e, $x'_e = \gamma \cdot i$ for some integer $1 \leq i \leq h'$, and $v' = \gamma \cdot h'$. $\overline{x'}$ can then be decomposed into a k-route flow by calculating at most h' unit capacity maximum flows by the inductive hypothesis. Because a single maximum flow calculation was necessary to determine $\overline{y}$ in the current iteration, at most $h' + 1 \leq h$ maximum flow calculations in a unit capacity graph are required to decompose $\overline{x}$. $\square$

By Lemma 10, we are ensured that the number of maximum flow calculations required to find a k-route flow decomposition of a flow $\overline{x}$ can be greatly reduced if a modified flow $\overline{x'}$ can be found such that there exists some constant γ where the following properties hold:

1. $\overline{x'}$ sends kv total units of flow from s to t
2. $0 \leq x'_e \leq v$ for each edge $e \in E$
3. The flow x'_e on each edge $e \in E$ is an integer multiple of γ
4. $v = \gamma \cdot h$ for some integer h.

In particular, using Lemma 10, we will prove the following theorem in this section. For simplicity, we assume without loss of generality that $v = 1$ in the following analysis.

Theorem 3. *Given an acyclic s-t flow $\overline{x}$ in a directed graph $G = (V, E)$ sending k units of flow from s to t such that $0 \leq x_e \leq 1$ for all edges $e \in E$, for any sufficiently small $\epsilon > 0$, $\overline{x}$ can be decomposed with high probability into consituent elementary k-flows of total value at least $(1-2\epsilon)k$ in $O(nm+(k^2 \log n/\epsilon^2) \cdot T(n,m))$ time, where $n = |V|$, $m = |E|$, and $T(n,m)$ is the time required to compute a maximum s-t flow in a unit capacity graph with n vertices and m edges.*

2.2.1 Preprocessing Algorithm

Assume that we have assembled a set $\mathcal{P}$ of s-t paths by decomposing a starting flow $\overline{x}$ into paths and cycles via a standard flow decomposition algorithm in $O(nm)$ time. To create the modified flow $\overline{x'}$ described above from $\overline{x}$, we begin by grouping the paths in $\mathcal{P}$ into p bundles, each bundle carrying a total flow of exactly $\frac{1}{L}$, where $p = kL$ and L is a parameter to be determined later. Note that the combined flow from all such bundles will be exactly k.

More precisely, we create each bundle B by greedily stepping through each path $P \in \mathcal{P}$ in order, adding P to the bundle with the maximum possible flow while ensuring the total flow of all paths in B does not exceed $\frac{1}{L}$. If the flow on a path P is reduced to 0 and B still carries less than $\frac{1}{L}$ total units of flow, the next path in $\mathcal{P}$ is considered; if the total flow of all paths in B reaches $\frac{1}{L}$ before the flow on a path P has been reduced to 0, P will be the first path added to the next bundle created. In this way, each path in $\mathcal{P}$ may be used in multiple bundles (in addition, note that the paths in $\mathcal{P}$ are not necessarily edge-disjoint).

Define $\mathcal{B} = \{B_1, B_2, \cdots, B_p\}$ to be the set of all bundles created from the paths in $\mathcal{P}$, and for each bundle $B_i \in \mathcal{B}$ and each path $P \in B_i$, define $y_{i,P}$ as the flow on P within B_i. Note that by construction, for any bundle B_i and edge e, we have

$$\sum_{P \in B_i : e \in P} y_{i,P} \leq \frac{1}{L}.$$

To create the new flow $\overline{x'}$, we begin with a flow of $x'_e = 0$ for each edge $e \in E$, and process each of the bundles in $\mathcal{B}$ one-by-one. For each bundle $B_i \in \mathcal{B}$, we randomly select a single path P from B_i with probability $L \cdot y_{i,P}$, and increase the flow along the edges of P in $\overline{x'}$ by exactly $\frac{1}{L}$ units. After completing this process, because the same path may appear in multiple bundles, $\overline{x'}$ may send more than 1 unit of flow along some edges, but we are ensured that for each edge $e \in E$, the flow x'_e is a multiple of $\frac{1}{L}$.

The following lemma shows that if L is chosen correctly, then with high probability, the randomized rounding scheme described above produces a flow $\overline{x'}$ such that the flow x'_e on each edge $e \in E$ does not exceed 1 by more than ϵ for some small $\epsilon > 0$.

Lemma 11. *For sufficiently small $\epsilon > 0$, there exists a value of L such that with high probability, the flow x'_e on each edge $e \in E$ is at most $(1 + \epsilon)$.*

Proof. Let $0 \leq \epsilon < 1$. Fix any edge $e \in E$, and for each bundle $B_i \in \mathcal{B}$, define an independent 0-1 random variable $X_i \in [0, 1]$ such that $X_i = 1$ if the randomized rounding step added $1/L$ flow to a path in bundle B_i that contained e, and 0 otherwise. Let $X = \sum_i X_i$, and let $\mu = E[X]$. Because the total flow on any edge in the original flow $\overline{x}$ is no more than 1, we have

$$\mu = \sum_i \left(\sum_{P \in B_i : e \in P} y_{i,P} \cdot L \right) = L \cdot \sum_i \sum_{P \in B_i : e \in P} y_{i,P} \leq L.$$

Let $L = c \log n/\epsilon^2$ for some constant c to be determined later. By Theorem 1 and Lemma 4, we then have

$$\Pr[X \geq (1+\epsilon)(c \log n/\epsilon^2)] < e^{-(c \log n/\epsilon^2)\epsilon^2/3} = e^{-c \log n/3} = \frac{1}{n^{c/3}}.$$

Consider now scaling each variable X_i to create a new set of scaled variables; for each X_i, define $Y_i = X_i \cdot \frac{1}{L}$, and let $Y_e = \sum_i Y_i$, and $\mu' = E[Y_e]$. Note that Y_i now denotes the flow added to edge e during the randomized rounding step on bundle B_i (with value either 0 or $1/L$). As above, we have a simple upper bound on μ':

$$\mu' = \sum_i \left(\sum_{P \in B_i : e \in P} y_{i,P} \cdot L \right) \cdot \frac{1}{L} = \sum_i \sum_{P \in B_i : e \in P} y_{i,P} \leq 1.$$

Keeping $L = c \log n/\epsilon^2$, we note that $Y_i \in [0, 1/L]$. By Corollary 1, we then have

$$\Pr[Y_e \geq (1+\epsilon) \cdot 1] < e^{-\epsilon^2/(3/L)} = e^{-c \log n/3} = \frac{1}{n^{c/3}}.$$

Y_e denotes the total flow placed on a single edge $e \in E$ after the randomized rounding step has been completed for all bundles in $\mathcal{B}$; we must now consider the probability that any edge in E was given flow greater than $(1+\epsilon)$. There are at most n^2 edges, so we have

$$\begin{aligned} \Pr\left[\max_{e \in E} Y_e \geq (1+\epsilon)\right] &\leq \sum_{e \in E} \Pr[Y_e \geq (1+\epsilon)] \\ &< n^2 \cdot \frac{1}{n^{c/3}} = n^{2-c/3}. \end{aligned}$$

Setting $c \geq 12$ gives a probability of no more than $\frac{1}{n^2}$ that the statement fails to hold; we can make this probability arbitrarily small by increasing c. □

Lemma 11 establishes that by choosing $L = 12 \log n/\epsilon^2$ for some small $\epsilon > 0$, the randomized rounding scheme discussed above returns (with high probability) a flow $\overline{x'}$ where the flow x'_e on each edge $e \in E$ is a multiple of $1/L = \epsilon^2/12 \log n$ and does not exceed $(1+\epsilon)$. While $\overline{x'}$ carries an integer multiple of $1/L$ flow along each edge, as originally desired to make use of Lemma 10, because some edges may carry a flow of greater than 1, $\overline{x'}$ may not meet the properties necessary to be a k-route flow.

Noting that $\overline{x'}$ carries exactly k units of flow from s to t, the worst case (as illustrated in Figure 2.1) occurs when as many s-t paths as possible carry a flow of $(1+\epsilon)$. In particular, we consider $(k-1)$ paths carrying $(1+\epsilon)$ flow each, with a final path carrying exactly $1-\epsilon\cdot(k-1)$ flow.

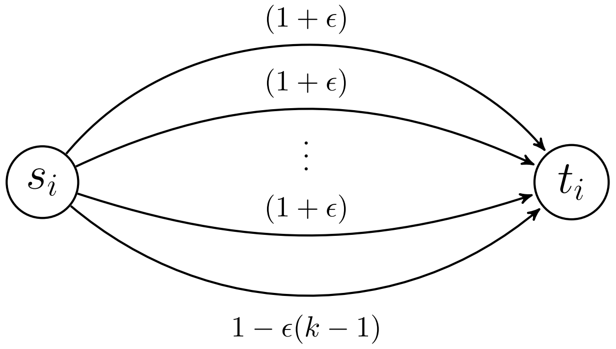

Figure 2.1: In the worst case, the randomized rounding technique described above yeilds a flow with $(1+\epsilon)$ flow on exactly $(k-1)$ edge-disjoint s-t paths, and $1-\epsilon\cdot(k-1)$ flow on one final s-t path (note that in the above figure, paths are depicted as single edges).

