Improved Bi-criteria Approximation for the All-or-Nothing Multicommodity Flow Problem in Arbitrary Networks

by

Anya Chaturvedi

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved May 2020 by the
Graduate Supervisory Committee:

Andréa Richa, Chair
Arunabha Sen
Stefan Schmid

ARIZONA STATE UNIVERSITY

August 2020

ABSTRACT

This thesis addresses the following fundamental maximum throughput routing problem: Given an arbitrary edge-capacitated n-node directed network and a set of k commodities, with source-destination pairs (s_i, t_i) and demands $d_i > 0$, admit and route the largest possible number of commodities – i.e., the maximum *throughput* – to satisfy their demands. The main contributions of this thesis are three-fold: First, a bi-criteria approximation algorithm is presented for this all-or-nothing multicommodity flow (ANF) problem. This algorithm is the first to achieve a *constant approximation of the maximum throughput* with an *edge capacity violation ratio that is at most logarithmic in* n, with high probability. The approach used is based on a version of randomized rounding that keeps splittable flows, rather than approximating those via a non-splittable path for each commodity: This allows it to work for *arbitrary directed edge-capacitated graphs*, unlike most of the prior work on the ANF problem. The algorithm also works if a weighted throughput is considered, where the benefit gained by fully satisfying the demand for commodity i is determined by a given weight $w_i > 0$. Second, a derandomization of the algorithm is presented that maintains the same approximation bounds, using novel pessimistic estimators for Bernstein's inequality. In addition, it is shown how the framework can be adapted to achieve a polylogarithmic fraction of the maximum throughput while maintaining a constant edge capacity violation, if the network capacity is large enough. Lastly, one important aspect of the randomized and derandomized algorithms is their *simplicity*, which lends to efficient implementations in practice. The implementations of both randomized rounding and derandomized algorithms for the ANF problem are presented and show their efficiency in practice.

ACKNOWLEDGMENTS

This thesis would not have been possible without the support and encouragement of many people.

First of all, I would like to thank my graduate supervisory committee who took out the time for supervising my work. I would specially like to thank my supervisor, Andréa Richa, for her guidance and support through each stage of the process. Also, I would like to thank Stefan Schmid and Matthias Rost for their contribution to this work.

I am grateful for the constant support provided by all my lab mates especially Jamison Weber and Joshua Daymude who were always there to help me. I would also like to thank all my friends who supported me while I worked on this.

I also want to thank my family especially my parents and grandfather for always believing in me.

TABLE OF CONTENTS

APPENDIX Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

A multi-commodity flow problem is a problem where one has to route multiple commodities or flow demands over a network to and from each commodity's respective source and destination. Such problems lie at the heart of many network optimization models. With respect to this thesis, we are interested in the practically relevant problem variant in which edges have capacities like is the case with any real world network. This further enforces that not all requests between pairs of nodes may be satisfied concurrently. Hence, in addition to assigning requests to paths, an admission control mechanism is required which prevents violating the edge capacities and thus, overloading the network.

The objective is to maximize the throughput, i.e., to transmit and satisfy the requests of as many commodities as possible while keeping a check on the edge capacities. We allow a request to be either transmitted as a whole or not at all. Also, in our problem we are allowed to split flows on to multiple edges from a node; combining everything this is known as the *All-or-Nothing (Splittable) Multicommodity Flow (ANF)* problem.

More formally, the problem can be modelled as a flow network where a capacitated directed graph $G(V, E)$ has V as the set of nodes and E as the set of edges, and each edge e has a capacity $c_e > 0$ associated with it. Let $n = |V|$ and $m = |E|$. We are given a set of source-destination pairs (s_i, t_i), where $s_i, t_i \in V$, $i \in [k]$[1], each with given (non-uniform) demand $d_i > 0$ and weight $w_i > 0$. The edge capacities c_e, the demands d_i and weights w_i can be arbitrary positive functions on n and k, for any $e \in E$.

A valid set of flows for commodities $1, \ldots, k$ in G (i.e., a valid *multicommodity flow*), must satisfy standard flow conservation constraints for each commodity i, which imply that the amount of flow for commodity i entering a node v has to be equal to the flow for commodity i leaving v, if $v \neq s_i, t_i$. The *load* of an edge e, given by the sum of the flows for all commodities on e, must not exceed the edge's capacity c_e. The flow can be *split* along many branching routes, provided that flow conservation and edge capacity constraints are satisfied. Commodity i is *satisfied* if d_i units of flow of this commodity can be successfully routed in the network and adds w_i amount to the overall throughput. Later, we will discuss a mixed integer programming formulation (MIP Formulation 1) for the ANF problem, which captures all of these constraints.

We aim to maximize the total number of commodities that are concurrently satisfied in a valid multicommodity flow. Specifically, we consider a weighted generalization of this problem, where, in addition to the demands d_i, each commodity is given a weight w_i and the goal is to find a subset $K' \subseteq [k]$ of commodities to be concurrently satisfied such that the (weighted) *throughput*, given by $\sum_{i \in K'} w_i$, is maximized over all possible K'. The flow can be *split* arbitrarily along many branching routes (subject to flow conservation and edge capacity constraints) and does not have to be integral.

[1]Let $[x]$ denotes the set $\{1, \ldots, x\}$, for any positive integer x.

Explaining this through an example let us assume our network to be the internet, our commodities as videos and the destination as the user who wants to view a particular video. I have displayed the same through Figure 1.1 where the color of the video matches the color of the figure I have depicted as the respective user. We obviously want the internet to support the maximum number of users it can at the same time but as we know in the real worlds there might be certain limitations like bandwidth here. Also, this is a situation where the "All-or-Nothing" aspect of the problem clearly shows – when a video or file goes over the network it gets divided into packets, for a file to be usable by the user it needs to have all its packets or an incomplete file would be as bad as receiving nothing. Thus, we want to focus on such problems where we want to transmit either the whole commodity or nothing at all while aiming to maximize the throughput at the same time. The demands of the commodities can be taken as the video or file sizes in this context and the weights can be the priority the customer has for the streaming application.

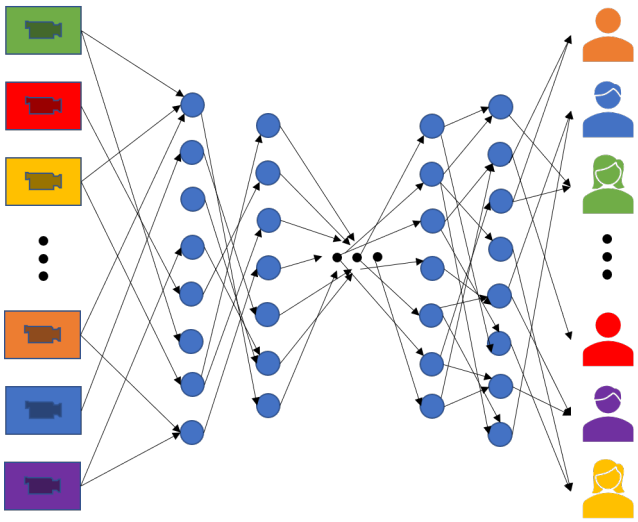

Figure 1.1: Example

The ANF problem has been shown to be APX-hard Chekuri *et al.* (2004); Garg *et al.* (1997) (even if the underlying graph is an tree) and hard to approximate within $n^{\Omega(1/c)}$ in directed graphs even with unit demands and with congestion c as show by Chuzhoy *et al.* (2007). Hence, the literature has followed a bi-criteria optimization approach where one is allowed to violate the edge capacities slightly as one seeks a good approximation on the throughput. For this thesis, we focused on an (α, β)-*approximation algorithm*: For parameters $\alpha \in (0, 1]$ and $\beta \geq 1$, we seek a polynomial time algorithm that outputs a multicommodity flow solution satisfying flow conservation constraints for each commodity i, whose *throughput is at least an α fraction of the maximum throughput* and whose *load on any edge e is at most β times the edge capacity c_e*, with high probability[2]. The parameter β hence provides an upper bound on the *edge capacity violation ratio* incurred by the algorithm. Following from Liu *et al.* (2019) we formed a more general algorithm involving both varying demands and weights. Further, we came up with a deterministic algorithm with the same (α, β)-approximation by derandomizing the randomized algorithm from Liu *et al.* (2019) using the method of pessimistic estimators. The implementations of both the randomized and deterministic algorithms can be found in Appendix A.

[2]With probability at least $1 - 1/n^c$, where $c > 0$ is a constant.

Chapter 2

RELATED WORK

The study of routing and multicommodity flow problems is motivated by many real-world applications — related to traffic networks, production lines, designing route networks for container ships, etc. — as well as by the important role that flows and cuts play in combinatorial optimization (e.g., see Chekuri and Ene (2015) and references within).

In Liu *et al.* (2019), they present a $(1/3, O(\sqrt{k \log n}))$-approximation algorithm for the ANF problem for the case of *uniform* demands in directed graphs, where k is the number of commodities. Our current work improves and generalizes the randomized rounding framework outlined in Liu *et al.* (2019), in many ways: Our bound on the edge capacity violation does not depend on the number of commodities k and is better than that of Liu *et al.* (2019) when $k = \Omega(\log n)$; we were also able to accommodate arbitrary non-uniform demands and commodity weights, while the results in Liu *et al.* (2019) hold for uniform demands and weights. In fact, one could combine the algorithm in this paper with the one in Liu *et al.* (2019) to obtain a $(1/3, \min\{(3 \log n + 1), O(\sqrt{k \log n})\})$-*approximation algorithm*, which is the best-known result for the ANF problem on arbitrary directed networks G with uniform demands and arbitrary number of commodities k.

Other work on bi-criteria (α, β)-approximation schemes for the ANF problem that are closely related to ours aims at keeping β constant, while letting α be a function of n. In this line of work, the paper by Chekuri *et al.* (2004) is the most relevant and was the first to formalize the ANF problem. The authors present an approximation algorithm for the general (weighted, non-uniform demands) ANF problem

on *undirected graphs* with constant edge capacity violation ratio and a throughput approximation ratio of $\Omega(1/(\log^3 n \log\log n))$ based on Räcke's hierarchical tree decomposition as shown in Räcke (2008). A requirement of their algorithm is that $\max d_i \leq \min c_e$, which implies that any commodity i can always be routed at demand d_i through *any single path* in G. This is a strong assumption, since it eliminates all (undirected) networks G where the above assumption fails but where the capacity of a minimum (s_i, t_i)-cut is at least d_i (i.e., where G could still sustain a d_i-flow for commodity i), such as for example complete graphs with unit edge capacities and demands $2 \leq d_i \leq n-1$, for all i. Hence, besides the fact that our approximation guarantees differ from those of Chekuri *et al.* (2004) (we have constant α and logarithmic β, while they achieve constant β at the expense of a polylogarithmic $1/\alpha$), our results also apply to *any directed graph* G, without any assumptions on how d_i compares to individual edge capacities or to the capacity of a minimum (s_i, t_i)-cut. In Section 6, we show how we can adapt our approach to yield polylogarithmic $1/\alpha$ and constant β approximation bounds for arbitrary directed networks provided that the capacity of a minimum (s_i, t_i)-cut is $\Omega(d_i \log n)$, for all satisfiable flows i. More specifically, we present a $(1/\Theta(\log n), \Theta(1))$-approximation algorithm for the general ANF problem on directed networks that have (s_i, t_i)-cuts of capacity $\Omega(d_i \log n)$, improving the bounds by Chekuri *et al.* (2004) under these conditions.

The ANF problem gets considerably more challenging in directed graphs. In Chekuri and Ene (2015), they consider a variation of the ANF problem — the *Symmetric All or Nothing Flow (SymANF)* problem — in *directed* graphs with *symmetric demand pairs*, also aiming at constant β and polylogarithmic $1/\alpha$. In SymANF, the input pairs are unordered and a pair (s_i, t_i) is routed if only if the ordered pairs (t_i, s_i) ais also routed; the goal is to find a maximum subset of the given demand pairs that can be routed. The authors provide a poly-logarithmic approximation with constant congestion for SymANF, by extending the well-linked decomposition framework of Chekuri *et al.* (2005) to the directed graph setting with symmetric demand pairs. Our work differs from Chekuri and Ene (2015) in that their results depend on the more restricted assumptions of unit edge capacity and symmetric unit demand. Our work considers a more general network setting and our original aim was to obtain a constant approximation of the throughput with polylogarithmic edge capacity violations (our constant edge capacity violation results in Section 6 would also apply here if the capacity of the network is large enough). One can show that the throughput for the SymANF with constant edge capacity violation c is hard to approximate within a factor of $(\log |V|)^{\Omega(1/c)}$; Chekuri and Ene (2015); Arora *et al.* (1998).

The *Maximum Edge-Disjoint Paths (MEDP)* problem as presented by Erlebach and Jansen (2001) considers a set of pairs of nodes to be routable if they can be connected using edge-disjoint paths and aims at finding the largest number of routable pairs. One can obtain the MEDP from ANF by requiring all flow to go on a single path. That is, MEDP can be viewed as a restricted version of the ANF problem when all d_i's, w_i's and c_e's are equal to 1, and the flow on each edge is required to be integral. For arbitrary directed graphs, the MEDP problem was shown to be NP-hard to approximate within $m^{1/2-\epsilon}$, for any $\epsilon > 0$ as shown in Guruswami *et al.* (2003). Approximation algorithms with approximation ratio $O(m^{1/2})$ are known

through Kleinberg and Goemans (1996); see also Kolliopoulos and Stein (1998); Srinivasan (1997). As for the ANF problem in general, better approximation ratios can be achieved for restricted classes of graphs. For example, the MEDP problem can be solved optimally in polynomial time for bidirected trees of constant degree; however it is APX-hard for bidirected trees of arbitrary degree as shown in Erlebach and Jansen (2001).

Finally, our work leverages randomized rounding techniques presented in work by Rost and Schmid (2018) and Rost *et al.* (2019) through the different context of virtual network embedding problems (i.e., in their context, flow endpoints are subject to optimization).

Chapter 3

PRELIMINARIES

In this chapter, we go over the preliminary theorems and facts which will be used in the further chapters.