To recover a proper k-route flow in this case, we add a single dummy edge e' from s to t carrying a flow of value $\alpha(\frac{1}{L}) \geq \epsilon k$, where α is integer. Setting $\alpha = \lceil 12k\log n/\epsilon \rceil$ ensures the flow on this dummy edge is at least ϵk, while remaining an integer multiple of $\frac{1}{L}$; noting that this choice of α ensures that $\alpha(\frac{1}{L}) \leq \epsilon k + \frac{1}{L}$, we assume ϵ is chosen to be sufficiently small such that $\epsilon k + \frac{1}{L} \leq 1+\epsilon$.

The total s-t flow is now at least $k+\epsilon k = k(1+\epsilon)$ units, all edges carry a flow of no more than $(1+\epsilon)$ units, and the flow on each edge is guaranteed to be an integer multiple of $\frac{1}{L}$, so this newly created flow can be decomposed into elementary k-flows by calculating at most $(1+\epsilon)/(\frac{1}{L}) = (1+\epsilon)L$ maximum flows (by Lemma 10). Once such a decomposition has been found, removing from it all elementary k-flows that contain the dummy edge e' yeilds a new flow of value at least $k(1+\epsilon)-k(\frac{\alpha}{L}) \geq k(1+\epsilon)-k(\epsilon k+\frac{1}{L}) = k(1-\frac{1}{L}-\epsilon(k-1))$. Choosing $\epsilon = \epsilon'/(k-1)$ for some $\epsilon' > 0$ then allows us to recover a total of $k(1-\epsilon'-\frac{1}{L}) \geq k(1-2\epsilon')$ flow.

2.2.2 Runtime Analysis

Let $n = |V|$ and $m = |E|$. Based on the analysis of the preprocessing algorithm presented in Section 2.2.1, the valid k-route flow created using the bundling techniques introduced there can be decomposed into constituent elementary k-flows by calculating at most $(1+\epsilon)L$ maximum flows, where $L = 12\log n/\epsilon^2$. The analysis is completed by choosing $\epsilon = \epsilon'/(k-1)$ for some $\epsilon' > 0$, meaning a total of

$$(1+\epsilon)L = L + \epsilon L = \frac{12\log n}{(\epsilon'/(k-1))^2} + \frac{12\log n}{(\epsilon'/(k-1))} = O(k^2\log n/\epsilon'^2)$$

maximum flows are required to fully decompose the flow resulting from the preprocessing algorithm. Accounting for the time required to decompose the initial flow into paths and cycles, this gives the algorithm an overall runtime of $O(nm+(k^2\log n/\epsilon'^2)\cdot T(n,m))$ (where $T(n,m)$ is the time required to compute a single unit-capacity maximum flow), therefore establishing Theorem 3.

CHAPTER 3

APPLICATIONS TO MINIMUM CONGESTION ROUTING PROBLEM

In the Minimum Congestion Routing Problem, we are given a directed network $G = (V, E)$ along with l *commodities*, each of which consists of a single pair of vertices (s_i, t_i) in G. For each commodity (s_i, t_i), the node s_i is referred to as the *source* node, and the node t_i is referred to as the *sink* node. Given a set of paths $\mathcal{P} = \{P_1, P_2, \cdots, P_l\}$ in a directed graph, the value of the *congestion* created by the paths in $\mathcal{P}$ is defined formally as $\max_{e\in E} |\{i \mid P_i \in \mathcal{P}, e \in P_i\}|$ (in other words, the maximum over all edges $e \in E$ of the number of paths in $\mathcal{P}$ that contain e). In its simplest form, the goal of the Minimum Congestion Routing Problem is to find a single (s_i, t_i) path in G for each commodity $i \in [l]$ such that the congestion resulting from the chosen paths is minimized.

While the problem is similar in nature to a network flow problem, the restriction that only a single path is desired for each s_i-t_i pair means that a valid solution is not guaranteed by simply obtaining a flow from s_i to t_i for each routing request $i \in [l]$, as such a flow may be routed along multiple paths. In some variants of the problem, a capacity $c_e > 0$ is also given for each edge $e \in E$, with the goal being to minimize the *relative congestion* of the set $\mathcal{P}$ of chosen source-sink paths, defined as $\max_{e\in E}(1/c_e)\cdot|\{i \mid P_i \in \mathcal{P}, e \in P_i\}|$. For our purposes, we will assume $c_e = 1$ for all edges $e \in E$ when considering the problem in this manner, meaning for any set $\mathcal{P}$ of paths chosen between the given commodities, the value of the congestion and the relative congestion are equivalent.

3.1 Single Path Routing

Early applications of randomized rounding, such as those in [8], considered the classic variant of the problem described above where only a single path between each source-sink pair is desired. [8] assumes paths between each s_i-t_i pair are implicit, meaning the number of potential paths between any such pair is exponential; for this reason, the authors first solve the following simple integer programming (IP) formulation of the problem using edge variables $x_{i,e}$ for each edge $e \in E$ and for each routing request $i \in [l]$, where $x_{i,e} = 1$ if e lies along the path chosen from s_i to t_i, and 0 otherwise.

$$\begin{array}{lllll}
\text{minimize} & C & & & \\
\text{subject to} & \displaystyle\sum_{i=1}^{l} & x_{i,e} \leq C, & & e \in E \\
& \displaystyle\sum_{e=(w,v)\in E} x_{i,e} - & \displaystyle\sum_{e=(v,w)\in E} & x_{i,e} = 0, & v \in V \setminus \{s_i, t_i\}, i = 1, \ldots, l \\
& \displaystyle\sum_{e=(s_i,v)\in E} x_{i,e} - & \displaystyle\sum_{e=(v,s_i)\in E} & x_{i,e} = 1, & i = 1, \ldots, l \\
& & x_{i,e} \in \{0,1\}, & & e \in E, i = 1, ..., l
\end{array}$$

Solving the linear programming (LP) relaxation of this IP gives an optimal solution $\overline{x^*}$ of value C^*, where for each edge $e \in E$, $\sum_{i=1}^{l} x^*_{i,e} \leq C^*$; note that C^* naturally provides a lower bound for the value of the optimal solution of the original IP. To perform randomized rounding, the fractional edge-based flows obtained by solving the linear relaxation of the IP are converted into path-based flows for each s_i-t_i pair using a standard flow decomposition algorithm, such as the one found in [1], in $O(nm)$ time.

For each $i \in [l]$, this gives a collection $\mathcal{P}_i = \{p_{i,1}, p_{i,2}, \cdots, p_{i,N}\}$ of s_i-t_i paths, with each path $p_{i,j} \in \mathcal{P}_i$ assigned a fractional weight $y_{i,j}$; note that the sum of the weights of the paths in $\mathcal{P}_i$ is 1. For each $i \in [l]$, let $\overline{y_i} = \{y_{i,1}, y_{i,2}, \ldots, y_{i,N}\}$ be the vector holding the fractional weights on the paths in $\mathcal{P}_i$. Randomized rounding is performed by selecting a single path for each $i \in [l]$ based on the probability distribution $\overline{y_i}$ over the paths in $\mathcal{P}_i$.

Simple arguments using the Chernoff bounds of Theorem 1 and Lemma 4 give provably good approximation guarantees for the congestion obtained using this rounding scheme. Let n be the number of vertices in G.

Lemma 12. *If $C^* \geq 1$ and n is sufficiently large, then with high proba-*

bility, the congestion resulting from the randomized rounding algorithm is $O(\log n/\log\log n)\cdot C^*$.