3.1 Some Convexity Arguments

Lemma 1. *The following holds for any $\theta \in \mathbb{R}$ and $x \in [0,1]$:*

$$\exp(\theta \cdot x) \leq 1 + (\exp(\theta) - 1) \cdot x$$

Proof. The function $f(x) = c^x$ is convex as $f''(x) = \ln^2(c) \cdot c^x > 0$ holds for any $c > 0$ and any $x \in \mathbb{R}$. As f is convex, the following holds for any $z \in [x_1, x_2]$

$$f(z) \leq \frac{f(x_2) - f(x_1)}{x_2 - x_1} \cdot (z - x_1) + f(x_1)$$

Setting $c = \exp(\theta)$, $x_1 = 0$, and $x_2 = 1$, the result is obtained for all $z \in [0,1]$:

$$\exp(\theta \cdot z) \leq (\exp(\theta) - 1) \cdot z + 1\,. \qquad \square$$

Lemma 2. *The following holds for any $\theta \in \mathbb{R}$ and $x \in [0,1]$:*

$$1 + (\exp(\theta) - 1) \cdot x \leq \exp((\exp(\theta) - 1) \cdot x)$$

Proof. The inequality follows from the observation that $1 + z \leq \exp(z)$ holds for $z \in \mathbb{R}$, which follows from the convexity of $f(z) = \exp(z)$. By substituting z with $(\exp(\theta) - 1) \cdot x$ the result is obtained $\square$

Lemma 3. *Let $X_l \in [0,1]$ denote a single random variable of expectation $\mu_l = \mathsf{Ex}(X_l)$. For any $\theta \in \mathbb{R}$ the following holds for the random variable $Y_l = \exp(\theta \cdot X_l)$:*

$$\mathsf{Ex}(Y_l) \leq \exp((\exp(\theta) - 1) \cdot \mu_l) \tag{3.1}$$

Proof. As the function $f(x) = \exp(\theta \cdot x)$ is convex for any $t \in \mathbb{R}$, the following holds:

$$\begin{aligned}
\mathsf{Ex}(Y_i) =& \mathsf{Ex}(\exp(\theta \cdot X_l)) \\
\leq& \mathsf{Ex}(1 + (\exp(\theta) - 1) \cdot X_l) && \text{[by Lemma 1]} \\
=& 1 + (\exp(\theta) - 1) \cdot \mathsf{Ex}(X_l) && \text{[by linearity of expectation]} \\
\leq& \exp((\exp(\theta) - 1) \cdot \mu_l) && \text{[by Lemma 2]}
\end{aligned}$$

□

Lemma 4. *The following inequality holds for any $x \in [0,1)$:*

$$-x - (1-x)\ln(1-x) \leq -x^2/2\,. \tag{3.2}$$

Proof. To start off, we note that the above inequality is satisfied for $x = 0$ and that the inequality is preserved as long as the derivative of the left-hand side is less than the derivative of the right-hand side. Considering the derivatives $\frac{d}{dx}(-x^2/2) = -x$ and $\frac{d}{dx}(-x - (1-x)\ln(1-x)) = \ln(1-x)$, we note again that for $x = 0$ the respective derivatives have the same value. Repeating the argument, the derivative of the left-hand side is smaller than the derivative of the right-hand side if the second derivative of the left-hand side is always smaller than the second derivative of the right-hand side. Hence, as $\frac{d^2}{dx}(-x - (1-x)\ln(1-x)) = -1/(1-x)$, $\frac{d^2}{dx}(-x^2/2) = -1$, and $-1/(1-x) \leq -1$ holds for $x \in (0,1)$, the second and first derivatives of the left-hand side are smaller than the respective derivatives of the right-hand side. Hence, Inequality 3.2 holds for any $x \in [0,1)$. □

3.2 Chernoff Bound

Fact 1. *Let X be the sum of k independent random variables $X_1, \ldots, X_k$ with $X_l \in [0,1]$ for $l \in [k]$. Denoting by $\tilde{\mu}_l \leq \mu_l = \mathsf{Ex}(X_\ell)$ lower bounds on the expected value of random variable X_l, $l \in [k]$, the following holds for any $\delta \in (0,1)$ with $\tilde{\mu} = \sum_{l\in[k]} \tilde{\mu}_l$ and $\theta = \ln(1-\delta)$;*

$$\mathsf{Pr}\left(X \leq (1-\delta)\cdot\tilde{\mu}\right) \overset{(a)}{\leq} e^{-\theta\cdot(1-\delta)\cdot\tilde{\mu}} \cdot \prod_{l\in[k]} \mathsf{Ex}\left(e^{\theta\cdot X_l}\right) \overset{(b)}{\leq} e^{-\delta^2\cdot\tilde{\mu}/2} \tag{3.3}$$

Proof. We first prove the inequality $\mathsf{Pr}\left(X \leq (1-\delta)\cdot\tilde{\mu}\right) \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\tilde{\mu}}$ for $\delta \in (0,1)$. Let $Y_l = \exp(\theta \cdot X_l)$, $l \in [k]$, for $\theta = \ln(1-\delta)$. Note that $\theta < 0$ holds.

By Lemma 3 Equality 3.1 holds. As $\exp(\theta) - 1 = -\delta < 0$ holds for $\theta < 0$, the exponential function $f(z) = \exp((\exp(\theta)-1)\cdot z)$ is monotonically decreasing. Using the lower bound $\tilde{\mu}_l \leq \mu_l$ and $\tilde{\mu} = \sum_{l\in[k]} \tilde{\mu}_l$ the following is obtained:

$$\mathsf{Ex}(Y_l) \leq \exp((\exp(\theta)-1)\cdot\tilde{\mu}_l) \tag{3.4}$$

As the variables $X_1, \ldots, X_k$ are pairwise independent, the variables $Y_1, \ldots, Y_k$ are also pairwise independent. Accordingly, the following holds for $Y = e^{\theta\cdot X}$:

$$\mathsf{Ex}(Y) = \mathsf{Ex}\left(e^{\theta\cdot\sum_{l\in[k]} X_l}\right) = \mathsf{Ex}\left(\prod_{l\in[k]} e^{\theta\cdot X_l}\right) = \prod_{l\in[k]} \mathsf{Ex}(Y_l)\ . \tag{3.5}$$

Accordingly, the following is obtained:

$$\begin{aligned}
&\Pr[X \leq (1-\delta)\cdot\tilde{\mu}] && && (3.6)\\
&= \Pr[e^{\theta\cdot X} \geq e^{\theta\cdot(1-\delta)\cdot\tilde{\mu}}] && [\text{as } \theta < 0] && (3.7)\\
&\leq \frac{\mathsf{Ex}(\exp[\theta\cdot X])}{e^{\theta\cdot(1-\delta)\cdot\tilde{\mu}}} && [\text{by Markov's inequality}] && (3.8)\\
&= \frac{\prod_{l\in[k]} \mathsf{Ex}(Y_l)}{e^{\theta\cdot(1-\delta)\cdot\tilde{\mu}}} && [\text{by Equation 3.5}] && (3.9)\\
&\leq \frac{\prod_{l\in[k]} e^{(\exp(\theta)-1)\cdot\tilde{\mu}_l}}{e^{\theta\cdot(1-\delta)\cdot\tilde{\mu}}} && [\text{by Equation 3.4}] && (3.10)
\end{aligned}$$

$$= e^{\left(\sum_{l\in[k]} e^{\theta}-1\cdot\tilde{\mu}_l\right)-\left(\theta\cdot(1-\delta)\cdot\tilde{\mu}\right)} \qquad \text{[one exponent]} \tag{3.11}$$

$$= e^{\left(\sum_{i\in[N]}((1-\delta)-1)\cdot\tilde{\mu}_l\right)-\left(\ln(1-\delta)\cdot(1-\delta)\cdot\tilde{\mu}\right)} \qquad \text{[using } \theta = \ln(1-\delta)\text{]} \tag{3.12}$$

$$= e^{\left(-\delta\cdot\tilde{\mu}\right)-\left(\ln(1-\delta)\cdot(1-\delta)\cdot\tilde{\mu}\right)} \qquad \text{[definition of } \tilde{\mu} = \sum_{l\in[k]}\tilde{\mu}_l\text{]} \tag{3.13}$$

$$= \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\tilde{\mu}} \tag{3.14}$$

Given Equation 3.14, Inequality (b) is a corollary of Lemma 4, which showed the following for $\delta \in (0,1)$

$$-\delta - (1-\delta)\ln(1-\delta) \leq -\delta^2/2\,. \tag{3.15}$$

Multiplying both sides with $\tilde{\mu}$ and exponentiating both sides yields the desired result.

Regarding the Inequality (a), we note that this follows by the above proof from Equation 3.9. □

3.3 Bernstein Bounds

To prove Theorem 4.3 of Page 21, we first prove the following general Bernstein bound.

Theorem 1 (General Bernstein Concentration Bound Boucheron *et al.* (2013a))**.** *Given is a collection* $\{X_l\}_{l\in[k]}$ *of* $k \in \mathbb{N}$ *independent (positive) random variables and a constant* $R > 0$*, such that the following holds for all* $l \in [k]$ *and all* $n \in \mathbb{N}$*,* $n \geq 2$*:*

$$\mathsf{Ex}\left(|X_l - \mathsf{Ex}\left(X_l\right)|^n\right) \leq \frac{n!}{2}\cdot R^{n-2}\cdot\sigma_l^2 \tag{3.16}$$

The following holds for all $t > 0$*:*

$$\mathsf{Pr}\left(\sum_{l\in[k]}\left(X_l - \mathsf{Ex}\left(X_l\right)\right) \geq t\right) \leq \exp\left(-\frac{t^2/2}{\sigma^2 + R\cdot t}\right) \tag{3.17}$$

Proof.

$$\mathsf{Pr}\left(\sum_{l\in[k]}\left(X_l - \mathsf{Ex}\left(X_l\right)\right) \geq t\right) \tag{3.18}$$

$$= \mathsf{Pr}\left(e^{\theta \cdot \sum_{l\in[k]} X_l - \mathsf{Ex}(X_l)} \geq e^{\theta \cdot t}\right) \qquad \text{[for an arbitrary } \theta > 0\text{]} \tag{3.19}$$

$$\leq e^{-\theta \cdot t} \cdot \mathsf{Ex}\left(e^{\theta \cdot \sum_{l\in[k]} X_l - \mathsf{Ex}(X_l)}\right) \qquad \text{[Markov's inequality]} \tag{3.20}$$

$$\leq e^{-\theta \cdot t} \cdot \prod_{l\in[k]} \mathsf{Ex}\left(e^{\theta \cdot (X_l - \mathsf{Ex}(X_l))}\right) \qquad \text{[independence]} \tag{3.21}$$

Using the assumption of Equation 3.16, we now derive a bound for $\mathsf{Ex}\left(e^{\theta \cdot X_l}\right)$.

$$\mathsf{Ex}\left(e^{\theta \cdot (X_l - \mathsf{Ex}(X_l))}\right) \tag{3.22}$$

$$= \sum_{n=0}^{\infty} \frac{\mathsf{Ex}\left((\theta \cdot (X_l - \mathsf{Ex}\left(X_l\right)))^n\right)}{n!} \qquad \text{[definition of exp. function]} \tag{3.23}$$

$$= 1 + \sum_{n=2}^{\infty} \frac{\theta^n}{n!} \cdot \mathsf{Ex}\left((X_l - \mathsf{Ex}\left(X_l\right))^n\right) \qquad \text{[splitting sum]} \tag{3.24}$$

$$\leq 1 + \sum_{n=2}^{\infty} \frac{\theta^n}{n!} \cdot \frac{n!}{2} \cdot R^{n-2} \cdot \sigma_l^2 \qquad \text{[by Equation 3.16]} \tag{3.25}$$

$$= 1 + \frac{\theta^2 \cdot \sigma_l^2}{2} \cdot \sum_{n=2}^{\infty} (\theta \cdot R)^{n-2} \tag{3.26}$$

$$= 1 + \frac{\theta^2 \cdot \sigma_l^2}{2} \cdot \frac{1}{1 - \theta \cdot R} \qquad \text{[geometric series for } \theta \cdot R < 1\text{]} \tag{3.27}$$

$$\leq e^{\frac{\theta^2 \cdot \sigma_l^2}{2} \cdot \frac{1}{1-\theta \cdot R}} = e^{\theta^2 \cdot \sigma_l^2 / (2 \cdot (1 - \theta \cdot R))} \qquad \text{[as } 1 + x \leq e^x \text{ for } x \in \mathbb{R}\text{]} \tag{3.28}$$

Using the bound of Equation 3.28 within Equation 3.21, we obtain:

$$\mathsf{Pr}\left(\sum_{l\in[k]} \left(X_l - \mathsf{Ex}\left(X_l\right)\right) \geq t\right) \tag{3.29}$$

$$\leq e^{-\theta \cdot t} \cdot \prod_{l\in[k]} e^{\theta^2 \cdot \sigma_l^2 / (2 \cdot (1 - \theta \cdot R))} \qquad \text{[by Equation 3.28]} \tag{3.30}$$

$$= e^{-\theta \cdot t} \cdot e^{\theta^2 / (2 \cdot (1 - \theta \cdot R))} \cdot e^{\sum_{l\in[k]} \sigma_l^2} \tag{3.31}$$

$$= \exp\left(-\theta \cdot t + \frac{\theta^2 \cdot \sigma^2}{2 \cdot (1 - \theta \cdot R)}\right) \tag{3.32}$$

We now set $\theta = t/(\sigma^2 + R \cdot t)$. Notably, this choice satisfies $\theta \cdot R < 1$, which is required in Equation 3.27. Accordingly, we obtain

$$\mathsf{Pr}\left(\sum_{l\in[k]} (X_l - \mathsf{Ex}(X_l)) \geq t\right) \tag{3.33}$$

$$= \exp\left(-\frac{t^2}{\sigma^2 + R \cdot t} + 1/2 \cdot \left(\frac{t}{\sigma^2 + R \cdot t}\right)^2 \cdot \frac{\sigma^2}{(1 - \frac{R\cdot t}{\sigma^2 + R\cdot t})}\right) \tag{3.34}$$

$$= \exp\left(-\frac{t^2}{\sigma^2 + R \cdot t} + 1/2 \cdot \left(\frac{t}{\sigma^2 + R \cdot t}\right)^2 \cdot (\sigma^2 + R \cdot t)\right) = \exp\left(\frac{-t^2/2}{\sigma^2 + R \cdot t}\right) \tag{3.35}$$

which completes the proof. □

Fact 2 (Bernstein Concentration Bound)**.** *Given is a collection* $\{Y_l\}_{l\in[k]}$ *of* $k \in \mathbb{N}$ *independent binary random variables, i.e.,* $Y_l \in \{0, 1\}$*, with* $\mu_l = \mathsf{Ex}(Y_l)$ *for* $l \in [k]$*. Let* $X_l = a_l \cdot Y_l$ *for constant* $0 < a_l \leq M$*. The following holds for any* $t > 0$ *with* $\theta = \left(\frac{t}{\sum_{l\in[k]} a_l^2 \cdot (\mu_l - \mu_l^2) + M \cdot \frac{t}{3}}\right)$*:*

$$\mathsf{Pr}\left(\sum_{l\in[k]} (X_l - \mathsf{Ex}(X_l)) \geq t\right) \overset{(a)}{\leq} e^{-\theta\cdot t} \cdot \prod_{l\in[k]} \mathsf{Ex}\left(e^{\theta\cdot(X_l - \mathsf{Ex}(X_l))}\right) \tag{3.36}$$

$$\overset{(b)}{\leq} \exp\left(-\frac{t^2/2}{\sum_{l\in[k]} a_l^2 \cdot (\mu_l - \mu_l^2) + M \cdot t/3}\right) \tag{3.37}$$

Proof. We will apply Theorem 1 to obtain the result and first prove inequality (b). Choosing $R = M/3$, we first prove that the assumption of Equation 3.16 holds. We first note the following for $\mathsf{Ex}(|X_l - \mathsf{Ex}(X_l)|^n)$:

$$\mathsf{Ex}(|X_l - \mathsf{Ex}(X_l)|^n) = \mathsf{Ex}\left((X_l - \mathsf{Ex}(X_l))^2 \cdot |X_l - \mathsf{Ex}(X_l)|^{n-2}\right) \tag{3.38}$$

$$\leq \mathsf{Ex}\left((X_l - \mathsf{Ex}(X_l))^2 \cdot M^{n-2}\right) \tag{3.39}$$

$$= M^{n-2} \cdot \sigma_l^2 \tag{3.40}$$

Using the fact that $n!/2 \cdot (1/3)^{n-2} \geq 1$ holds for all $n \geq 2$, we obtain:

$$\mathsf{Ex}\left(|X_l - \mathsf{Ex}\left(X_l\right)|^n\right) \leq M^{n-2} \cdot \sigma_l^2 \leq \frac{n!}{2} \cdot \underbrace{(M/3)}_{=R}^{n-2} \cdot \sigma_l^2 \tag{3.41}$$

Hence, we have proven that Equation 3.16 holds for each bounded random variable X_l, $l \in [k]$, and each $n \geq 2$ when $R = M/3$.

Preparing to apply Theorem 1, we note that $\sigma_l^2 = \mathsf{Var}\left(X_l\right) = a_l^2 \cdot (\mu_l - \mu_l^2)$ holds for $l \in [k]$ and that $\sigma^2 = \sum_{l \in [k]} a_l^2 \cdot (\mu_l - \mu_l^2)$ holds. Plugging these values into Theorem 1, the inequality (b) is proven, i.e., we obtain:

$$\mathsf{Pr}\left(\sum_{l \in [k]} \left(X_l - \mathsf{Ex}\left(X_l\right)\right) \geq t\right) \leq \exp\left(-\frac{t^2/2}{\sum_{l \in [k]} a_l^2 \cdot (\mu_l - \mu_l^2) + M \cdot t/3}\right) \tag{3.42}$$

Regarding the inequality (a), we note that the right-hand side is a side-product of the proof of Theorem 1. Specifically, the right-hand side of inequality (a) equals the term of Equation 3.21. As Equation 3.21 eventually led to the inequality (b) under the choice of θ as stated in this theorem, the theorem follows. □

Chapter 4

THE RANDOMIZED APPROACH

This chapter focuses on the randomized rounding algorithm by Liu *et al.* (2019) and how we generalized it and strengthened the existing bounds. The generalization is done in order to accommodate arbitrary demands and commodity weights. We then show how we achieved an improved bound for the edge capacity violation ratio β, logarithmic in n, while still keeping the throughput approximation α constant in the generalized setting.

4.1 MIP Formulation

We are given an n-node flow network $G(V, E)$ with edge capacities $c_e > 0$. We have k commodities that may be routed in this network, where each commodity i is represented by (s_i, t_i), where $s_i, t_i \in V$ denote the respective source and destination nodes. Each commodity i also has an associated demand d_i and a weight w_i, both of which are positive constants. We use an indicator variable $f_i \in \{0, 1\}$ to indicate whether a commodity i is successfully routed through G. Next, we denote $f_{i,e} \in [0, 1]$ as the fraction of flow for commodity i allocated to a particular edge $e \in E$. The total flow assigned to a fixed edge e is given by $\sum_i d_i \cdot f_{i,e}$ and the total weighted throughput is given by $\sum_i w_i f_i$. We present this problem as an integer program in MIP Formulation 1. Without loss of generality, we only consider commodities that each can be satisfied if it were to be routed alone in G (any commodity that does not satisfy this requirement will never be part of any feasible solution to the ANF problem, and hence is irrelevant to the problem). This updated set of commodities forms the input of the mixed integer program. Regarding the MIP formulation, Constraints 4.1

define the value of the total flow for each commodity i, Constraints 4.2 enforce flow conservation for each i, and Constraints 4.3 stipulate that no edge capacity is violated by the flow assignments. Constraints 4.4 ensure that for a fixed commodity i, the ratio of flow assigned to an edge e to the total flow of that commodity does not exceed the capacity of e: These constraints are actually redundant for the MIP formulation, but will strengthen the corresponding linear relaxation (LP) of MIP Formulation 1 — in fact these constraints will be crucial in order to prove our approximation bounds, which will be based on the optimal value of the LP-relaxation, as we will see below. Constraints 4.5 are the non-negativity constraint for flow assignments to edges, and Constraints 4.6 enforce an integral solution to the optimization problem.

MIP Formulation 1: Splittable All-or-Nothing Flow

$$\boldsymbol{w_{OPT}} = \text{Maximize } \sum_{i=1}^{k} w_i f_i$$

Subject to

$$\sum_{(s_i,v)\in E} f_{i,(s_i,v)} = f_i \qquad \forall i \in [k] \tag{4.1}$$

$$\sum_{(u,v)\in E} f_{i,(u,v)} = \sum_{(v,u)\in E} f_{i,(v,u)} \qquad \forall i \in [k], \forall v \in V - \{s_i, t_i\} \tag{4.2}$$

$$\sum_{i=1}^{k} f_{i,(u,v)} \cdot d_i \leq c_{(u,v)} \qquad \forall (u,v) \in E \tag{4.3}$$

$$f_{i,(u,v)} \cdot d_i \leq f_i \cdot c_{(u,v)} \qquad \forall i \in [k], \forall (u,v) \in E \tag{4.4}$$

$$f_{i,(u,v)} \geq 0 \qquad \forall i \in [k], \forall (u,v) \in E \tag{4.5}$$

$$f_i \in \{0,1\} \qquad \forall i \in [k] \tag{4.6}$$

Note that it is safe to discard any commodity i such that i cannot be satisfied if routed alone in G (i.e., if the capacity of min (s_i, t_i)-cut is less than d_i), as we will do in Algorithm 2, since such a commodity will never be part of any feasible solution to the MIP.

Algorithm 2: Randomized Rounding Algorithm

Input : Directed Graph $G(V, E)$ with edge capacities $c_e > 0, \forall e \in E$

Set of k distinct commodities $1, \ldots, k$

Source-sink pair (s_i, t_i), demand $d_i \geq 0$ and weight $w_i \geq 0$ $\forall i \in [k]$

Constant $\alpha \in (0, 1]$

Output: The final values of f_i and $f_{i,e}$ and $w_{ALG} = \sum w_i f_i$

1 Discard any commodity i that cannot be satisfied if routed alone in G (i.e., if the capacity of min (s_i, t_i)-cut is less than d_i).

2 Solve the LP to obtain optimal solution $\tilde{f}_i$ and $\tilde{f}_{i,e}$, $\forall i \in [k]$.

3 For each $i \in [k]$, set $f_i = 1$ with probability $\tilde{f}_i$, otherwise set $f_i = 0$.

4 Rescale the fractional flow $\tilde{f}_{i,e}$ from the LP solution on edge e for commodity i by $\frac{1}{\tilde{f}_i}$: i.e., $f_{i,e} = \frac{\tilde{f}_{i,e}}{\tilde{f}_i} \cdot f_i$ and the flow for commodity i on e is given by $f_{i,e} d_i$.

5 If $w_{ALG} = \sum_i w_i f_i \geq \alpha \sum w_i \tilde{f}_i = \alpha \cdot w_{LP}$, return the corresponding flow assignments given by f_i and $f_{i,e}, \forall i \in [k]$ and $e \in E$. Otherwise, repeat steps 3 and 4, at most $\Theta(\log n)$ times.

Let f_i^* and $f_{i,e}^*$, $\forall i \in [k], e \in E$, be the optimal solution for MIP, and let $w_{OPT} = \sum_i w_i f_i^*$ be its optimal value. Let $\tilde{f}_i$ and $\tilde{f}_{i,e}$, $\forall i \in [k], e \in E$, be the optimal solution for the linear relaxation LP of MIP (i.e., when Constraints 4.6 are relaxed to $0 \leq f_i \leq 1$, $\forall i$); let $w_{LP} = \sum_i w_i \tilde{f}_i$ denote the value of the optimal (fractional) solution of LP.

4.2 Algorithm

Algorithm 2 is a generalization of the algorithm in Liu *et al.* (2019) to accommodate non-uniform commodity demands and weights. First, we solve LP to obtain an optimal relaxed solution for the total fractions of the flow for commodity i, $\tilde{f}_i$, being routed, and for the fractions of $\tilde{f}_i$ being routed through edge e, namely, $\tilde{f}_{i,e}$, for all i and e. We then use randomized rounding to round the fraction $\tilde{f}_i$ of d_i to $f_i = 1$, with probability $\tilde{f}_i$, and to 0 otherwise. If we set f_i to 1, then in order to satisfy flow conservation constraints (Constraints 4.2), we need to rescale all the $\tilde{f}_{i,e}$ values by $1/\tilde{f}_i$, obtaining the flows $f_{i,e}$ in accordance with the MIP constraints (if $f_i = 0$ then $f_{i,e} = 0$, for all $e \in E$). We repeat the randomized rounding steps, Steps 3-4, in Algorithm 2 for $\Theta(\log n)$ iterations until we obtain a throughput within the desired α bound, amplifying the probability of getting a desired outcome (we will show later that in each execution of Steps 3-4 of the algorithm, the probability of violating some edge capacity by more than $O(\log n)$ is very small, basically at most $1/n$). Algorithm 2 clearly runs in polynomial time overall.

4.3 Analysis

In this section, we analyze the theoretical bounds for both α and β in our generalized framework. We state below (in Fact 3) the exact formulation of the variation of Chernoff-type bounds that we will use in our analysis, both when proving the approximation bounds on throughput for Algorithm 2 (Theorem 2) and when presenting our derandomization framework in Section 5. The proofs of this variant of Chernof's bound for non-Poisson binary random variables is given in the Chapter 3 but similar bounds can also be found in classic textbooks as, e.g., Mitzenmacher and Upfal (2017).

Fact 3. *Let X be the sum of k independent random variables $X_1, \ldots, X_k$ with $X_l \in [0,1]$ for $l \in [k]$. Denoting by $\tilde{\mu}_l \leq \mu_l = \mathsf{Ex}(X_\ell)$ lower bounds on the expected value of random variable X_l, $l \in [k]$, the following holds for any $\delta \in (0,1)$ with $\tilde{\mu} = \sum_{l\in[k]} \tilde{\mu}_l$ and $\theta = \ln(1-\delta)$;*

$$\mathsf{Pr}\left(X \leq (1-\delta)\cdot\tilde{\mu}\right) \overset{(a)}{\leq} e^{-\theta\cdot(1-\delta)\cdot\tilde{\mu}} \cdot \prod_{l\in[k]} \mathsf{Ex}\left(e^{\theta\cdot X_l}\right) \overset{(b)}{\leq} e^{-\delta^2\cdot\tilde{\mu}/2} \tag{4.7}$$

We are now ready to state our main result bounding the throughput approximation, in the presence of non-uniform commodity demands and weights:

Theorem 2. *The probability of achieving less than $1 - 2/3 \cdot \sqrt{w_{\max}/w_{LP}} \geq 1/3$ of the maximum throughput is at most $e^{-2/9} \leq 65/81$, for $n \geq 9$.*

Proof. We must calculate the difference between the throughput w_{ALG} and the expected throughput $\mathsf{Ex}(w_{ALG}) = w_{LP} = \sum_i w_i \tilde{f}_i$ of Algorithm 2 to observe α and the related probabilities. Let $w_{max} = \max_i w_i$. We can w.l.o.g.[1] assume that each of the k commodities can be satisfied if routed by itself in G, and hence $w_{OPT} \geq w_{\max}$. We define the following (normalized) variables: Let $X_i = w_i/w_{\max}$ if commodity i is routed and $X_i = 0$ otherwise. Then $\mathsf{Pr}(X_i = w_i/w_{\max}) = \tilde{f}_i$ and $\mathsf{Pr}(X_i = 0) = 1 - \tilde{f}_i$. Let $X = \sum_i X_i$. Then we have $\mathsf{Ex}(X) = \sum_i \mathsf{Ex}(X_i) = \sum_i \tilde{f}_i \cdot w_i/w_{\max}$ and that $w_{ALG} = X \cdot w_{\max}$. Since $\mathsf{Ex}(w_{ALG}) = w_{LP}$ is an upper bound on the optimal achievable throughput w_{OPT}, we also have $w_{max} \leq w_{OPT} \leq w_{LP}$.

Noting that the variables X_i conform with the requirements of Fact 3 and setting $\delta = 2/3 \cdot \sqrt{w_{\max}/w_{LP}}$ and $\tilde{\mu} = \mathsf{Ex}(w_{ALG})/w_{\max} = w_{LP}/w_{\max} = \mathsf{Ex}(X)$, we obtain

$$\mathsf{Pr}\left(X < (1 - 2/3 \cdot \sqrt{w_{\max}/w_{LP}}) \cdot \mathsf{Ex}(X)\right) \leq e^{-2/9}$$

[1] Without loss of generality.