Proof. For each edge $e \in E$, define X_e^i to be a random $0-1$ variable such that $X_e^i = 1$ if e lies on the s_i-t_i path chosen by the above randomized rounding scheme, and 0 otherwise. Let $Y_e = \sum_{i\in[l]} X_e^i$ be the total number of paths using edge e. Note that

$$E[Y_e] = \sum_{i\in[l]} E[X_e^i] = \sum_{i\in[l]} \sum_{P_{i,j}\in\mathcal{P}_i : e\in P_{i,j}} y_{i,j} = \sum_{P_{i,j}:e\in P_{i,j}} y_{i,j} = \sum_{i\in[l]} x_{i,e}^* \le C^*,$$

from the first constraint in the LP. For any edge $e \in E$, the variables X_e^i are independent for all $i \in [l]$; therefore, the bound in Theorem 1 applies. Choose δ such that $(1+\delta) = \frac{c\ln n}{\ln\ln n}$ for some constant c that will be determined later. Assume $n > e$ so that $\ln\ln n - \ln\ln\ln n > 0.5\ln\ln n$. By letting $\mu = C^* \ge 1$ in Theorem 1, we then have

$$\begin{aligned}
\Pr[Y_e \ge (1+\delta)C^*] &< \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{C^*} \\
&\le \frac{e^\delta}{(1+\delta)^{(1+\delta)}} \qquad (*)\\
&\le \frac{e^{(1+\delta)}}{(1+\delta)^{(1+\delta)}} \\
&= \left(\frac{c\ln n}{e\ln\ln n}\right)^{(-c\ln n/\ln\ln n)} \\
&= \exp((\ln c/e + \ln\ln n - \ln\ln\ln n)(-c\ln n/\ln\ln n)) \\
&\le \exp(0.5\ln\ln n(-c\ln n/\ln\ln n)) \\
&\le \frac{1}{n^{c/2}}
\end{aligned}$$

There are at most n^2 edges, so by the union bound, we have

$$\begin{aligned}
\Pr\left[\max_{e\in E} Y_e \ge (1+\delta)C^*\right] &\le \sum_{e\in E} \Pr[Y_e \ge (1+\delta)C^*] \\
&\le n^2 \cdot \frac{1}{n^{c/2}} = n^{2-c/2}.
\end{aligned}$$

Choosing $c = 8$ makes the claim fail to hold with probability at most $\frac{1}{n^2}$, and ensures that the inequality marked with (*) above is true (this choice of

c ensures that the value of the expression within parenthesis is less than 1). This probability can be made arbitrarily small by increasing c. Because

$$(1+\delta)C^* = (c\ln n/\ln\ln n)C^* = O(\log n/\log\log n)\cdot C^*,$$

this completes the proof. $\square$

Lemma 13. *If $C^* \geq 1$, then for any δ with $0 \leq \delta \leq 1$, there exists some constant $c > 0$ such that the congestion resulting from the randomized rounding algorithm is no more than $(1+\delta)\big[C^* + 3c\log n/\delta^2\big]$ with high probability.*

Proof. As before, for each edge $e \in E$, define X_e^i to be a random $0-1$ variable such that $X_e^i = 1$ if e lies on the s_i-t_i path chosen by the above randomized rounding scheme, and 0 otherwise. Let $Y_e = \sum_{i\in[l]} X_e^i$ be the total number of paths using edge e. Note that

$$E[Y_e] = \sum_{i\in[l]} E[X_e^i] = \sum_{i\in[l]}\ \sum_{P_{i,j}\in\mathcal{P}_i : e\in P_{i,j}} y_{i,j} = \sum_{P_{i,j}:e\in P_{i,j}} y_{i,j} = \sum_{i\in[l]} x_{i,e}^* \leq C^*,$$

from the first constraint in the LP. For any edge $e \in E$, the variables X_e^i are independent for all $i \in [l]$; assuming $0 \leq \delta \leq 1$, we can apply Lemma 4. Because the lemma holds for all $\mu \geq E[Y_e]$ and we know $E[Y_e] \leq C^*$, we can choose a value of μ such that $\mu \geq C^*$. $3c\log n/\delta^2 \geq 0$ for $c > 0$, so letting $\mu = (C^* + 3c\log n/\delta^2)$ gives us

$$\begin{aligned}\Pr[Y_e \geq (1+\delta)\mu] &= \Pr\left[Y_e \geq (1+\delta)[C^* + 3c\log n/\delta^2]\right] \\ &< \exp\left[(-\delta^2/3)(C^* + 3c\log n/\delta^2)\right] \\ &\leq \exp(-c\log n) \leq \frac{1}{n^c}\end{aligned}$$

Taking the union bound over at most n^2 edges gives us

$$\begin{aligned}&\Pr\left[\max_{e\in E} Y_e \geq (1+\delta)\big[C^* + 3c\log n/\delta^2\big]\right] \\ &\qquad\qquad \leq \sum_{e\in E} \Pr\left[Y_e \geq (1+\delta)[C^* + 3c\log n/\delta^2]\right] \\ &\qquad\qquad \leq n^2\cdot\frac{1}{n^c} = n^{2-c}\end{aligned}$$

Setting $c = 4$ makes the claim fail to hold with probability at most $\frac{1}{n^2}$. $\square$

If C^* is large enough, a simple argument using the Chernoff bounds of Theorem 1 and Lemma 4 gives a 2-approximation for the congestion obtained using the randomized rounding scheme.

Lemma 14. *If $C^* \geq c \ln n$ for some constant c, then with high probability, the resulting congestion is at most $C^* + \sqrt{C^*(c \ln n)}$.*

Proof. For each edge $e \in E$, define X_e^i to be a random $0-1$ variable such that $X_e^i = 1$ if e lies on the s_i-t_i path chosen by the above randomized rounding scheme, and 0 otherwise. Let $Y_e = \sum_{i \in [l]} X_e^i$ be the total number of paths using edge e. Note that

$$E[Y_e] = \sum_{i \in [l]} E[X_e^i] = \sum_{i \in [l]} \sum_{P_{i,j} \in \mathcal{P}_i : e \in P_{i,j}} y_{i,j} = \sum_{P_{i,j} : e \in P_{i,j}} y_{i,j} = \sum_{i \in [l]} x_{i,e}^* \leq C^*,$$

from the first constraint in the LP. For any edge $e \in E$, the variables X_e^i are independent for all $i \in [l]$; therefore, the bound in Theorem 1 applies. Let $\delta = \sqrt{(c \ln n)/C^*}$. We have $C^* \geq c \ln n$, so $\delta \leq 1$. Therefore, setting $\mu = C^*$, by Theorem 1 and Lemma 4 we have

$$\Pr[Y_e \geq (1 + \delta)C^*] < e^{-C^* \delta^2/3} = e^{-(c \ln n)/3} = \frac{1}{n^{c/3}}.$$

Note that $(1 + \delta)C^* = C^* + \sqrt{C^*(c \ln n)}$. There are at most n^2 edges, so

$$\begin{aligned} \Pr\left[\max_{e \in E} Y_e \geq (1 + \delta)C^*\right] &\leq \sum_{e \in E} \Pr[Y_e \geq C^* + \sqrt{C^*(c \ln n)}] \\ &\leq n^2 \cdot \frac{1}{n^{c/3}} = n^{2-c/3}. \end{aligned}$$

Setting $c \geq 12$ gives a probability of no more than $\frac{1}{n^2}$ that the claim fails to hold; this probability can be made arbitrarily small by increasing c. □

Because $C^* \geq c \ln n$, the resulting congestion is at most $C^* + \sqrt{C^*(c \ln n)} \leq 2C^* \leq 2$ OPT.

3.2 Multiple Path Routing

Building off of the techniques introduced for the well-studied variant of the Minimum Congestion Routing Problem where only a single path is desired between each s_i-t_i pair, more general variants have been considered where for each commodity $i \in [l]$, the goal is to find k_i edge-disjoint paths between the source vertex s_i and the target vertex t_i, while still minimizing the overall congestion of all chosen paths; for simplicity, the value of k_i will be assumed to be equal for all s_i-t_i pairs from this point forward (we will refer to this common value as k). This small change in the problem's parameters leads the standard randomized rounding approach of [8] to fail, and seemingly requires more advanced approaches to obtain the same approximation bounds as in [8]. In Section 3.3, we will show that the k-route flow decomposition algorithm of Chapter 2 can be used to allow for a simple randomized rounding scheme similar to that of [8] while achieving the same approximation guarantees for the case where multiple edge-disjoint paths between each s_i-t_i pair are desired.

Edge-disjoint s_i-t_i paths given: In [9], the author assumes that a set $\mathcal{P}_i = \{p_{i,1}, p_{i,2}, \cdots, p_N\}$ of edge-disjoint paths are explicitly given for each s_i-t_i pair. This allows for a simple integer programming formulation of the problem using path variables $x_{i,j}$ for each path $p_{i,j} \in \mathcal{P}_i$; $x_{i,j} = 1$ if $p_{i,j}$ is chosen, and 0 if not. The integer program for this formulation of the problem is as follows:

$$\begin{array}{lll} \text{minimize} & C & \\ \text{subject to} & \displaystyle\sum_{j} x_{i,j} = k, & i = 1, ..., l \\ & \displaystyle\sum_{(i,j):e\in P_{i,j}} x_{i,j} \leq C, & e \in E \\ & x_{i,j} \in \{0,1\}, & i = 1, ..., l, p_{i,j} \in \mathcal{P}_i \end{array}$$

Solving the linear relaxation of this IP gives an optimal solution $\overline{x^*}$ of value C^*; for each $i \in [l]$, this gives a vector $\overline{v_i} = \{x^*_{i,1}, x^*_{i,2}, \cdots, x^*_{i,N}\}$ where $x^*_{i,j} \in [0,1]$ and $\sum_j x^*_{i,j} = k$. Randomized rounding is performed by selecting k paths between each s_i-t_i pair; independently for each i, a dependent rounding scheme is used that guarantees exactly k s_i-t_i paths are chosen, and that with high probability, the algorithm achieves the same bounds on overall congestion as in [8].