As $w_{ALG} = X \cdot w_{\max}$, we have

$$\mathsf{Pr}\left(w_{ALG} < (1-\delta)\cdot w_{LP}\right) \geq \mathsf{Pr}\left(w_{ALG} < (1-\delta)\cdot w_{OPT}\right)$$

which in turn implies that

$$\mathsf{Pr}\left(w_{ALG} < (1 - 2/3 \cdot \sqrt{w_{\max}/w_{LP}}) \cdot w_{OPT}\right) \leq e^{-2/9}$$

Lastly, by using the Taylor series expansion of the function $f(x) = e^x$, we obtain that $e^x \leq 1 + x + x^2/2$ holds for $x < 0$. Accordingly, $e^{-2/9} \leq 65/81$ holds. □

Now, we bound the edge capacity violation ratio of a single iteration of Steps 3-4 of Algorithm 2. We will use the Bernstein concentration bound (as stated in Fact 4.3; Boucheron *et al.* (2013b)) in order to prove a logarithmic bound on the edge capacity violation ratio while also handling non-uniform demands and weights, generalizing and improving the bounds in Liu *et al.* (2019).

Bernstein Concentration Bound: Given is a collection $\{Y_l\}_{l\in[k]}$ of $k \in \mathbb{N}$ independent binary random variables, i.e., $Y_l \in \{0,1\}$, with $\mu_l = \mathsf{Ex}\left(Y_l\right)$ for $l \in [k]$. Let $X_l = a_l \cdot Y_l$ for constant $0 < a_l \leq M$. The following holds for any $t > 0$ with $\theta = \left(\frac{t}{\sum_{l\in[k]} a_l^2\cdot(\mu_l - \mu_l^2) + M\cdot\frac{t}{3}}\right)$:

$$\begin{aligned}\mathsf{Pr}\left(\sum_{l\in[k]}\left(X_l - \mathsf{Ex}\left(X_l\right)\right) \geq t\right) &\overset{(a)}{\leq} e^{-\theta\cdot t}\cdot\prod_{l\in[k]}\mathsf{Ex}\left(e^{\theta\cdot(X_l - \mathsf{Ex}(X_l))}\right)\\ &\overset{(b)}{\leq} \exp\left(-\frac{t^2/2}{\sum_{l\in[k]} a_l^2\cdot(\mu_l-\mu_l^2) + M\cdot t/3}\right)\end{aligned}$$

This brings us to our edge capacity violation theorem:

Theorem 3. *Given an edge $e \in E$, consider the total flow $F^e = \sum_i d_i f_{i,e}$ on e, where $f_{i,e}$ is the flow on edge e for commodity i, resulting from one iteration of Steps 3-4 of Algorithm 2. The probability that F^e exceeds c_e by a factor of at least $(1 + j \log n)$ is upper bounded by $1/n^j$, where $j \geq \frac{6}{\log n}$.*

Proof. Fix an edge $e \in E$ and a commodity $i \in [k]$. With probability $1 - \tilde{f}_i$, the flow on edge e for commodity i is set to 0, i.e., $f_{i,e} = 0$. With probability $\tilde{f}_i$, the flow on edge e for commodity i is set to $\tilde{f}_{i,e} \cdot \frac{1}{\tilde{f}_i} \cdot d_i$. Then the expected value of f_i and the flow for commodity i on edge e are respectively

$$\mathsf{Ex}\,(f_i) = \tilde{f}_i, \quad \text{and} \quad \mathsf{Ex}\,(f_{i,e} \cdot d_i) = d_i \cdot ((\tilde{f}_{i,e} \cdot \frac{1}{\tilde{f}_i}) \cdot \tilde{f}_i + 0 \cdot (1 - \tilde{f}_i)) = \tilde{f}_{i,e} \cdot d_i \tag{4.8}$$

We have that $F^e = \sum_i f_{i,e} \cdot d_i$ and that $\mathsf{Ex}\,(F^e) = \sum_{i, \tilde{f}_{i,e} \neq 0} \tilde{f}_{i,e} \cdot d_i \leq c_e$. In order to bound $\mathsf{Pr}\,(F^e \geq (1 + j \log n) \cdot c_e)$, we apply Bernstein's inequality as stated in Fact 4.3 with

1. Indicator r.v. Y_i, for all $i \in [k]$, such that $Y_i = 1$ if $f_i = 1$, and 0 otherwise; it follows that $\mu_i = \mathsf{Ex}\,(Y_i) = \tilde{f}_i$
2. Constants $a_i = \frac{\tilde{f}_{i,e} \cdot d_i}{\tilde{f}_i}, \forall i \in [k]$
3. Constant $M = \max_i a_i = \max_i \frac{\tilde{f}_{i,e} \cdot d_i}{\tilde{f}_i} \leq c_e$ (inequality follows from Constraint 4.4 of LP)
4. Parameter $t = (j \log n) \cdot c_e$, where j is a positive constant.

Let the exponent on the upper bound given by Bernstein's inequality in Equation 3.37 be z, which upon substituting the above values becomes

$$z = -\frac{(j \log n)^2 \cdot c_e^2}{2 \sum_i \frac{\tilde{f}_{i,e}^2 \cdot d_i^2}{\tilde{f}_i^2} (\tilde{f}_i - \tilde{f}_i^2) + [2 \max_i (\frac{\tilde{f}_{i,e} \cdot d_i}{\tilde{f}_i}) \cdot (j \log n) \cdot c_e]/3} \tag{4.9}$$

Since $\frac{\tilde{f}_{i,e} \cdot d_i}{\tilde{f}_i} \leq c_e$, we obtain

$$z \leq -\frac{(j \log n)^2 \cdot c_e^2}{2 \sum_i c_e (\frac{\tilde{f}_{i,e} \cdot d_i}{\tilde{f}_i}) \tilde{f}_i (1 - \tilde{f}_i) + (2jc_e^2 \log n)/3} \leq -\frac{(j \log n)^2 \cdot c_e^2}{2c_e \sum_i \tilde{f}_{i,e} \cdot d_i + (2jc_e^2 \log n)/3} \tag{4.10}$$

where we arrive at the second inequality by bounding $(1 - \tilde{f}_i)$ by 1. Since $\sum_i d_i \tilde{f}_{i,e} \leq c_e$, we obtain

$$z \leq -\frac{(j \log n)^2 \cdot c_e^2}{2c_e^2 + (2jc_e^2 \log n)/3} = -\frac{3(j \log n)^2}{6 + 2j \log n} \tag{4.11}$$

If we assume that $6 \leq j \log n$, which holds e.g. for any $j \geq 2$ and $n \geq 8$, we have

$$z \leq -\frac{3(j \log n)^2}{j \log n + 2j \log n} \leq -\frac{(j \log n)^2}{j \log n} = -j \log n \tag{4.12}$$

We now upper bound the failure probability as follows:

$$\mathsf{Pr}\left(\sum_{i \in [k]} (X_i - \mathsf{Ex}(X_i)) \geq t\right) = \tag{4.13}$$

$$= \mathsf{Pr}\left(\sum_{i \in [k]} \left(\frac{\tilde{f}_{i,e} \cdot d_i}{\tilde{f}_i} \cdot f_i - \mathsf{Ex}\left(\frac{\tilde{f}_{i,e} \cdot d_i}{\tilde{f}_i} \cdot f_i\right)\right) \geq t\right) = \tag{4.14}$$

$$= \mathsf{Pr}\left(\sum_{i \in [k]} f_{i,e} d_i - \mathsf{Ex}\left(\sum_{i \in [k]} f_{i,e} d_i\right) \geq (j \log n) \cdot c_e\right) \leq e^{-j \log n} \tag{4.15}$$

Hence, since $\mathsf{Ex}(F_e) \leq c_e$,

$$\mathsf{Pr}(F^e - \mathsf{Ex}(F^e) \geq (j \log n) \cdot c_e) \leq e^{-j \log n} \Rightarrow \mathsf{Pr}(F^e \geq (1 + j \log n) \cdot c_e) \leq \frac{1}{n^j} \tag{4.16}$$

□

This result is for a particular edge e. To extend this result for the entire network G, we compute the union bound over all edges, of which there are at most n^2. Thus, we obtain the following bound for the edge capacity violation ratio β:

Corollary 1. *The probability that one execution of Steps 3-4 of Algorithm 2 exceeds any of the edge capacity constraints by a factor of at least* $(j \log n + 1)$ *is at most* $\frac{1}{n^{j-2}}$ *for any constant* $j \geq 3$ *and all* $n \geq 4$.

Using Theorem 2 and Corollary 1, we have that for $j = 3$ and any $n \geq 9$, the failure probability for a single round of execution of Steps 3-4 of the algorithm — i.e. the probability of not finding a solution with a 1/3-approximation on the throughput and an edge capacity violation ratio of $(3 \log n + 1)$ within a single round of the randomized algorithm — is bounded from above by $65/81 + 1/9 = 74/81$ (since $1/n^{j-2} = 1/n \leq 1/9$). The probability of finding a feasible solution within $c \log n$ rounds of Algorithm 2, where c is a constant, is then bounded from below by $1 - (74/81)^{c \log n} = 1 - \frac{1}{n^b}$, where b is a constant. Hence, Algorithm 2 gives a solution with a 1/3-approximation on the throughput and an edge capacity violation ratio of at most $3 \log n + 1$ with high probability.

Hence, choosing $j = 3$ and assuming the number of nodes to be at least 9 to satisfy the requirements of Corollary 1, we obtain our main result:

Theorem 4. *Assuming* $n \geq 9$*, the randomized rounding algorithm finds an* (α, β)*-approximate solution with* $\alpha = 1 - 2/3 \cdot \sqrt{w_{\max}/w_{LP}} \geq 1/3$ *and* $\beta = (3 \log n + 1)$ *within* $c \log n$ *iterations with high probability, where* c *is a positive constant.*

Chapter 5

DERANDOMIZATION AND THE DETERMINISTIC APPROACH

In this Section, we give a deterministic approach of the same problem by the derandomization of Algorithm 2. Our deterministic algorithm uses the method of pessimistic estimators first introduced by Raghavan (1988) to efficiently compute conditional expectations, which will guide the construction of the (α, β)-approximate solution. Given the analysis in Section 4, in the forthcoming analysis, we always assume $\alpha = 1 - 2/3 \cdot \sqrt{w_{\max}/Ex[w_{ALG}]} = 1 - 2/3 \cdot \sqrt{w_{\max}/w_{LP}} \geq 1/3$ and $\beta = 1 + 3 \cdot \log n$.

5.1 Pessimistic Estimator

We first introduce the following notation. Let $z_i = 0$ if Algorithm 2 has not selected commodity i to be routed, and let $z_i = 1$ if i was admitted. Now, let $\mathsf{fail}(z_1, \ldots, z_k) \rightarrow \{0, 1\}$ denote the failure function of not constructing an (α, β)-approximate solution, i.e., $\mathsf{fail}(z_1, \ldots, z_k) = 1$ if and only if the constructed solution either does not achieve an α-fraction of the LP's (weighted) throughput or some edge's capacity is exceeded by a factor larger than β. We use Z_i to denote the $\{0, 1\}$-indicator random variable for whether commodity i is routed or not in one execution of Steps 3-4 of Algorithm 2, i.e., $\mathsf{Pr}(Z_i = 1) = \tilde{f}_i$ and $\mathsf{Pr}(Z_i = 0) = 1 - \tilde{f}_i$. We have shown in Chapter 4 that $\mathsf{Ex}(\mathsf{fail}(Z_1, \ldots, Z_k)) < 1$ holds, implying the existence of an (α, β)-approximate solution. Given the above definitions, we employ the following notation to denote the conditional expectation of a function $\mathsf{f} : \{0, 1\}^k \rightarrow \{0, 1\}$:

$$\mathsf{Ex}(\mathsf{fail}(z_1, \ldots, z_i, Z_{i+1}, \ldots, Z_k)) = \mathsf{Pr}(\mathsf{fail}(Z_1, \ldots, Z_k) = 1 \mid Z_1 = z_1, \ldots, Z_i = z_i) \ .$$

As computing $\mathsf{Ex}\left(\mathsf{fail}(z_1, \ldots, z_i, Z_{i+1}, \ldots, Z_k)\right)$ is generally computationally prohibitive, we will now derive a pessimistic estimator $\mathsf{est} : \{0,1\}^k \to \mathbb{R}_{\geq 0}$, such that the following holds for all $i \in [k]$ and all $(z_1, \ldots, z_i) \in \{0,1\}^i$:

Upper Bound

$$\mathsf{Ex}\left(\mathsf{fail}(z_1, \ldots, z_i, Z_{i+1}, \ldots, Z_k)\right) \leq \mathsf{Ex}\left(\mathsf{est}(z_1, \ldots, z_i, Z_{i+1}, \ldots, Z_k)\right) \tag{5.1}$$

Efficiency

$$\mathsf{Ex}\left(\mathsf{est}(z_1, \ldots, z_i, Z_{i+1}, \ldots, Z_k)\right) \text{ can be computed efficiently.} \tag{5.2}$$

Furthermore, the estimator's value must initially always lie strictly below 1 to perform the derandomization:

Base Case

$$\mathsf{Ex}\left(\mathsf{est}(Z_1, \ldots, Z_k)\right) < 1 \text{ holds initially.} \tag{5.3}$$

5.2 Algorithm

In the following, we discuss how such a pessimistic estimator is used to derandomize the decisions of Algorithm 2 before introducing the actual estimator $\mathsf{est}_\beta^\alpha$ in Lemma 5. Algorithm 3 first computes an LP solution just as Algorithm 2, but then uses the pessimistic estimator to guide its decision towards deterministically constructing an approximate solution. Specifically, each commodity is either routed or rejected such that the conditional expectation $\mathsf{Ex}\left(\mathsf{est}_\beta^\alpha(z_1, ..., z_i, Z_i, ..., Z_n)\right)$ is minimized. Given that initially $\mathsf{Ex}\left(\mathsf{est}_\beta^\alpha(Z_1, \ldots, Z_k)\right) < 1$, this procedure terminates with a solution $(z_1, \ldots, z_k)$ such that the failure function $\mathsf{fail}(z_1, \ldots, z_k)$ is strictly upper bounded by 1. Specifically, $1 > \mathsf{Ex}\left(\mathsf{est}_\beta^\alpha(Z_1, \ldots, Z_k)\right) \geq \mathsf{Ex}\left(\mathsf{est}_\beta^\alpha(z_1, Z_2, \ldots, Z_k)\right) \geq \ldots \geq \mathsf{Ex}\left(\mathsf{est}_\beta^\alpha(z_1, \ldots, z_k)\right)$ is guaranteed and therefore, for the binary function fail,

$\mathsf{fail}(z_1, \ldots, z_k) = 0$ must hold. Furthermore, the algorithm is efficient (i.e., runs in polynomial time) as long as the pessimistic estimator function $\mathsf{est}^{\alpha}_{\beta}$ can be evaluated in polynomial time. W.l.o.g., we assume that only commodities that can be satisified in G are given as input to Algorithm 3.