Note that if we are initially given a set of edge-disjoint paths between each s_i-t_i pair (as in [9]), after finding a fractional optimal solution $\overline{x^*}$ to the linear relaxation of the IP of [9], we can easily find a k-route flow decomposition of the flow encoded by the vector $\overline{v_i} = \{x^*_{i,1}, x^*_{i,2}, \cdots, x^*_{i,N}\}$ for each $i \in [l]$ as follows. Let h be the number of paths constituting the flow encoded by $\overline{v_i}$. Because all such paths are mutually edge-disjoint, we think of each as a single edge, and begin by sorting these edges by their fractional weight in $\overline{v_i}$ in $O(h \log h)$ time and storing each edge in a max-priority queue Q. We then greedily consider the top k paths from Q, reduce the flow on these paths equally until the flow on at least one path is reduced to zero, and add the resulting fractional elementary k-flow to the decomposition being constructed. Continuing this process, we are guaranteed that the algorithm will terminate with the flow on all paths being reduced to zero, as at least one path must have its flow reduced to zero during each iteration. Because v_i is made up of h edges, the algorithm terminates in $O(h)$ iterations, and because each iterataion requires updating the keys of k items in Q, the overall runtime of this method is $O(kh \log h)$.

No edge-disjoint s_i-t_i paths given: [10] also considers the problem in the setting where multiple edge-disjoint paths are desired, but as in [8], the authors assume no paths are known initially between each pair. Therefore, an IP similar to that of [8] is used with variables for each edge:

$$\begin{array}{lllll}
\text{minimize} & C & & & \\
\text{subject to} & \displaystyle\sum_{i=1}^{l} & x_{i,e} \leq C, & & e \in E \\
& \displaystyle\sum_{e=(w,v)\in E} x_{i,e} & - \displaystyle\sum_{e=(v,w)\in E} x_{i,e} = 0, & & v \in V \setminus \{s_i, t_i\}, i = 1, \dots, l \\
& \displaystyle\sum_{e=(s_i,v)\in E} x_{i,e} & - \displaystyle\sum_{e=(v,s_i)\in E} x_{i,e} = k, & & i = 1, \dots, l \\
& & x_{i,e} \in \{0,1\}, & & e \in E, i = 1, ..., l
\end{array}$$

Once again, the authors of [10] first solve the linear relaxation of the IP to obtain a fractionally optimal solution $\overline{x^*}$ of value C^*. Flow decomposition is then performed such that each s_i-t_i flow encoded in $\overline{x^*}$ is decomposed into a weighted sum $\overline{f_i} = \sum_{P \in \mathcal{P}_i} w_i^P f_i^P$, where $\mathcal{P}_i$ is a finite set of s_i-t_i paths, f_i^P is a flow of 1 unit along each edge of the path $P \in \mathcal{P}_i$, and $w_i^P \in [0,1]$ is a fractional weight assigned to P; note that $\sum_{P \in \mathcal{P}_i} w_i^P = k$.

Randomized rounding is then performed by rounding each fractional flow $w_i^P f_i^P$ to a new flow $y_i^P \in \{0, 1\}$ such that $\sum_{P \in \mathcal{P}_i} y_i^P = k$ for each $i \in [l]$. Intuitively, this seems to guarantee that $\overline{y_i}$ sends flow along exactly k edge-disjoint paths between each s_i-t_i pair, and the authors of [10] show using Chernoff-style bounds that the resulting congestion follows the bounds established in [8] and [9]. However, this rounding scheme does not account for the fact that for each $i \in [l]$, the paths in $\mathcal{P}_i$ found via flow decomposition are not necessarily edge-disjoint. This means that k paths will be selected between each s_i-t_i pair during the randomized rounding stage, but some of these paths may still share an edge; therefore, the paths returned by the randomized algorithm of [10] are not guaranteed to satisfy the constraints of the problem.

In the following section, we present an algorithm for the Minimum Congestion Routing Problem where $k \geq 1$ and no s_i-t_i paths are given (as considered in [10]) utilizing a different approach to that of [10] that hinges on the k-route flow decomposition algorithm discussed in Chapter 2.

3.3 Multiple Path Routing via k-route Flows

Our algorithm begins in the same way as in [10]; we solve the linear relaxation of the IP used there, with variables $x_{i,e}$ for each edge $e \in E$ and each $i \in [l]$, and obtain an optimal fractional solution $\overline{x^*}$ of value C^*. For each s_i-t_i pair, flow decomposition gives a finite set of s_i-t_i paths $\mathcal{P}_i$ and a flow $\overline{f_i} = \sum_{P \in \mathcal{P}_i} w_i^P f_i^P$, where each $f_i^P \in [0, 1]$ is a flow of 1 unit along a path $P \in \mathcal{P}_i$ and $w_i^P \in [0, 1]$ is a fractional weight. Recall that $\sum_{P \in \mathcal{P}_i} w_i^P = k$. In the linear relaxation of the IP, we have $x^*_{i,e} \leq 1$ for each edge $e \in E$ and each $i \in [l]$; therefore, for each edge e in the flow $\overline{f_i}$, we have $\sum_{P \in \mathcal{P}_i : e \in P} w_i^P \leq 1$. Therefore, for each $i \in [l]$, $\overline{f_i}$ is a flow of k units such that the flow on each edge is no more than 1. By Lemma 3, $\overline{f_i}$ is a k-route flow, and we can therefore use the decomposition algorithm of Section 2 to decompose it into k-tuples of edge-disjoint s_i-t_i paths (i.e., elementary k-flows).

Once $\overline{f_i}$ is found and decomposed into a k-route flow $\overline{f_i^*}$, we perform randomized rounding in a style similar to that of [8], but rather than selecting only a single path between each s_i-t_i pair, we use rounding to choose a single elementary k-flow made up of k edge-disjoint s_i-t_i paths. More formally, the k-route flow decomposition of $\overline{f_i}$ gives a weighted vector $\overline{v_i} = \{w_1^* f_1^*, w_2^* f_2^*, \cdots, w_{N_i}^* f_{N_i}^*\}$ where for each $j \in [N_i]$, $w_j^* \in [0,1]$ and f_j^* is an s_i-t_i flow of 1 unit along k edge-disjoint paths (i.e., an elementary k-flow from s_i to t_i); naturally, $\overline{f_i^*} = \sum_{j\in[N_i]} w_j^* f_j^*$. Randomized rounding is performed by selecting a single elementary k-flow from $\overline{v_i}$ by sampling from the probability distribution created by the weights $\{w_1^*, w_2^*, \cdots, w_{N_i}^*\}$. Chernoff bounds can then be used to show that the resulting congestion is relatively small (with high probability) using arguments similar to those of [8]. Let n be the number of vertices in G.

Lemma 15. *If $C^* \geq 1$ and n is sufficiently large, then with high probability, the congestion resulting from the randomized rounding algorithm is $O(\log n/\log\log n) \cdot C^*$.*

Proof. For each edge $e \in E$, define X_e^i to be a random $0-1$ variable such that $X_e^i = 1$ if e lies on one of the s_i-t_i paths making up the elementary k-flow chosen by the above randomized rounding scheme, and 0 otherwise. Let $Y_e = \sum_{i\in[l]} X_e^i$ be the total number of paths using edge e. Note that

$$E[Y_e] = \sum_{i\in[l]} E[X_e^i] = \sum_{i\in[l]} \sum_{j\in[N_i]:e\in f_j^*} w_j^* = \sum_{i\in[l]} x_{i,e}^* \leq C^*,$$

from the first constraint in the LP. For any edge $e \in E$, the variables X_e^i are independent for all $i \in [l]$; therefore, the bound in Theorem 1 applies. Choose δ such that $(1+\delta) = \frac{c \ln n}{\ln\ln n}$ for some constant c that will be determined later. Assume $n > e$ so that $\ln\ln n - \ln\ln\ln n > 0.5 \ln\ln n$. By letting $\mu = C^* \geq 1$

in Theorem 1, we then have

$$\begin{aligned}
\Pr[Y_e \geq (1+\delta)C^*] &< \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{C^*} \\
&\leq \frac{e^\delta}{(1+\delta)^{(1+\delta)}} \qquad (*) \\
&\leq \frac{e^{(1+\delta)}}{(1+\delta)^{(1+\delta)}} \\
&= \left(\frac{c\ln n}{e\ln\ln n}\right)^{(-c\ln n/\ln\ln n)} \\
&= \exp((\ln c/e + \ln\ln n - \ln\ln\ln n)(-c\ln n/\ln\ln n)) \\
&\leq \exp(0.5\ln\ln n(-c\ln n/\ln\ln n)) \\
&\leq \frac{1}{n^{c/2}}
\end{aligned}$$

There are at most n^2 edges, so by the union bound, we have

$$\begin{aligned}
\Pr\left[\max_{e\in E} Y_e \geq (1+\delta)C^*\right] &\leq \sum_{e\in E} \Pr[Y_e \geq (1+\delta)C^*] \\
&\leq n^2 \cdot \frac{1}{n^{c/2}} = n^{2-c/2}.
\end{aligned}$$

Choosing $c = 8$ makes the claim fail to hold with probability at most $\frac{1}{n^2}$, and ensures that the inequality marked with (*) above is true (this choice of c ensures that the value of the expression within parenthesis is less than 1). This probability can be made arbitrarily small by increasing c. Because

$$(1+\delta)C^* = (c\ln n/\ln\ln n)C^* = O(\log n/\log\log n)\cdot C^*,$$

this completes the proof. $\square$

Lemma 16. *If $C^* \geq 1$, then for any δ with $0 \leq \delta \leq 1$, there exists some constant $c > 0$ such that the congestion resulting from the randomized rounding algorithm is no more than $(1+\delta)\left[C^* + 3c\log n/\delta^2\right]$ with high probability.*

Proof. As before, for each edge $e \in E$, define X_e^i to be a random $0-1$ variable such that $X_e^i = 1$ if e lies on one of the s_i-t_i paths making up the elementary k-flow chosen by the above randomized rounding scheme, and 0 otherwise.

Let $Y_e = \sum_{i \in [l]} X_e^i$ be the total number of paths using edge e. Note that

$$E[Y_e] = \sum_{i \in [l]} E[X_e^i] = \sum_{i \in [l]} \sum_{j \in [N_i] : e \in f_j^*} w_j^* = \sum_{i \in [l]} x_{i,e}^* \leq C^*,$$

from the first constraint in the LP. For any edge $e \in E$, the variables X_e^i are independent for all $i \in [l]$; assuming $0 \leq \delta \leq 1$, we can apply Lemma 4. Because the lemma holds for all $\mu \geq E[Y_e]$ and we know $E[Y_e] \leq C^*$, we can choose a value of μ such that $\mu \geq C^*$. $3c \log n/\delta^2 \geq 0$ for $c > 0$, so letting $\mu = (C^* + 3c \log n/\delta^2)$ gives us

$$\begin{aligned} \Pr[Y_e \geq (1+\delta)\mu] &= \Pr\left[Y_e \geq (1+\delta)[C^* + 3c\log n/\delta^2]\right] \\ &< \exp\left[(-\delta^2/3)(C^* + 3c\log n/\delta^2)\right] \\ &\leq \exp(-c \log n) \leq \frac{1}{n^c} \end{aligned}$$

Taking the union bound over at most n^2 edges gives us

$$\begin{aligned} &\Pr\left[\max_{e \in E} Y_e \geq (1+\delta)\left[C^* + 3c\log n/\delta^2\right]\right] \\ &\qquad \leq \sum_{e \in E} \Pr\left[Y_e \geq (1+\delta)[C^* + 3c\log n/\delta^2]\right] \\ &\qquad \leq n^2 \cdot \frac{1}{n^c} = n^{2-c} \end{aligned}$$

Setting $c = 4$ makes the claim fail to hold with probability at most $\frac{1}{n^2}$. □

As in the single path variant of the problem, if C^* is large enough, a simple argument using the Chernoff bounds of Theorem 1 and Lemma 4 gives a 2-approximation for the congestion obtained using the randomized rounding scheme, matching the original results of [8] in the setting where $k \geq 1$.

Lemma 17. *If $C^* \geq c \ln n$ for some constant c, then with high probability, the resulting congestion is at most $C^* + \sqrt{C^*(c \ln n)}$.*

Proof. For each edge $e \in E$, define X_e^i to be a random $0-1$ variable such that $X_e^i = 1$ if e lies on one of the s_i-t_i paths making up the elementary k-flow chosen by the above randomized rounding scheme, and 0 otherwise.

Let $Y_e = \sum_{i\in[l]} X_e^i$ be the total number of paths using edge e. Note that

$$E[Y_e] = \sum_{i\in[l]} E[X_e^i] = \sum_{i\in[l]} \sum_{j\in[N_i]:e\in f_j^*} w_j^* = \sum_{i\in[l]} x_{i,e}^* \leq C^*,$$

from the first constraint in the LP. For any edge $e \in E$, the variables X_e^i are independent for all $i \in [l]$; therefore, the bound in Theorem 1 applies. Let $\delta = \sqrt{(c \ln n)/C^*}$. We have $C^* \geq c \ln n$, so $\delta \leq 1$. Therefore, setting $\mu = C^*$, by Theorem 1 and Lemma 4 we have

$$\Pr[Y_e \geq (1+\delta)C^*] < e^{-C^*\delta^2/3} = e^{-(c\ln n)/3} = \frac{1}{n^{c/3}}.$$

Note that $(1+\delta)C^* = C^* + \sqrt{C^*(c\ln n)}$. There are at most n^2 edges, so

$$\begin{aligned}\Pr\left[\max_{e\in E} Y_e \geq (1+\delta)C^*\right] &\leq \sum_{e\in E} \Pr[Y_e \geq C^* + \sqrt{C^*(c\ln n)}] \\ &\leq n^2 \cdot \frac{1}{n^{c/3}} = n^{2-c/3}.\end{aligned}$$

Setting $c \geq 12$ gives a probability of no more than $\frac{1}{n^2}$ that the claim fails to hold; this probability can be made arbitrarily small by increasing c. □

Because $C^* \geq c \ln n$, the resulting congestion is at most $C^* + \sqrt{C^*(c\ln n)} \leq 2C^* \leq 2$ OPT.

If we allow for a slight degradation in the bound on the congestion found by the randomized rounding algorithm proposed in this section, we can utilize the approximate k-route flow decomposition algorithm of Section 2.2. Recall that given a flow $\overline{x}$ of k total units with $0 \leq x_e \leq 1$ for all edges e, for any sufficiently small $\epsilon > 0$ the algorithm returns a k-route flow of value at least $(1-\epsilon)k$. Because the flow $\overline{f_i}$ obtained for each $i \in [l]$ (as described above) sends k units from s_i to t_i with $0 \leq f_{i,e} \leq 1$ for all $e \in E$, the approximate decomposition algorithm can be applied. Suppose the algorithm returns a flow of value $(1-\epsilon')k$ for some $\epsilon' \leq \epsilon$. By scaling the flow on each edge by a factor of $1/(1-\epsilon')$, we obtain a flow of k total units decomposed into k-tuples of edge-disjoint paths, and the resulting congestion (after performing the randomized rounding algorithm described above) increases by the same factor.

CHAPTER 4

IMPLEMENTATION AND EXPERIMENTS

To experimentally evaluate the effectiveness of the decomposition algorithms presented in Chapter 2, the simple decomposition algorithm of Lemma 6 and the optimized algorithm laid out in Section 2.1.3 were implemented and tested on a variety of flow instances. The goal of the performed testing, laid out in the body of this chapter, was to determine if the theoretical improvements laid out in Section 2.1.3 to the "naive" decomposition algorithm (where a complete maximum flow is calculated from scratch during each iteration) actually provide a meaningful improvement to the real-world runtime of the algorithm, as well as to obtain practical insights as to how the algorithms perform on different graph structures.

All experiments were performed on a consumer laptop equipped with an Intel Core i3-5005U quad-core processor (clocked at 2.00 GHz per core), 4GB of RAM, and running Gallium OS, a lightweight Linux distribution built off of the Ubuntu operating system. While the machine used for testing is fairly low-powered by modern standards, the purpose of the experimentation performed was to evaluate how the decomposition algorithms devised in Chapter 2 might perform in a real-world setting using an "everyday" consumer machine.

All algorithms were implemented using the LEMON C++ Graph Library [11], an open-source combinatorial optimization and graph modeling library providing straightforward and efficient implementations of a wide range of data structures and graph-based optimization algorithms. All code was compiled using the g++ compiler, version 5.4.0, and code was written in C++11. Several key aspects of the experiments presented in this chapter, laid out in the sections that follow, relied heavily on the simple tools provided by the LEMON library. In particular, the implementaion was divided into two distinct subsections: the generation of unique "starting flows" with the necessary characteristics to be classified as k-route flows (as laid out in Lemma 3), and the implementation of the naive and optimized versions of the k-route flow decomposition algorithms, as described in Chapter 2.

4.1 Flow Generation

Several different approaches were taken to generate flows with the necessary properties to be characterized as valid k-route flows, as described in Lemma 3. Recall that for a given value of k and v, this particular characterization defines a k-route flow $\overline{f}$ as a flow of kv total units such that $0 \leq f_e \leq v$ for all edges e. With this in mind, the general approach taken to generate starting flows of this type to be used for the purposes of experimentally testing the decomposition algorithms of Chapter 2 was to first generate a random graph or import an existing graph structure, and to then run a maximum flow algorithm with specific capacities on each edge with the goal of extracting a k-route flow with the necessary properties. For all generated flows, values of $k = 3$ and $v = 100$ were used, and all flows generated were integral. Unlike in previous chapters, flows generated using the following methods were not assumed to be acyclic, and no check for acyclicity was performed.