Algorithm 3: Deterministic Approximation for the ANF Problem

Input : Directed Graph $G(V, E)$

Source-Sink Pair (s_i, t_i) for each satisfiable commodity $i \in [k]$

Capacity $c(u, v)$ $\forall (u, v) \in E$

Estimator $\mathsf{est}^{\alpha}_{\beta} : \{0,1\}^k \to \mathbb{R}_{\geq 0}$ for obtaining an (α, β)-approximate sol.

Output: (α, β)-approximate solution to the ANF instance

compute optimal solution $\vec{\tilde{f}}$ to LP (linear relaxation of MIP 1)

let the $\{0,1\}$-random variable Z_i be such that $\mathsf{Pr}(Z_i = 1) = \tilde{f}_i$ and $\mathsf{Pr}(Z_i = 0) = 1 - \tilde{f}_i$, for all $i \in [k]$

compute failure_estimate $\leftarrow \mathsf{Ex}\left(\mathsf{est}^{\alpha}_{\beta}(Z_1, ..., Z_{i-1}, Z_i, ..., Z_n)\right)$

foreach $i \in [k]$ **do** `// iterate over all commodities`

  if $\mathsf{Ex}\left(\mathsf{est}^{\alpha}_{\beta}(z_1, \ldots, z_{i-1}, 0, Z_{i+1}, ..., Z_n)\right) <$ failure_estimate **then**

    set $z_i \leftarrow 0$ `// commodity` i `is` ***not*** `routed`

  else

    set $z_i \leftarrow 1$ `// commodity` i `is routed`

  update failure_estimate $\leftarrow \mathsf{Ex}\left(\mathsf{est}^{\alpha}_{\beta}(z_1, \ldots, z_i, Z_{i+1}, \ldots, Z_n)\right)$

return *solution pertaining to selection* $\vec{z}$*:*

if $z_i = 1$ *then* $f_i = 1$ *and* $f_{i,e} = \tilde{f}_{i,e}/\tilde{f}_i$*, otherwise* $f_i = f_{i,e} = 0$*, for* $i \in [k]$

5.3 Analysis

Given the above intuition, we now introduce the following specific pessimistic estimator $\mathsf{est}_\beta^\alpha$ for which we will prove the above three correctness criteria (upper bound, efficiency, base case).

Lemma 5 (Pessimistic Estimator for the ANF)**.** *The function* $\mathsf{est}_\beta^\alpha$ *introduced below is a pessimistic estimator for the ANF.*

$$\mathsf{est}_\beta^\alpha(Z_1,\ldots,Z_k) = \mathsf{est}_\alpha(Z_1,\ldots,Z_k) + \sum_{(u,v)\in E} \mathsf{est}_\beta^{(u,v)}(Z_1,\ldots,Z_k)$$

$$\text{where } \mathsf{est}_\alpha(Z_1,\ldots,Z_k) = e^{-\theta_\alpha\cdot(1-\delta)\cdot\tilde{\mu}} \cdot \prod_i \mathsf{Ex}\left(e^{\theta_\alpha\cdot Z_i\cdot w_i/w_{\max}}\right),$$

$$\text{with } \delta = 2/3\cdot\sqrt{w_{\max}/w_{LP}}\,, \qquad \tilde{\mu} = w_{LP}/w_{\max}\,, \qquad \theta_\alpha = \ln(1-\delta),$$

$$\text{and } \mathsf{est}_\beta^{(u,v)}(Z_1,\ldots,Z_k) = e^{-\theta_\beta(u,v)\cdot t(u,v)} \cdot \prod_i \mathsf{Ex}\left(e^{\theta_\beta(u,v)\cdot(Z_i-\mathsf{Ex}(Z_i))}\right)$$

$$\text{with } t(u,v) = 4\cdot\log n\cdot c_{(u,v)}\,, \qquad a_i(u,v) = d_i\cdot\tilde{f}_{i,(u,v)}/\tilde{f}_i\,, \text{ and}$$

$$\theta_\beta(u,v) = t(u,v)/\left(\sum_i \left(a_i(u,v)\left(\mathsf{Ex}\left(Z_i\right) - \mathsf{Ex}\left(Z_i\right)^2\right)\right) + \max_i a_i(u,v)\cdot t(u,v)/3\right)$$

Proof. The following three properties are to be shown: (i) upper bound, (ii) efficiency, and (iii) base case (cf. Equations 5.1 - 5.3). We first discuss properties (i) and (iii).

The analysis in Chapter 4 has demonstrated that the probability of obtaining an (α,β)-approximate solution via randomized rounding is bounded from below by $1-74/81$. To obtain this result, a union bound argument was employed, which used probabilistic bounds on not achieving at least an α fraction of the optimal throughput and exceeding the capacity of each single edge by a factor of β.

For the throughput, the Chernoff bound of Theorem 3 was applied, while for each edge's capacity violation, the Bernstein bound of Theorem 4.3 was used. The pessimistic estimators est_α and $\mathsf{est}_\beta^{(u,v)}$ are a direct result of these respective theorems:

- est_α is obtained from the application of the Chernoff bound of Theorem 3 within the proof of Theorem 2. Specifically, the application of the Chernoff bound in Theorem 2 yielded the following — restated over the variables Z_i, with throughput given by $\sum_i Z_i$ $(= \sum_i f_i)$ — with the parameters δ, θ_α and $\tilde{\mu}$ as specified above:

  $$\mathsf{Pr}\left(\sum_{i\in[k]} Z_i < \alpha \cdot w_{OPT}\right) \leq e^{-\theta_\alpha\cdot(1-\delta)\cdot\tilde{\mu}} \cdot \prod_{i\in[k]} \mathsf{Ex}\left(e^{\theta_\alpha \cdot Z_i \cdot w_i/w_{\max}}\right) \leq e^{-2/9} \leq 65/81$$

  The middle expression directly yields the pessimistic estimator for the throughput.

- $\mathsf{est}_\beta^{(u,v)}$ is analogously obtained from the application of the Bernstein bound of Theorem 4.3 within the proof of Theorem 3 for each edge $(u,v) \in E$. Specifically, for a single edge (u,v), the following is obtained when setting $j = 3$ and using the constants defined above:

  $$\mathsf{Pr}\left(\sum_{i\in[k]} f_{i,(u,v)} > \beta \cdot c(u,v)\right) \leq e^{-\theta_\beta\cdot t(u,v)} \cdot \prod_{i\in[k]} \mathsf{Ex}\left(e^{\theta_\beta\cdot(Z_i - \mathsf{Ex}(Z_i))}\right) \leq n^{-3}$$

  Again, the middle expression is used to obtain the pessimistic estimator $\mathsf{est}_\beta^{(u,v)}$ for the specific edge $(u,v) \in E$.

Revisiting the union bound argument, we obtain that $\mathsf{est}_\beta^\alpha$ indeed yields an upper bound on the failure probability to construct an (α,β)-approximate solution, and that initially $\mathsf{Ex}\left(\mathsf{est}_\beta^\alpha(Z_1,\ldots,Z_k)\right) \leq 65/81 + n^{-1} < 1$ holds for $n \geq 9$. This shows that properties (i) and (iii) are satisfied.

Considering the efficiency property (ii), we note the following. Both est_α and $\mathsf{est}_\beta^{(u,v)}$ consist of products, where expectations for different commodities can be computed independently. Given the binary nature of the variables Z_i, these expectations can be computed in constant time. □

Theorem 5. *Using* $\mathsf{est}_\beta^\alpha$ *as a pessimistic estimator, the derandomized algorithm is a deterministic* (α, β)*-approximation for the ANF with* $\alpha = 1 - 2/3 \cdot \sqrt{w_{\max}/w_{LP}} \geq 1/3$ *and* $\beta = 1 + 3 \log n$ *for* $n \geq 9$.

Chapter 6

TRADING OFF THROUGHPUT AND CAPACITY VIOLATIONS

The deterministic approximations for the ANF given, achieve a throughput of $\alpha \geq 1/3$ and violate edge capacities by the logarithmic term $\beta \in \mathcal{O}(\log n)$. In the following, we discuss trading off the respective approximation objectives. Specifically, we show that by scaling the capacities by a factor $0 < c \leq 1$, a $(c \cdot \alpha, c \cdot \beta)$-approximation can be obtained whenever the achieved throughput is not rendered too small.

We first discuss the changes pertaining to the randomized algorithm 2. After pruning requests that cannot be embedded and computing the LP in Line 2, the flow values are scaled by the factor $0 < c \leq 1$ by setting $\tilde{f}'_i = c \cdot \tilde{f}_i$ and $\tilde{f}'_{i,e} = c \cdot \tilde{f}_{i,e}$. After scaling the flows, the scaled solution is valid with respect to the scaled down capacities $c'_e = c \cdot c_e$ and demands $d'_i = c \cdot d_i$. In the rounding phase of the algorithm, we round each flow f'_i to 1 with probability $\tilde{f}'_i = c \cdot \tilde{f}_i$ and zero otherwise. If $f'_i = 1$ then $f'_{i,e} = \tilde{f}'_{i,e}/\tilde{f}'_i$ and zero otherwise. This modified algorithm achieves a throughput w'_{ALG} such that $\mathsf{Ex}\,(w'_{ALG}) = c \cdot w_{LP}$.

A commodity was said to be valid for our solution only if it could be routed when present alone in the network. Note that, in the scaled solution all commodities may no longer be valid with respect to the new capacities. For the analysis in Theorem 2 to hold for the scaled down flows, we only only require that $w_{\max} \leq c \cdot w_{LP}$ holds as we show in Theorem 6. Assuming this condition to hold, the subsequent analyses remain valid and by randomized rounding a (α, β)-approximate solution regarding to the scaled down capacities and scaled down flows is obtained.

Reinterpreting the scaled solution with respect to the *original* capacities and the originally achieved LP throughput, it is easy to check that the obtained solution is indeed $(c \cdot \alpha, c \cdot \beta)$-approximate as we have shown in our proofs for Theorems 6 and 7.

Theorem 6. *After scaling the capacities of all the edges by a factor* $0 < c \leq 1$*, the probability of achieving less than* $(\frac{1}{3} \cdot c)$ *of the maximum throughput is at most* $e^{-2/9} \leq 65/81$*, for* $n \geq 9$*.*

Proof. We must calculate the difference between the throughput w'_{ALG} and the expected throughput $\mathsf{Ex}(w'_{ALG}) = \sum_i w_i(c \cdot \tilde{f}_i) = c \cdot w_{LP}$ of Algorithm 2 to observe α and the related probabilities. Let $w_{max} = \max_i w_i$. As assumed earlier, we continue with the proof taking $w_{max} \leq c \cdot w_{LP}$. We define the following (normalized) variables: Let $X'_i = w_i/w_{\max}$ if commodity i is routed and $X'_i = 0$ otherwise. Then $\mathsf{Pr}(X'_i = w_i/w_{\max}) = c \cdot \tilde{f}_i$ and $\mathsf{Pr}(X'_i = 0) = 1 - (c \cdot \tilde{f}_i)$. Let $X' = \sum_i X'_i$. Then we have that $\mathsf{Ex}(X') = \sum_i \mathsf{Ex}(X'_i) = \sum_i (c \cdot \tilde{f}_i) \cdot w_i/w_{\max}$ and that $w'_{ALG} = X' \cdot w_{\max}$. We also know that w_{LP} is an upper bound on the optimal achievable throughput w_{OPT}.

Noting that the variables X'_i conform with the requirements of Fact 3 and setting $\delta = 2/3 \cdot \sqrt{w_{\max}/c \cdot w_{LP}}$ and $\tilde{\mu} = \mathsf{Ex}(w'_{ALG})/w_{\max} = c \cdot w_{LP}/w_{\max} = \mathsf{Ex}(X')$, we obtain

$$\mathsf{Pr}\left(X' < (1 - 2/3 \cdot \sqrt{w_{\max}/c \cdot w_{LP}}) \cdot \mathsf{Ex}(X')\right) \leq e^{-2/9}$$

As $w'_{ALG} = X' \cdot w_{\max}$ and $c \cdot w_{LP}/w_{\max} = \mathsf{Ex}(X')$, we have

$$\mathsf{Pr}(w'_{ALG} < (1 - \delta) \cdot (c \cdot w_{LP})) \geq \mathsf{Pr}(w'_{ALG} < c \cdot (1 - \delta) \cdot w_{OPT})$$

Since, $\delta \leq 2/3$ due to our assumption that $w_{max} \leq c \cdot w_{LP}$, it follows that:

$$\mathsf{Pr}\left(w'_{ALG} < (\frac{1}{3} \cdot c) \cdot w_{OPT}\right) \leq e^{-2/9}$$

Lastly, by using the Taylor series expansion of the function $g(x) = e^x$, we obtain that $e^x \leq 1 + x + x^2/2$ holds for $x < 0$. Accordingly, $e^{-2/9} \leq 65/81$ holds. $\square$

Theorem 7. *After scaling the capacities of all the edges by a factor* $0 < c \leq 1$ *such that the new capacities are now denoted by* $c'_e = c \cdot c_e$*, given an edge* $e \in E$*, consider the total flow* $F'^e = \sum_i d'_i f'_{i,e}$ *on* e*, where* $f_{i,e}$ *is the flow on edge* e *for commodity* i*, resulting from one iteration of Steps 3-4 of Algorithm 2. The probability that* F'^e *exceeds* c_e *by a factor of at least* $c \cdot (1 + j \log n)$ *is upper bounded by* $1/n^j$*, where* $j \geq \frac{6}{\log n}$.

Proof. Fix an edge $e \in E$ and a commodity $i \in [k]$. With probability $1 - (c \cdot \tilde{f}_i)$, the flow on edge e for commodity i is set to 0, i.e., $f'_{i,e} = 0$. With probability $(c \cdot \tilde{f}_i)$, the flow on edge e for commodity i is set to $\tilde{f}_{i,e} \cdot \frac{1}{\tilde{f}_i}$. Then the expected value of f'_i and the flow for commodity i on edge e are respectively

$$\mathsf{Ex}\left(f'_i\right) = c \cdot \tilde{f}_i, \quad \text{and} \quad \mathsf{Ex}\left(f'_{i,e} \cdot d'_i\right) = d'_i \cdot ((\tilde{f}_{i,e} \cdot \frac{1}{\tilde{f}_i}) \cdot c \cdot \tilde{f}_i + 0 \cdot (1 - (c \cdot \tilde{f}_i))) = c \cdot \tilde{f}_{i,e} \cdot d'_i \quad (6.1)$$

We have that $F^e = \sum_i f'_{i,e} \cdot d'_i$ and that $\mathsf{Ex}\left(F^e\right) = \sum_{i, \tilde{f}_{i,e} \neq 0} \tilde{f'}_{i,e} \cdot d'_i \leq c'_e$. In order to bound $\mathsf{Pr}\left(F^e \geq c \cdot (1 + j \log n) \cdot c_e\right)$, we apply Bernstein's inequality as stated in Fact 4.3 with

1. Indicator r.v. Y_i, for all $i \in [k]$, such that $Y_i = 1$ if $f'_i = 1$, and 0 otherwise; it follows that $\mu_i = \mathsf{Ex}\left(Y_i\right) = c \cdot \tilde{f}_i = \tilde{f}'_i$
2. Constants $a_i = \frac{\tilde{f}'_{i,e} \cdot d'_i}{\tilde{f}'_i}, \forall i \in [k]$
3. Constant $M = \max_i a_i = \max_i \frac{\tilde{f}'_{i,e} \cdot d'_i}{\tilde{f}'_i} \leq c'_e$ (inequality follows from Constraint 4.4 of LP)
4. Parameter $t = (j \log n) \cdot c'_e$, where j is a positive constant.

Let the exponent on the upper bound given by Bernstein's inequality in Equation 3.37 be z, which upon substituting the above values becomes

$$z = -\frac{(j \log n)^2 \cdot c_e'^2}{2\sum_i \frac{\tilde{f}_{i,e}'^2 \cdot d_i'^2}{\tilde{f}_i'^2}(\tilde{f}_i' - \tilde{f}_i'^2) + [2\max_i(\frac{\tilde{f}_{i,e}' \cdot d_i'}{\tilde{f}_i'}) \cdot (j \log n) \cdot c_e']/3} \tag{6.2}$$

Since $\frac{\tilde{f}_{i,e} \cdot d_i'}{\tilde{f}_i} \leq c_e'$, we obtain

$$z \leq -\frac{(j \log n)^2 \cdot c_e'^2}{2\sum_i c_e'(\frac{\tilde{f}_{i,e}' \cdot d_i'}{\tilde{f}_i'}) \cdot \tilde{f}_i'(1 - \tilde{f}_i') + (\frac{2}{3} j c_e'^2 \log n)} \leq -\frac{(j \log n)^2 \cdot c_e'^2}{2c_e' \sum_i \tilde{f}_{i,e}' \cdot d_i' + (\frac{2}{3} j c_e'^2 \log n)} \tag{6.3}$$

where we arrive at the second inequality by bounding $(1 - \tilde{f}_i')$ by 1.

Since $\sum_i d_i' \tilde{f}_{i,e}' \leq c_e'$, we obtain

$$z \leq -\frac{(j \log n)^2 \cdot c_e'^2}{2c_e'^2 + (2j c_e'^2 \log n)/3} = -\frac{3(j \log n)^2}{6 + 2j \log n} \tag{6.4}$$

If we assume that $6 \leq j \log n$, which holds e.g. for any $j \geq 2$ and $n \geq 8$, we have

$$z \leq -\frac{(j \log n)^2}{j \log n + 2j \log n} \leq -\frac{(j \log n)^2}{j \log n} = -j \log n \tag{6.5}$$

We now upper bound the failure probability where $X_i = a_i \cdot Y_i$ as follows:

$$\mathsf{Pr}\left(\sum_{i \in [k]} (X_i - \mathsf{Ex}(X_i)) \geq t\right) = \tag{6.6}$$

$$= \mathsf{Pr}\left(\sum_{i \in [k]} \left(\frac{\tilde{f}_{i,e}' \cdot d_i'}{\tilde{f}_i'} \cdot f_i' - \mathsf{Ex}\left(\frac{\tilde{f}_{i,e}' \cdot d_i'}{\tilde{f}_i'} \cdot f_i'\right)\right) \geq t\right) = \tag{6.7}$$

$$= \mathsf{Pr}\left(\sum_{i \in [k]} f_{i,e}' d_i' - \mathsf{Ex}\left(\sum_{i \in [k]} f_{i,e}' d_i'\right) \geq (j \log n) \cdot c_e\right) \leq e^{-j \log n} \tag{6.8}$$

Hence, since $\mathsf{Ex}(F'^e) \leq c_e$ and $c_e' = c \cdot c_e$,

$$\mathsf{Pr}(F'^e - \mathsf{Ex}(F'^e) \geq (j \log n) \cdot c_e') \leq e^{-cj \log n} \Rightarrow \mathsf{Pr}(F'^e \geq c \cdot (1 + j \log n) \cdot c_e) \leq \frac{1}{n^j} \tag{6.9}$$

□

We are now ready to state our main result bounding the throughput approximation, in the presence of non-uniform commodity demands and weights. Given our above argument, we state the following:

Theorem 8. *By scaling the flows by a factor* $0 < c \leq 1$*, a deterministic* $(c \cdot \alpha, c \cdot \beta)$*-approximation for the ANF is obtained with* $\alpha = 1 - 2/3 \cdot \sqrt{w_{\max}/w_{LP}} \geq 1/3$ *and* $\beta = 1 + 3 \log n$ *for* $n \geq 9$ *as long as* $c \cdot w_{LP} \geq w_{\max}$ *holds.*

Assuming that all commodities can be routed using only a $\Theta(\log n)$-fraction of the capacities, the following corollary is obtained:

Corollary 2. *Assuming that all commodities are routable using an* $\Theta(\log n)$ *fraction of the original capacities in the absence of other commodities, then a* $(1/\Theta(\log n), \Theta(1))$*-approximate solution can be computed by scaling the flows by* $1/\Theta(\log n)$ *before rounding.*

Chapter 7

IMPLEMENTATION AND EXPERIMENTAL EVALUATION

7.1 Experiment Datasets

This section elaborates on the datasets we use to perform our experiments on, for both the Randomized and Derandomized algorithms.

Germany 50 Network

One of the networks that we use was the Germany 50 Network that we obtained from SNDLib; Orlowski *et al.* (2010). This network comprises of 50 nodes and 88 edges. There are 662 commodities to route through this network. The edge capacity of each of the edges given is 40. Also, we set the demands of all the commodities in general to be 50 and the weights to be 1. Later in our experiments we modify the weights and demands. This graph was originally used for a research project on telecommunication network operators in Germany. The topology of this graph as presented in Orlowski *et al.* (2010) is shown in Figure 7.1.

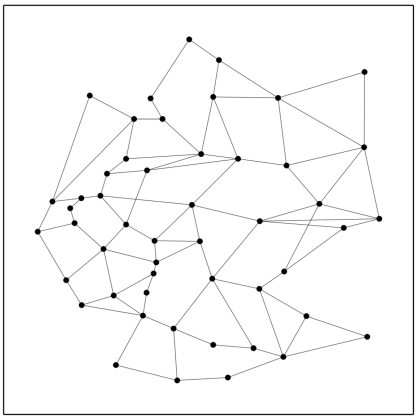

Figure 7.1: Topology of Germany 50 Network

Atlanta Network

Another network that we took was the Atlanta network which is again from SNDLib; Orlowski *et al.* (2010). This represents an ATM network in Los Angeles. The topology of the network as given by Orlowski *et al.* (2010) is shown in Figure 7.2. It is a relatively sparser graph than Germany 50 and has 15 nodes and 22 edges. We need to route 210 commodities through this network and we assigned the demand for each commodity to be 50 and the weight for each to be 1. Also, the capacity for each edge is 40. We will have experiments later which modify these graphs to have varying demands, weights or edge capacities.

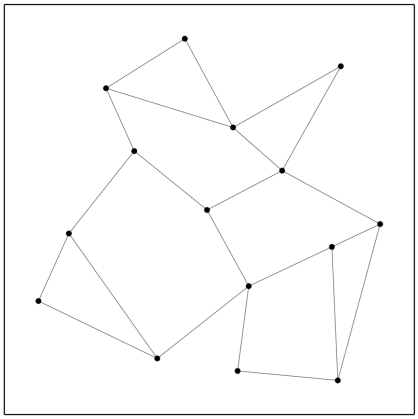

Figure 7.2: Topology of Atlanta Network

7.2 Randomized Algorithm Results

This section focuses on the implementation of our randomized algorithm, the code for which is present in Appendix A.1. There are less than 200 lines of code (CPLEX 12.10 and Python) which strengthens our claim of the algorithm being really simple to implement and execute. The implementation time varies depending on the graph. The run time is only 30 seconds for the Atlanta graph with 1000 iterations of the randomized rounding while for the Germany-50 graph takes about 30 minutes. We will elaborate more on the specific experiments carried out in the following section.

(α,β) Distribution

For the first experiment, we repeated the randomized rounding involved in our algorithm 1000 times and recorded the α and β values calculated for each iteration. The first graph as depicted in Figure 7.3 shows the distribution of (α, β) over these iterations on the Germany 50 Network. Similarly, we see the distribution for the Atlanta Network in Figure 7.4. The star denotes the optimal value i.e., both α and β are 1 and is only for display purposes. The triangle denotes the result we obtained from our deterministic algorithm. We have proven that theoretically our β can be proportional to $1 + 3\log n$ which for the German 50 Network is 12.74 and 9.1 for the Atlanta Network. However, here in the graphs we see it is much less in practice both for the randomized and deterministic algorithm.

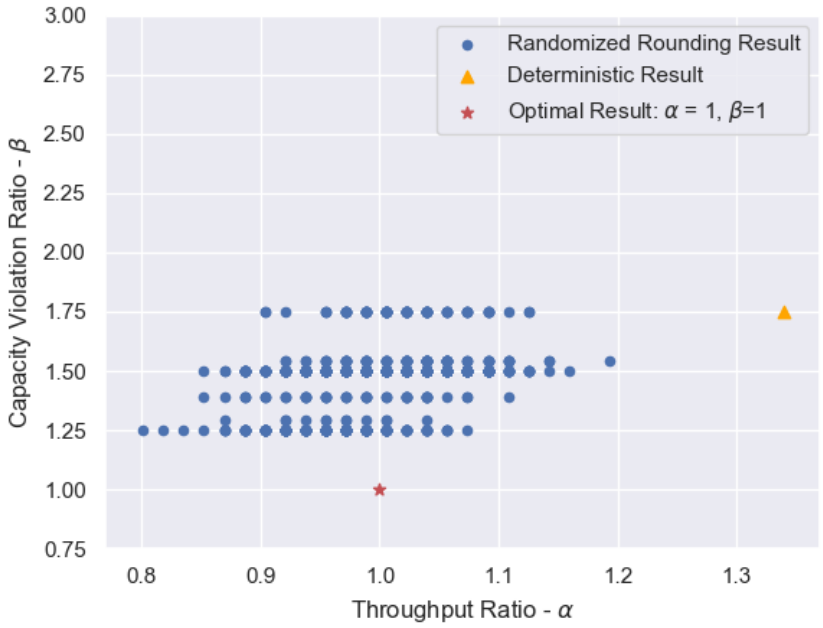

Figure 7.3: (α,β) – distribution for Germany 50 Network

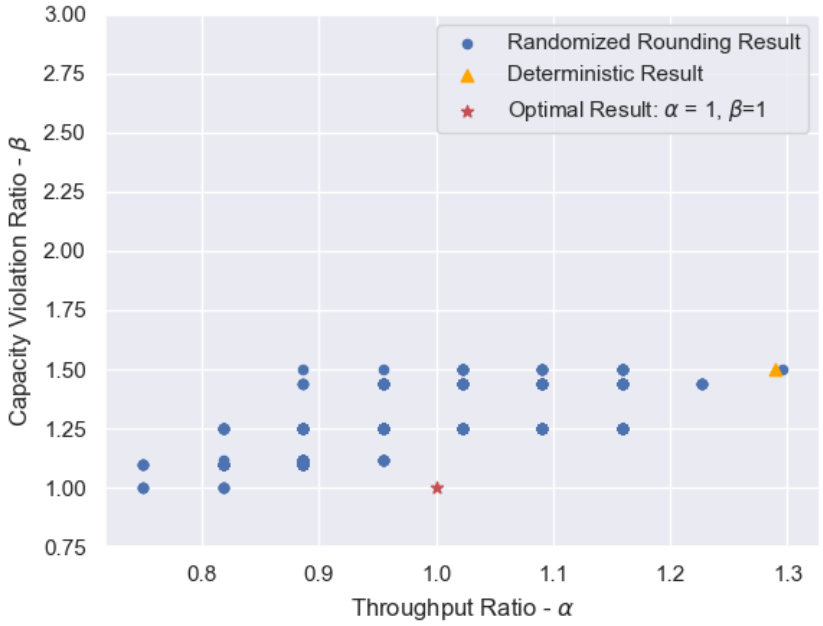

Figure 7.4: (α,β) – distribution for Atlanta Network

(α,β) Distance

Further, we calculate the (α, β) distance for each iteration i.e., the euclidean distance between each pair and the optimal value (1,1). This is plotted as a histogram in Figure 7.5 for the German 50 Network where each block is of 0.05 range. Here, we can see that all the values we receive are within 0.7 distance of the optimum value and more than 90% within 0.55 distance. We can see the related histogram for the Atlanta Network in Figure 7.6 where all the values we receive are within 0.55 distance of the optimum value and more than 90% within 0.45 distance.