Flow Generation via d-Regular Random Graphs: An initial attempt at generating valid k-route flows began with the generation of d-regular random graphs, using an open-source random graph generator contributed to the LEMON library as a unofficial add-on module [12]. To ensure that the degree of each node in these randomly generated graphs was sufficiently large for the purpose of obtaining a valid k-route flow, d was chosen as an integer larger than k; tests were performed with values of d ranging from 4 to 10 (with $k = 3$ being fixed).

k-route flows were generated by utilizing the structure of these initially generated graphs (i.e., the resulting flows were created as subgraphs of the initial graphs); this technique was repeated in the subsequent flow generation methods explored, with the only change being the inherent structure of the graphs that the generation algorithm started with. More specifically, given a d-regular random graph G generated using the aforementioned LEMON module, a valid k-route flow $\overline{f}$ was generated as follows. First, a source node s was selected randomly in G, which would act as the source node in the generated flow $\overline{f}$; a target node t was randomly selected in a similar fashion. Two additional nodes, s^* and t^*, were then added to G, along with edges (s^*, s) and (t, t^*). A maximum flow instance was then generated by assigning a capacity of kv to the newly-added s^*-s and t-t^* edges, and a random capacity $c_e \leq v$ to each edge originally in G.

If a feasible flow of value kv (i.e., a flow saturating the added edges (s^*, s) and (t, t^*)) could be calculated between s^* and t^*, then removing s^* and t^* from G would yeild a flow of kv total units where the flow on each edge is no more than v (therefore satisfying the necessary conditions of a k-route flow, as defined in Lemma 3). In most tests, random capacities were assigned in the range $[\lfloor v/2 \rfloor, v]$ to improve the chances that the flow calculated did in fact saturate the added s^*-s and t-t^* edges (while a range of $[0, v]$ is another rather obvious choice, experimentation revealed that allowing for such low capacities on edges in G often resulted in flows of value significantly less than kv units).

Flows generated using this approach generally utilized d-regular random graphs that were initially large, ranging in size from 1,000,000 to 3,000,000 nodes, to ensure that k-route flows generated were large enough to gain useful insights when using the flows to test the efficiency of the k-route flow decomposition algorithms experimentally. Unfortunately, this initial approach to generating flows via the use of d-regular random graphs (even ones containing several million nodes) failed to generate flows containing more than a few thousand edges.

Several modifications were made to the setup of the maximum flow instance described above in an attempt to increase the size of the flows being generated, such as running a breadth-first search from the randomly selected source node s and selecting the target node t as the vertex positioned furthest from s in G, but such attempts did not significantly affect the size of the generated flows. One reason why this approach to generating starting flows (utilizing d-regular random graphs as an initial structure) may not have yeilded flows of sufficient size comes from a well-known graph-theoretic result showing that d-regular random graphs are known to have a small diameter [13]; this observation explains why even when using graphs initially constructed with several million nodes and edges, flows found within those graphs often only consisted of several thousand nodes and edges.

Flow Generation via Real-World Road Networks: Based on the desire to generate sufficiently large starting flows for use in experimentally testing the practical efficiency of the various decomposition algorithms presented in Chapter 2, attempts were made to generate flows using pre-existing graph structures, rather than through the initial generation of random graphs. One approach utilized the structure of real-world road networks by using publicly available datasets representing existing roads and highways spanning various geographical regions of the United States. The specific instances utilized come from the 9th DIMACS Implementation Challenge [14], which provided graph instances representing the road networks of various states and metropolitan areas generated from raw geographic data originating from the TIGER/Line dataset. DIMACS graph instances are represented in a specialized format specific to their series of implementation challenges, but the format is commonly used within experimental algorithmics, and LEMON provides simple tools to decode the DIMACS graph format into a usable graph structure.

DIMACS road network instances are undirected graphs represented using directed edges (i.e., each undirected edge in the network is represented as two directed edges). In addition to the nodes and edges of the graph, different versions of the dataset also include one of several costs associated with each edge, representing either the geographical distance between the two map points represented by the endpoints of the edge or the approximate travel time between those two locations. Because we hoped to generate a directed k-route flow using these graphs as a starting point, only the structure of the road network instances (the nodes and edges themselves) were utilized after being imported into LEMON, and the costs associated with each edge in the original dataset were ignored.

An instance representing the road network of the New York City metropolitan area was selected to be used as a starting graph for the purpose of generating valid k-route flows; the original graph contained 264,346 nodes and 733,846 directed edges. The method used to generate such flows was identical to the method employed previously when working with d-regular random graphs; two nodes s and t were randomly chosen as source and target nodes for the flow being generated, special nodes s^* and t^* were attached to s and t with edges assigned a capacity of kv units, and all other edges in the graph were assigned a random capacity in the range $[\lfloor v/2 \rfloor, v]$. One maximum flow algorithm was run from s^* to t^*, and if a flow was found that saturated the added s^*-s and t-t^* edges (meaning the flow from s to t was exactly kv units), then the added nodes s^* and t^* were removed from the graph (along with all adjacent edges) and the resulting flow was output to file. In order to keep the representation of each flow generated as succinct as possible, prior to outputting each generated flow, all edges carrying a flow value of 0 were removed, along with all nodes through which no flow was routed.

Unlike initial attempts using d-regular random graphs, the use of real-world road networks as initial graph structures for the purposes of flow generation proved to be highly successful in generating large k-route flow instances made up of edges numbering in the hundreds of thousands. One possible reason for the success of this method in generating large flows may be the fact that the DIMACS road network instances, being built from real-world map data, are inherently less structured than d-regular random graphs, potentially providing graph structures with higher diameters and forcing flow algorithms to route flow along more nodes and edges to obtain the same overall flow value. One may also note that the choice of the New York City metropolitan area provided a fairly dense starting graph, which may have helped to ensure that at least k edge-disjoint paths could be found between a given source and target node (something that may not have been ensured in sparser road networks representing more rural geographical areas).

Flow Generation via Grid-Based Graphs: To obtain starting flows with structural qualities differing from those of the flows generated from the real-world road networks discussed above, graphs containing grid-like structures were also utilized. The specific graph instance used came from an experimental paper published by one of the authors of the LEMON Graph Library [15], and was originally generated using the GRIDGRAPH generator [16]; the family of graph instances utilized in [15] were of particular interest, as they had already been converted into the native format used by LEMON to store graph data. GRIDGRAPH generates graphs with a single source node s, a single target node t, and a grid of nodes (the size of which can be specified prior to generation) lying between the source and the target. Directed edges leave from s to each node of the first column of the grid, and similarly, directed edges leave from each node in the last column of the grid to t. For each node (u, v) within the grid (with u denoting the row in the grid and v denoting the column), directed edges travel from (u, v) to nodes $(u+1, v)$ and $(u, v+1)$, except for nodes in the last row and column, respectively.

The specific grid-graph instance utilized from [15] consisted of a 1024x1024 grid of nodes, in addition to the source and sink nodes mentioned above. As was the case with the road networks used to generate k-route flows, the edges of the graph instances from [15] were associated with several capacities and costs, as the instances were originally generated for the experimental testing of minimum-cost flow algorithms. As before, such costs and capacities were ignored, and only the structure of the graph was utilized for the generation of flows. To aid in the generation of flows containing a large number of vertices and edges, connectivity was increased within the graph by attaching directed back edges between the nodes within the internal grid structure, aside from the source and target nodes, s and t. In other words, for each edge within the grid from a node (u, v) to a node $(u+1, v)$, a directed edge was added from $(u+1, v)$ to node (u, v); similar additions were made between nodes $(u, v+1)$ and (u, v).

Once the structure of the grid-based graph instance had been finalized, k-route flows were generated as before. First, special source and sink nodes s^* and t^* were added to the graph, and attached with a single edge of capacity kv to s and t, respectively. Then, each edge in the original graph was assigned a random capacity in the range $[\lfloor v/2 \rfloor, v]$, and a maximum flow algorithm was run from s^* to t^*. If the added s^*-s and t-t^* edges were properly saturated in the resulting flow, a valid k-route flow had been recovered, and after removing s^* and t^* (along with all nodes and edges through which no flow had been routed), the final generated flow was output to file.

4.2 Implementation of Decomposition Algorithms

Using LEMON, efficient C++ implementations were developed for both the naive decomposition algorithm of Lemma 6 (where a complete maximum flow is calculated during each iteration) and the optimized decomposition laid out in the remainder of Section 2.1.3 (where a new maximum flow is generated during each iteration by modifying a previously generated flow). In this section, we will review the built in tools and algorithms provided by LEMON that were utilized to complete the implementation of each of these variants of the exact decomposition algorithm.

Naive Algorithm Implementation: Due to the fairly simple structure of the naive k-route flow decomposition algorithm, the graph algorithms provided by the LEMON library allowed for a compact implementation spanning only several hundred lines of code. Recall the basic design of the naive algorithm, given a valid k-route flow $\overline{f}$ to be decomposed, with source node s and target node t. During each iteration, an elementary k-flow $\overline{y}$ is computed such that for each edge e with $f_e = 0$, $y_e = 0$, and for each edge e with $f_e = v$, $y_e = 1$. In Section 2.1.1, it is shown that a simple instance of the Lower-Bounded Circulation Problem can be generated to find such an elementary flow, and a feasible circulation meeting the lower and upper bound constraints on each edge can be calculated using a simple reduction to the Maximum Flow Problem.