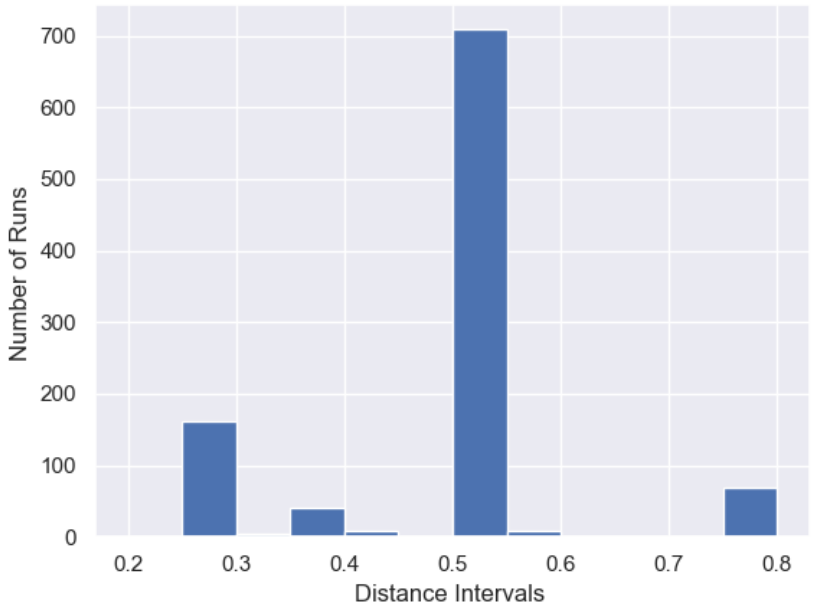

Figure 7.5: (α,β) – distances for Germany 50 Network

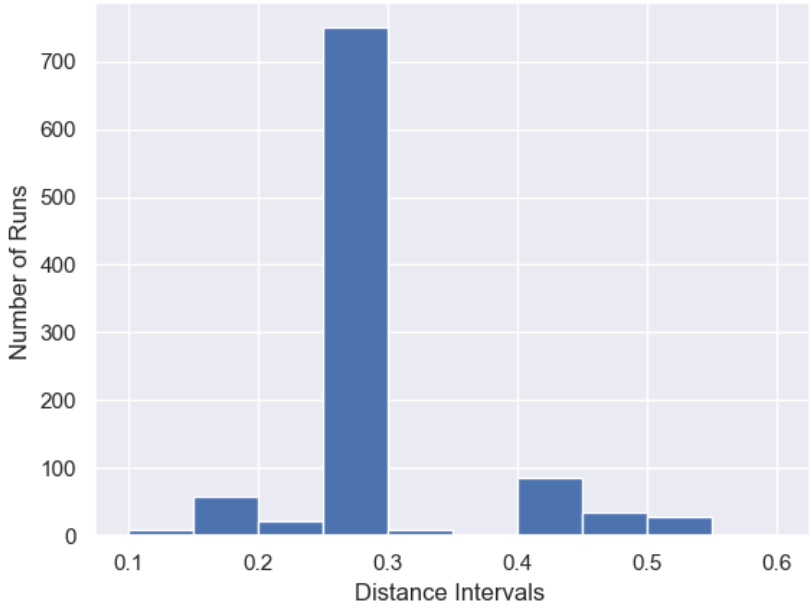

Figure 7.6: (α,β) – distances for Atlanta Network

Varying Weights histogram

Next, for performing our experiment we change the weights that were all 1s till now and assign weights from $[1, 2, 3, 4]$ to each commodity so that there is an equal distribution of commodities for each weight while keeping all demands to be 50. As we described earlier in Chapter 1 through the video example, the weights are almost like priorities assigned to each commodity as the larger weight commodities contribute more to the objective function. We see a supporting trend over here. On plotting the commodities which got routed as a histogram depending on the weights we can see clearly in Figure 7.7 for the German 50 Network that the commodities with higher weight were favoured even though all commodities had equal demands 50. We see a similar trend for Atlanta Network in Figure 7.8.

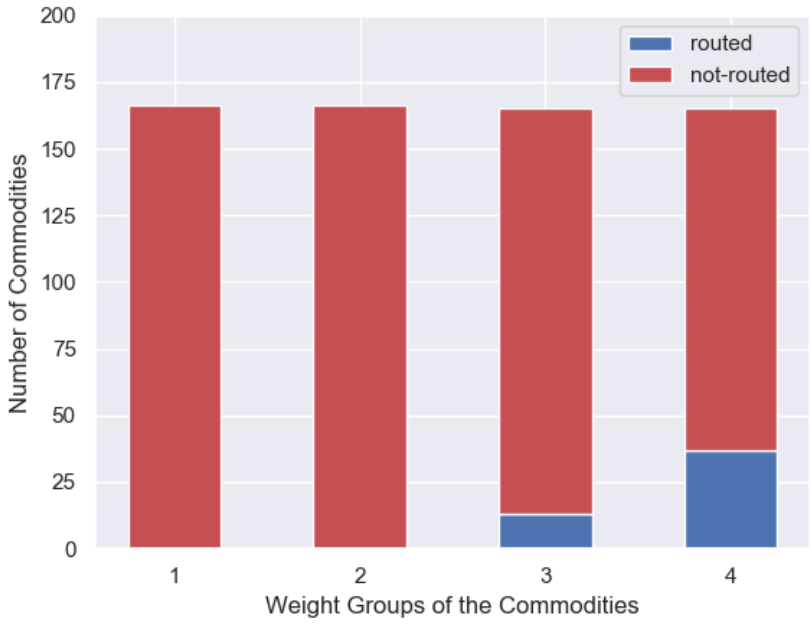

Figure 7.7: Routed Commodities Distribution based on Weight Groups - Germany

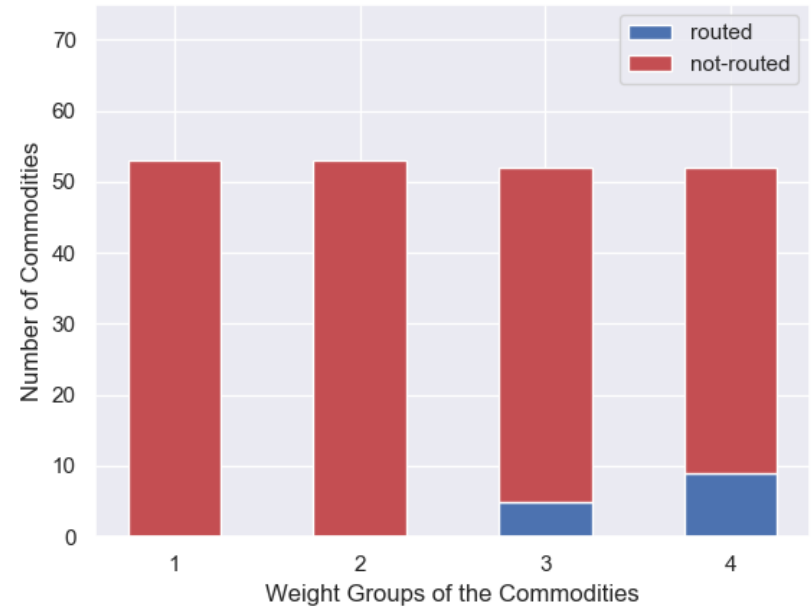

Figure 7.8: Routed Commodities Distribution based onWeight Groups - Atlanta

Varying Demands histogram

We do a similar experiment again but now with varying demands and the same weight 1 for each commodity. The commodities were uniformly divided into four groups each having demands from $[50, 60, 70, 80]$. We know that a demand is proportional to the amount of flow that a commodity requires in order to be satisfied. The related histogram for the German 50 Network can be seen in Figure 7.9 and in Figure 7.10 for the Atlanta Network. We see that the lower demand commodities were favoured more. This is because even though they contribute the same to the objective function but more commodities are routed if we have lower demand commodities flowing through the network.

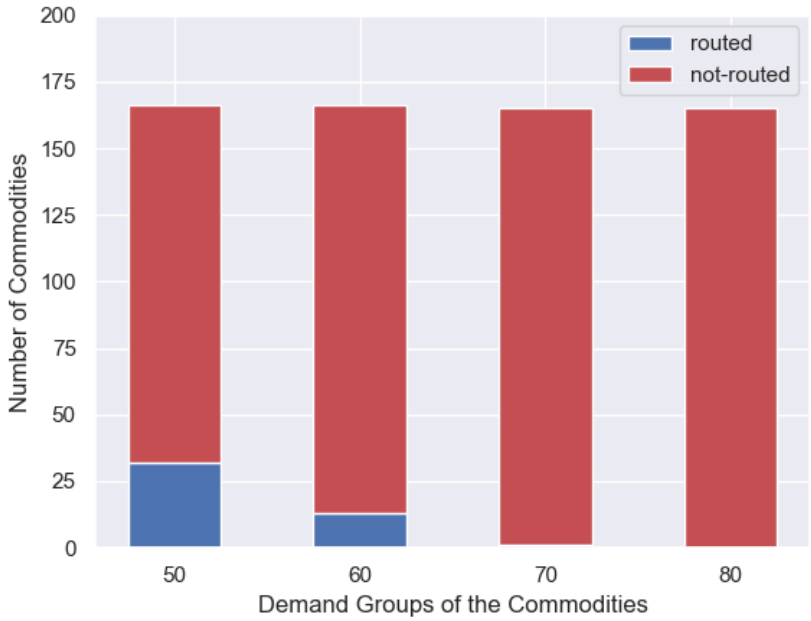

Figure 7.9: Routed Commodities Distribution based on Demand Groups - Germany

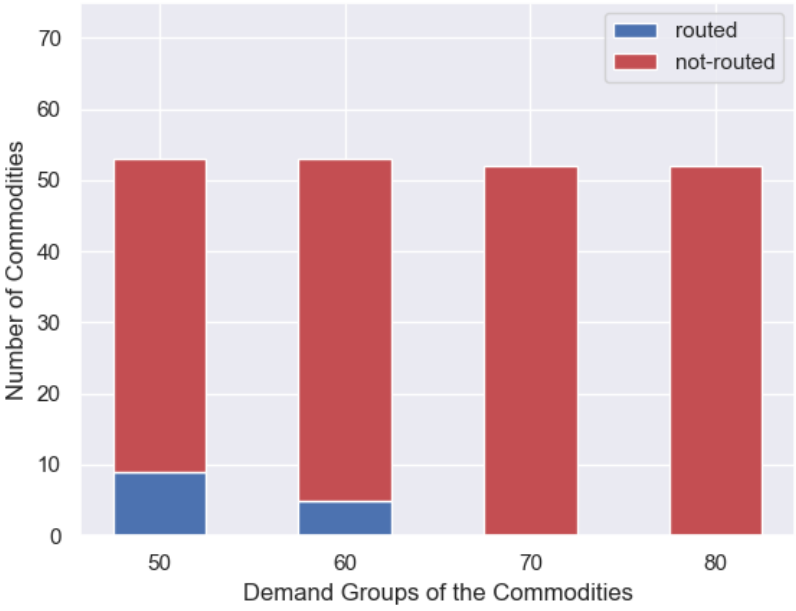

Figure 7.10: Routed Commodities Distribution based on Demand Groups - Atlanta

7.3 Deterministic Algorithm Results

In this section, we present the results we obtained by running our deterministic algorithm. We use the same two graphs as before even for this algorithm. The program is of less than 250 lines of code but as a trade off for removing the randomization the execution time for this is definitely more as compared to the randomized algorithm's implementation. The program takes less than 3 minutes to compute the result for the Atlanta Network and we obtain $\alpha = 1.29$ and $\beta = 1.5$ which are well within our theoretical bounds. This has been shown in Figure 7.4.

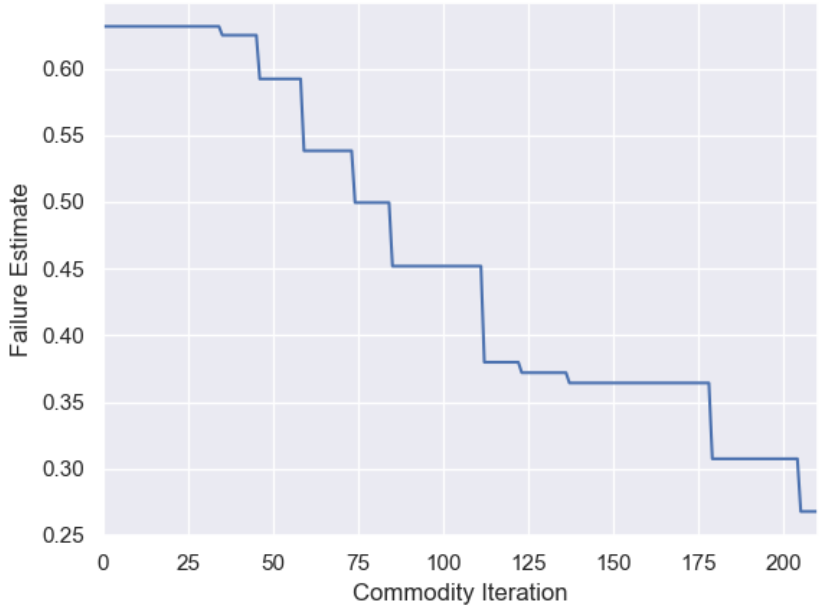

Figure 7.11: Decreasing Failure Estimate Function as we fix flow values - Atlanta

In our deterministic algorithm, we calculate an estimator function called failure_estimate which we use as an upper bound on our failure probability. As we move through this algorithm and set our flows for each commodity to be either 0 or 1 we need to proceed in such a way that our estimator function value keeps reducing. We confirm this by keeping a track of all the failure estimates calculated for each iteration when we ran it for the Atlanta Network. We found the same to be monotonically decreasing as can be seen in Figure 7.11.

We ran the algorithm similarly for the Germany 50 Network. This execution takes about 4 hours however, the results we got were still within our theoretical bounds. The α value was 1.34 and the β value was 1.75 as shown in Figure 7.3. We also plotted the estimator value similar as before. The corresponding graph can be seen in Figure 7.12.

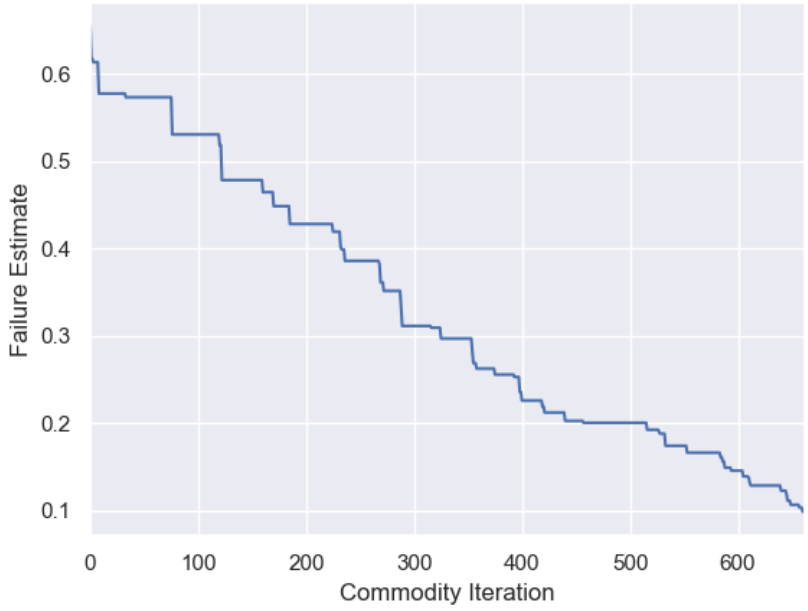

Figure 7.12: Decreasing Failure Estimate Function as we fix flow values - Germany

Chapter 8

CONCLUSION

We conclude by going over the major contributions made in this thesis work. First, we present the first polynomial-time algorithm for the ANF problem with *non-uniform commodity demands and weights* in *arbitrary directed graphs*, which achieves at most a *(poly)logarithmic violation ratio of the edge capacities* while ensuring a *constant approximation on the maximum throughput*, with high probability. More specifically, we present, a $(1/3, 3\log n+1)$-*approximation algorithm* for the ANF problem in *arbitrary directed graphs* with *arbitrary commodity demands and weights*.