Fortunately, such a reduction is not necessary using the tools provided by the LEMON library. LEMON contains an efficient implementation of an algorithm used to solve the Lower-Bounded Circulation Problem, built off of the Push-Relabel algorithm [17] used to solve the Maximum Flow Problem. Prior to the first iteration of the algorithm, a single back edge with lower and upper bounds equal to k is added from the target node t to the source node s, just as is done in Section 2.1.1. Similarly, for each iteration of the algorithm, each edge in the graph is cycled through and assigned lower and upper bounds of 0 or 1 to create a valid instance of the Lower-Bounded Circulation Problem, which can then be used to obtain an elementary k-flow with the desired properties. Once the bounds on each edge have been set at the start of each iteration, a new instance of LEMON's circulation algorithm is created, and a feasible circulation is aquired. At the end of each iteration, each edge is looped through once again to determine the correct value of Δ (as described in Lemma 3), and the starting flow $\overline{f}$ is updated accordingly to reflect the reduction in flow resulting from the calculation of a new elementary k-flow (via the calculation of a feasible circulation).

To detect when the decomposition algorithm has properly terminated, a loop is included at the end of each iteration to ensure that edges remain in the starting flow $\overline{f}$ that still hold non-zero flow (implying that additional decomposition is still required); when no such edges exist, the flow has been fully decomposed into constituent elementary k-flows, and the algorithm terminates.

Optimized Algorithm Implementation: Unlike the relatively straightforward implementation used for the naive decomposition algorithm described above, the optimized version of the algorithm required a more fine-tuned approach, as potentially many specific graph modifications and flow augmentations were required during each iteration. Recall the structure of the optimized decomposition algorithm. Given a valid k-route flow $\overline{f}$ to be decomposed, along with a source node s and a target node t, the algorithm first computes an initial elementary k-flow by running a single unit-capacity maximum flow algorithm (following the standard reduction from the Lower-Bounded Circulation Problem outlined in Section 2.1.1). For each subsequent iteration, rather than recomputing a new maximum flow from scratch (as is done in the naive decomposition algorithm), the flow from the last iteration is kept track of, and simple updates are performed to account for edges that must now be included (or removed) to generate a new elementary k-flow satisfying the conditions laid out in Lemma 2.

Because small local modifications are required to the unit-capacity maximum flow stored by the algorithm during each iteration, it was no longer possible to simply create a new instance of the Lower-Bounded Circulation Problem for each new elementary k-flow calculated, as was done in the implementation of the naive decomposition algorithm. Rather, a unit-capacity maximum flow instance was created and modified throughout the course of the algorithm, with the new elementary k-flow generated at the end of each iteration of the algorithm formed implicitely from this unit-capacity maximum flow instance (recall that valid elementary k-flows can be generated from such maximum flows by ignoring the added "super-source" and "super-sink" nodes and assigning the proper flow to each tight edge in the graph, as outlined in Section 2.1.1).

With this in mind, the optimized k-route flow decomposition algorithm was implemented as follows. Given a k-route flow $\overline{f}$ to be decomposed (along with a source node s and a target node t), an initial unit-capacity maximum flow $\overline{x}$ was calculated following the reduction laid out in Section 2.1.1; "super-source" and "super-sink" nodes s^* and t^*, respectively, were added to the graph, along with edges (s^*, s) and (t, t^*) (each with capacity k), and for each tight edge $e = (u, v)$ (with $f_e = v$), e was removed by setting its capacity to

0 and edges (s^*, v) and (u, t^*) were added, each with capacity 1. All edges e with flow $f_e = 0$ were also implicitely removed from the graph by having their capacities set to 0 (ensuring they would not be assigned flow in the soon-to-be computed unit-capacity maximum flow). All other edges in the graph were assigned a capacity of 1.

Once all capacities were assigned appropriately and all edges had been added to and from t^* and s^*, an instance of the Maximum Flow Problem was created and an initial unit-capacity flow was calculated using an algorithm provided by the LEMON library. LEMON provides several built-in maximum flow algorithms, ranging from the basic path-augmenting approach of the Edmonds-Karp algorithm [18] to more advanced algorithms, such as Goldberg's Push-Relabel algorithm [17]. For the initial unit-capacity maximum flow computation, the Push-Relabel algorithm was used, as it is often the fastest algorithm offered by the LEMON library in practice.

Once an initial unit-capacity flow $\overline{x}$ had been generated, a value of Δ was calculated, and the starting flow $\overline{f}$ was updated accordingly based on the elementary k-flow resulting from the calculation of $\overline{x}$, as was done in the naive decomposition algorithm. It should be noted that such an elementary k-flow was never explicitly generated in the implementation of the optimized decomposition algorithm; rather, the elementary flow was implicitly referenced using the already-generated unit-capacity flow $\overline{x}$, by ignoring the added edges adjacent to s^* and t^* and assuming all tight edges carried a flow of 1 unit (note that implicitly "creating" elementary k-flows in this way mimics exactly the steps taken in Section 2.1.1 to convert a unit-capacity max-flow instance back into a proper elementary k-flow).

After $\overline{x}$ had been initially generated by calling a single unit-capacity max-flow algorithm, subsequent iterations followed the outline of the optimized decomposition algorithm presented in Section 2.1.3. Dead edges (those with a flow of 0 units in $\overline{f}$) and tight edges (those with a flow of v units in $\overline{f}$) were marked and kept track of througout the course of the algorithm; in this way, after $\overline{f}$ was updated at the end of each iteration (along with the values of Δ and v), edges that had become newly tight and newly dead could easily be identified in each subsequent iteration.

Following the outline of Section 2.1.3, once $\overline{f}$, Δ, and v had been updated, edges that had newly become tight were dealt with first. For each newly tight edge $e = (u, v)$, the edges (s^*, v) and (u, t^*) were each added to the graph with capacity 1, and e was effectively removed from the graph by dropping its capacity to 0. Once all such modifications had been completed for each newly tight edge, the resulting flow was augmented once per each newly tight edge, to ensure the resulting flow saturated all edges leaving s^* and all edges entering t^*. Because individual augmentations were required during this stage of the algorithm, once $\overline{x}$ was initially calculated during the first iteration via the Push-Relabel algorithm, an instance of the Edmonds-Karp algorithm was initialized for use during the remainder of the decomposition. With this choice of max-flow algorithm, LEMON allows for the fine-tuned control of the algorithm's execution, including the ability to call for single path augmentations. LEMON's ability to allow for such low-level modifications to existing flows made the minor flow alterations necessary during each iteration of the optimized decomposition algorithm simple to implement.

Once each newly tight edge had been properly accounted for, newly dead edges were updated by once again following the method outlined in Section 2.1.3. Recall that in order to reduce the flow x_e on each newly dead edge e to 0, a breadth-first search must be performed over the edges in $\overline{x}$ to either identify a cycle containing e (in which case the flow on each edge of the cycle can simply be reduced to 0), or to identify an s^*-t^* path p containing e and to reduce the flow on each edge in p before removing e from the graph and re-augmenting to generate a new saturating flow. To accommodate the necessary executions of the breadth-first search algorithm, tools included in the LEMON library were utilized to create a subgraph containing only the edges of $\overline{x}$ carrying non-zero flow. As with other standard graph algorithms mentioned previously, LEMON also includes an efficient implementation of the breadth-first search algorithm, including a simple function used to return the path found by the algorithm between the source node of the search and any other node in the graph. Utilizing this built-in functionality of the library, along with the ability to search only over a specified subgraph of the flow network, made the modifications necessary to update the flow on each newly dead edge relatively simple to implement. As was the case when correcting $\overline{x}$ to accommodate for newly tight edges, newly dead edges were effectively removed from the graph by setting their capacities to 0, and the previously created instance of the Edmonds-Karp algorithm was utilized to re-augment $\overline{x}$ after the flow along any s^*-t^* path was reduced.

4.3 Experimental Results

Using the implementation methods described in Section 4.2 and several randomly generated k-route flow instances created using the techniques covered in Section 4.1, the naive and optimized versions of the k-route flow decomposition algorithm were experimentally compared to gain insight into their relative real-world performance on a consumer laptop. Three separate grid-based flows were randomly generated using the structure of the 1024x1024 GRIDGRAPH instance described in Section 4.1 as a starting point, along with three separate flows generated randomly using a DIMACS road-network instance representing the New York City metropolitan area. While the sizes of the testing instances varied due to the random nature of the flow generation procedure, flows containing relatively high numbers of nodes and edges were selected to properly measure the speed differences between the various decomposition algorithms.

All tests were timed using the high resolution clock included in the chrono header file, part of the C++11 standard library. The chrono high resolution clock offers simple tools used to measure the execution times of programs, and provides precision to the nanosecond or microsecond, depending on the specific system on which the tests are being performed. All raw time measurements were recorded in nanoseconds; measurements did not include the time required to import the initial k-route flow prior to the start of the decomposition algorithm's execution, and no print statements or extraneous code was included in the implementations used for testing.

Table 4.1 shows the final results of the testing carried out; the reported running times for each test represent the average running time over 5 individual timed runs. Interestingly, the results indicate that the optimized version of the decomposition algorithm performed significantly worse in practice than the naive algorithm, both on starting flows generated from grid-based graph instances as well as flows generated using real-world road networks. While the theoretical running time improvement of the optimized decomposition algorithm seems to intuitively back up the assumption that practical experiments would reflect a similar speed improvement (especially given the fact that only a single maximum flow algorithm must be run in full, as opposed to repeated computations during each iteration as in the naive algorithm), the results seem to indicate that the overhead required to update and maintain

Flow Instance	**Nodes**	**Edges**	**Decomposition Time (s)**	
			Naive	**Optimized**
Grid 1	139078	141096	70.59	618.13
Grid 2	149081	151260	92.66	561.14
Grid 3	134891	137077	88.44	544.56
Road 1	22673	60473	65.32	1174.37
Road 2	29310	74022	69.95	1788.38
Road 3	18131	48984	49.60	781.94

Table 4.1: Results of testing comparing the implementations of the naive and optimized k-route flow decomposition algorithms described in Section 4.2.

the single unit-capacity maximum flow instance throughout the lifetime of the optimized algorithm outweight any such potential running time improvements.

Although more exploration is necessary to determine the exact cause of the stark differences in the running times of the naive and optimized implementations of the decomposition algorithm, preliminary testing indicates that the repeated executions of the breadth-first search algorithm required to correct for newly dead edges has a significant impact on the running time of the optimized version of the algorithm, potentially negating the time saved in not recomputing a full maximum flow from scratch at the start of each iteration. Recall that each newly tight edge requires only a single flow augmentation via the Edmonds-Karp algorithm, which results in a single call to breadth-first search. In constrast, correcting a single newly dead edge can require multiple calls to breadth-first search: one to detect if the edge is part of a cycle in the previously-computed unit-capacity flow, two to find an s^*-t^* path containing the edge (in the event that it is not contained in such a cycle), and one additional call to breadth-first search to re-augment the resulting flow to ensure it remains saturating.

Another intriguing testing result is not included in Table 4.1, to avoid repetition: with both the naive and optimized implementations of the decomposition algorithm, on all flow instances included in the timed experiments, exactly 100 iterations were required to completely decompose the initial k-

route flow (recall that a value of $v = 100$ was used in all tests). While this doesn't necessary reflect on the functioning of the decomposition algorithms themselves, it indicates that the flow generation techniques used to create k-route flows (as outlined in Section 4.1) created starting flows with a wide range of flow values across edges (note that for a decomposition to require a full v iterations to terminate, the value of Δ must be equal to 1 over all iterations).

Also of interest is the contrast in the running times required for decomposing those flows built from road networks compared to those flows built from grid-based graph structures. While in both cases the naive decomposition algorithm significantly outperformed the optimized algorithm, it appears that the degree of slowdown between the naive and the optimized algorithm was considerably greater when testing with the road-based flow instances, as opposed to tests completed using the grid-based instances. Due to the increased time required by the optimized algorithm in particular when decomposing these specific flow instances, road-based flows containing somewhat fewer nodes and edges were used for testing purposes compared to the grid-based flows that were tested. While it is unclear what might have accounted for this discrepancy in running times between flows generated using different graph structures, one hypothesis is that the relatively random nature of road networks (as opposed to the extremely well-structured layout of grid-based graphs) contributed to the increased time required for decomposition.

CHAPTER 5

CONCLUSIONS

Throughout the course of this thesis, the study of k-route flows, first introduced by Kishimoto et al. in [3], was expanded through the introduction of both exact and approximate algorithms to efficiently decompose such flows into constituent elementary k-flows. The decomposition techniques introduced in Chapter 2 were then shown to lead to novel techniques for solving the Minimum Congestion Routing Problem in Chapter 3, providing simple approximation algorithms for the problem following the basic randomized rounding approach of [8] while attaining matching approximation guarantees, without requiring the use of specialized dependent rounding techniques (as were used in [9]). In Chapter 4, both the naive and optimized versions of the exact k-route flow decomposition algorithm introduced in Chapter 2 were efficiently implemented using the LEMON C++ optimization library, and were experimentally compared using k-route flow instances generated from various graph structures. Interestingly, the naive implementation significantly outperformed the theoretically optimal version of the algorithm, perhaps implying the negative real-world impacts of the extra graph modifications required by the optimized algorithm.

There are several interesting possibilities for future work on the topics discussed throughout this thesis. First, although several effective optimization techniques were applied to the k-route flow decomposition algorithms introduced in Chapter 2, future research could focus on taking new approaches to achieve faster theoretical running times for the algorithms discussed there. Many potential avenues for future work focus on the implementations of the decomposition algorithms discussed in Chapter 4, especially considering the unexpected experimental results contrasting the naive and optimized versions of the exact k-route flow decomposition algorithm. In particular, future work could aim to improve the practical running times of both algorithms by utilizing more formal benchmarking and software development techniques, taking into account the implications of using specific algorithms and data structures (specific to both the LEMON library and to the C++ language itself) within each implementation. In addition, the approximate decomposition algorithm of Section 2.2 could be implemented, to give insights into its performance as compared to that of the exact algorithms discussed in Chapter 4, as well as to examine the real-world quality of the decompositions it generates. Another important topic to be explored is the relative performance of the newly introduced algorithms for the Minimum Congestion Routing Problem of Chapter 3, relying on the k-route flow decomposition algorithms introduced in Chapter 2, as compared to the previously published algorithm of Srinivasan [9] for the multiple-path version of the problem. While such experiments were beyond the scope of this thesis, comparative testing of these algorithms in a real-world setting would provide invaluable insights into the practical usefulness of k-route flow decomposition algorithms as a tool in developing efficient approximation algorithms for optimization problems, and may lead to further interest in utilizing the properties of k-route flows in other domains.

REFERENCES

[1] M. T. L. Ahuja, Ravindra K. and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[2] W. Kishimoto, M. Takeuchi, and G. Kishi, "Two-route flows in an undirected flow network," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 76, no. 6, pp. 38–59, 1993. [Online]. Available: http://dx.doi.org/10.1002/ecjc.4430760604

[3] W. Kishimoto and M. Takeuchi, "m-route flows in a network," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 77, no. 5, pp. 1–18, 1994. [Online]. Available: http://dx.doi.org/10.1002/ecjc.4430770501

[4] C. C. Aggarwal and J. B. Orlin, "On multiroute maximum flows in networks," *Networks*, vol. 39, no. 1, pp. 43–52, 2002. [Online]. Available: http://dx.doi.org/10.1002/net.10008

[5] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *The Annals of Mathematical Statistics*, pp. 493–507, 1952.

[6] A. Madry, "Navigating central path with electrical flows: From flows to matchings, and back," in *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, ser. FOCS '13. Washington, DC, USA: IEEE Computer Society, 2013. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2013.35 pp. 253–262.

[7] J. Kleinberg and E. Tardos, *Algorithm Design.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[8] P. Raghavan and C. D. Tompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, Dec. 1987. [Online]. Available: http://dx.doi.org/10.1007/BF02579324

[9] A. Srinivasan, "Distributions on level-sets with applications to approximation algorithms," in *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, ser. FOCS '01. Washington, DC, USA: IEEE Computer Society, 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=874063.875563 pp. 588–.

[10] B. Doerr, M. Künnemann, and M. Wahlström, "Randomized rounding for routing and covering problems: Experiments and improvements," in *Proceedings of the 9th International Conference on Experimental Algorithms*, ser. SEA'10. Berlin, Heidelberg: Springer-Verlag, 2010. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13193-61_7 pp. 190–201.

[11] B. Dezs, A. Jüttner, and P. Kovács, "Lemon - an open source c++ graph template library," *Electronic Notes in Theoretical Computer Science*, vol. 264, no. 5, pp. 23–45, July 2011. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2011.06.003

[12] P. Kovács and A. Juttner, "Lemon random graph generators," 2009. [Online]. Available: http://lime.cs.elte.hu/~kpeter/hg/hgwebdir.cgi/lemon-generators/

[13] B. Bollobás and W. Fernandez de la Vega, "The diameter of random regular graphs," *Combinatorica*, vol. 2, no. 2, pp. 125–134, Jun 1982. [Online]. Available: http://dx.doi.org/10.1007/BF02579310"

[14] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*. American Mathematical Soc., 2009, vol. 74.

[15] P. Kovács, "Minimum-cost flow algorithms: An experimental evaluation," *Optimization Methods Software*, vol. 30, no. 1, pp. 94–127, Jan. 2015. [Online]. Available: http://dx.doi.org/10.1080/10556788.2014.895828

[16] M. Resende and G. Veiga, "An efficient implementation of a network interior point method," in *Network Flows and Matching: First DIMACS Implementation Challenge*, ser. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, D. Johnson and C. McGeoch, Eds. American Mathematical Society, 1993, vol. 12, pp. 299–348.

[17] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, no. 4, pp. 921–940, Oct. 1988. [Online]. Available: http://doi.acm.org/10.1145/48014.61051

[18] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, pp. 248–264, Apr. 1972. [Online]. Available: http://doi.acm.org/10.1145/321694.321699