Second, we show how to *derandomize* our algorithm, based on pessimistic estimators for Bernstein's inequality (see Fact 4.3) which can be viewed as a generalization of Chernoff's bound. The derandomized algorithm also runs in polynomial time and achieves the same approximation bounds as the randomized algorithm.

Lastly, our algorithms improve and generalize the randomized rounding framework outlined in Liu *et al.* (2019), and hence are simpler and more practical than many of the other approximation algorithms for the ANF problem that are based on Räcke trees and other more complex hierarchical decompositions of fractional multicommodity flows (see e.g., Chekuri *et al.* (2004)). This is confirmed by our implementation and experiments for both algorithms elaborated in Chapter 7. Moreover, the implementation generally provides better guarantees than our conservative analytical bounds predict.

REFERENCES

Arora, S., C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof verification and the hardness of approximation problems", J. ACM **45**, 3, 501–555, URL https://doi-org.ezproxy1.lib.asu.edu/10.1145/278298.278306 (1998).

Boucheron, S., G. Lugosi and P. Massart, *Concentration inequalities: A nonasymptotic theory of independence* (Oxford university press, 2013a).

Boucheron, S., G. Lugosi and P. Massart, *Concentration Inequalities: A Nonasymptotic Theory of Independence* (OUP Oxford, 2013b), URL https://books.google.com/books?id=koNqWRluhP0C.

Chekuri, C. and A. Ene, "The all-or-nothing flow problem in directed graphs with symmetric demand pairs", Math. Program. **154**, 1-2, 249–272, URL https://doi.org/10.1007/s10107-014-0856-z (2015).

Chekuri, C., S. Khanna and F. B. Shepherd, "The all-or-nothing multicommodity flow problem", in "Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing", STOC '04, p. 156–165 (Association for Computing Machinery, New York, NY, USA, 2004), URL https://doi.org/10.1145/1007352.1007383.

Chekuri, C., S. Khanna and F. B. Shepherd, "Multicommodity flow, well-linked terminals, and routing problems", in "Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005", edited by H. N. Gabow and R. Fagin, pp. 183–192 (ACM, 2005), URL https://doi.org/10.1145/1060590.1060618.

Chuzhoy, J., V. Guruswami, S. Khanna and K. Talwar, "Hardness of routing with congestion in directed graphs", in "Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing", STOC '07, p. 165–178 (Association for Computing Machinery, New York, NY, USA, 2007), URL https://doi.org/10.1145/1250790.1250816.

Erlebach, T. and K. Jansen, "The maximum edge-disjoint paths problem in bidirected trees", SIAM Journal on Discrete Mathematics **14**, 3, 326–355, URL https://doi.org/10.1137/S0895480199361259 (2001).

Garg, N., V. V. Vazirani and M. Yannakakis, "Primal-dual approximation algorithms for integral flow and multicut in trees", Algorithmica **18**, 1, 3–20 (1997).

Guruswami, V., S. Khanna, R. Rajaraman, B. Shepherd and M. Yannakakis, "Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems", Journal of Computer and System Sciences **67**, 3, 473 – 496, URL http://www.sciencedirect.com/science/article/pii/S0022000003000667 (2003).

Kleinberg, J. M. and M. X. Goemans, *Approximation Algorithms for Disjoint Paths Problems*, Ph.D. thesis, USA, aAI0597431 (1996).

Kolliopoulos, S. G. and C. Stein, "Approximating disjoint-path problems using greedy algorithms and packing integer programs", in "Integer Programming and Combinatorial Optimization", edited by R. E. Bixby, E. A. Boyd and R. Z. Ríos-Mercado, pp. 153–168 (Springer Berlin Heidelberg, Berlin, Heidelberg, 1998).

Liu, M., A. Richa, M. Rost and S. Schmid, "A constant approximation for maximum throughput multicommodity routing and its application to delay-tolerant network scheduling", in "INFOCOM 2019 - IEEE Conference on Computer Communications", Proceedings - IEEE INFOCOM, pp. 46–54 (Institute of Electrical and Electronics Engineers Inc., 2019).

Mitzenmacher, M. and E. Upfal, *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis* (Cambridge University Press, USA, 2017), 2nd edn.

Orlowski, S., R. Wessäly, M. Pióro and A. Tomaszewski, "Sndlib 1.0—survivable network design library", Netw. **55**, 3, 276–286 (2010).

Räcke, H., "Optimal hierarchical decompositions for congestion minimization in networks", in "Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing", STOC '08, p. 255–264 (Association for Computing Machinery, New York, NY, USA, 2008), URL `https://doi.org/10.1145/1374376.1374415`.

Raghavan, P., "Probabilistic construction of deterministic algorithms: Approximating packing integer programs", Journal of Computer and System Sciences **37**, 2, 130 – 143, URL `http://www.sciencedirect.com/science/article/pii/0022000088900037` (1988).

Rost, M., E. Döhne and S. Schmid, "Parametrized complexity of virtual network embeddings: Dynamic & linear programming approximations", ACM SIGCOMM Computer Communication Review **49**, 3–10 (2019).

Rost, M. and S. Schmid, "Virtual network embedding approximations: Leveraging randomized rounding", in "2018 IFIP Networking Conference (IFIP Networking) and Workshops", pp. 1–9 (2018).

Srinivasan, A., "Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems", in "Proceedings 38th Annual Symposium on Foundations of Computer Science", pp. 416–425 (1997).

APPENDIX A

CODE LISTING

```
import matplotlib.pyplot as plt
import numpy as np
import docplex.mp.model as cpx
import random as rnd
from copy import deepcopy
import math
import time

#Notations for the paper
#f_{i} --> f_i_round
#f_{i,e} --> f_i_e_round
#\tilde{f_{i}} --> f_i_vars
#\tilde{f_{i,e}} --> f_i_e_vars

#temporary global values for Germany50 from SNDlib
v = 50
e = 88
k= 50
constant = 100
alpha = 1.0/3
beta = 1

#repeated randomized rounding function
def rounding(f_i_vars,f_i_e_vars,edges):
    beta = 1
    f_i_round = {}
    f_i_e_round = {}
    for i in range(k):
        prob = rnd.uniform(0,1)
        if prob<=f_i_vars[i].solution_value:
            f_i_round[i] = 1.0
            #rescaling step
            for edge in edges:
                flow_edge = float(f_i_e_vars[(i,edge)].
   solution_value)
                flow_commodity = float(f_i_vars[i].solution_value)
                f_i_e_round[(i,edge)] = flow_edge/flow_commodity
        else:
            f_i_round[i] = 0.0
            for edge in edges:
                f_i_e_round[(i,edge)] = 0
    return f_i_round, f_i_e_round


#main
opt_model = cpx.Model(name="MIP Model")


#Reading the graph file
fgraph = open("germany50converted.txt","r")
graph = fgraph.read().split("\n")[:-1]

#Segregating the respective values
sourcesink = []
```

```
demands = []
weights = []
edges = [(i,j) for i in range(v) for j in range(v)]
capacity = []
for i in range(v):
    row = []
    for j in range(v):
        row.append(0)
    capacity.append(row)

for c in range(len(graph)):
    temp = graph[c].split(",")
    if c<k:
        sourcesink.append((int(temp[0]),int(temp[1])))
        demands.append(float(temp[2]))
        weights.append(float(temp[3]))
    else:
        capacity[int(temp[1])][int(temp[0])]= float(temp[2])

capacity = np.array(capacity)
print(capacity)

flows = []

for i in range(k):
    row = []
    for j in range(e):
        row.append(0)
    flows.append(row)

flows = np.array(flows)

# setting the constant edge capacities for the linear program
print("Adding the variables:")
start=time.time()
c_e = {edges[i]:capacity[edges[i][0]][edges[i][1]] for i in range(v
   **2)}

# setting the variable flow values for the linear program
f_i_vars = {i:opt_model.continuous_var(lb=0, ub=1, name="f_{0}".
   format(i))

for i in range(k)}

f_i_e_vars = {(i,edge):opt_model.continuous_var(lb=0, name="f_{0}_
   {1}".format(i,edge))

for i in range(k) for edge in edges}
end=time.time()
print("Time taken = ",end-start," seconds")

# adding the constraints for the linear program
print("Now adding constraints:")
start=time.time()
constraint1 = {i :
opt_model.add_constraint(
ct=opt_model.sum(f_i_e_vars[(i,(sourcesink[i][0],j))] for j in range
```

```
   (v)) == f_i_vars[i],
ctname="constraint1_{0}".format(i))
    for i in range(k)}

constraint2 = {(i,j) :
opt_model.add_constraint(
ct=opt_model.sum(f_i_e_vars[(i,(a,j))] for a in range(v) ) ==
   opt_model.sum(f_i_e_vars[(i,(j,a))] for a in range(v)),
ctname="constraint2_{0}_{1}".format(i,j))
    for i in range(k) for j in range(v) if j is not sourcesink[i][0]
    and j is not sourcesink[i][1]}

 #Constraint 3 for directed graph and varying demands
constraint3 = {(a,b) :
 opt_model.add_constraint(
 ct=opt_model.sum(f_i_e_vars[(i,(a,b))]*demands[i] for i in range(k)
   ) <= c_e[(a,b)] ,
 ctname="constraint3_{0}_{1}".format(a,b))
    for a in range(v) for b in range(v)}

# Constraint 4 for varying demands
constraint4 = {(i,(a,b)) :
opt_model.add_constraint(
ct=f_i_e_vars[(i,(a,b))]*demands[i] <= c_e[(a,b)]*f_i_vars[i] ,
ctname="constraint4_{0}_{1},{2}".format(i,a,b))
   for i in range(k) for a in range(v) for b in range(v)}

constraint5 = {(i,(a,b)) :
opt_model.add_constraint(
ct=f_i_e_vars[(i,(a,b))] >= 0 ,
ctname="constraint5_{0}_{1},{2}".format(i,a,b))
    for i in range(k) for a in range(v) for b in range(v)}

# objective function for the linear program in presense of varying
   weights
objective = opt_model.sum(weights[i]*f_i_vars[i]
                          for i in range(k))
end=time.time()
print("Time taken = ",end-start," seconds")


# for maximization
opt_model.maximize(objective)

#for exporting the lp as text constraints
opt_model.export_as_lp("./model.lp")

#solving the linear program
print("Now solving:")
start=time.time()
opt_model.solve()
end=time.time()
print("Time taken = ",end-start," seconds")
print("="*150)
#print lp results
print("The flow values assigned by the LP to each commodity are:")
x = {}
```

```
[x.update({i:f_i_vars[i].solution_value}) for i in range(k)]
print(x)
print("\n")

print("The flow values for each edge assigned by the LP to each
    commodity are:")
x=[tuple([float(f_i_e_vars[(i,e)].solution_value) for i in range(k)
    ]) for e in edges]
for i in range(v):
    print(x[v*i:v*(i+1)])
print("\n")

print("The objective function value given by the LP is:",objective.
    solution_value)
print("="*150)
rounds = constant * math.ceil(math.log(v))
round_obj=0.0

#Randomized Rounding
if(objective.solution_value!=0):
    for t in range(rounds):
        round_obj=0.0
        f_i_round, f_i_e_round = rounding(f_i_vars,f_i_e_vars,edges)
        for key in f_i_round.keys():
            round_obj+=f_i_round[key]*weights[key]
        temp_alpha = float(round_obj)/float(objective.solution_value
    )
        if temp_alpha >= alpha:
            alpha = temp_alpha
            break
else:
    print("Objective function value is zero!")
    exit()

#Calculating Beta
for e in edges:
    f_e=0
    if c_e[e] != 0:
        for i in range(k):
            f_e+=f_i_e_round[i,e]*demands[i]
        if f_e>c_e[e]:
            temp_beta = float(f_e)/float(c_e[e])
            if temp_beta > beta:
                beta = temp_beta


print("The flow values assigned by the randomized algorithm to each
    commodity are:")
print(f_i_round)
print("\n")

print("The flow values for each edge assigned by the randomized
    algorithm to each commodity are:")
x=[tuple([f_i_e_round[(i,e)] for i in range(k)]) for e in edges]
for i in range(v):
    print(x[v*i:v*(i+1)])
print("\n")
```

```
print("The objective function value given by the algorithm is:",
   round_obj )

print("Alpha is calculated as ",alpha)

print("Beta is calculated as ",beta)

print("="*150)
```

A.2 Derandomized Algorithm

```
import matplotlib.pyplot as plt
import numpy as np
import docplex.mp.model as cpx
import random as rnd
from copy import deepcopy
import math
import time
from estimators import * #for calculating pessimistic estimators for
    derandomization
from tqdm import tqdm
#Notations for the paper
#f_{i} --> f_i_round
#f_{i,e} --> f_i_e_round
#\tilde{f_{i}} --> f_i_vars
#\tilde{f_{i,e}} --> f_i_e_vars

#temporary global values for Germany50 from SNDlib
v = 50
e = 88
k= 662
constant = 100
alpha = 1.0/3
beta = 1

#main
opt_model = cpx.Model(name="MIP Model")

#Reading the graph file
fgraph = open("data/germany50converted.txt","r")
graph = fgraph.read().split("\n")[:-1]

#Segregating the respective values
sourcesink = []
demands = []
weights = []
edges = [(i,j) for i in range(v) for j in range(v)]
capacity = []
for i in range(v):
    row = []
    for j in range(v):
        row.append(0)
    capacity.append(row)

for c in range(len(graph)):
    temp = graph[c].split(",")
```

```
    if c<k:
        sourcesink.append((int(temp[0]),int(temp[1])))
        demands.append(float(temp[2]))
        weights.append(float(temp[3]))
    else:
        capacity[int(temp[1])][int(temp[0])]= float(temp[2])
        capacity[int(temp[0])][int(temp[1])]= float(temp[2])

capacity = np.array(capacity)
print(capacity)

flows = []

for i in range(k):
    row = []
    for j in range(e):
        row.append(0)
    flows.append(row)

flows = np.array(flows)

# setting the constant edge capacities for the linear program
print("Adding the variables:")
start=time.time()
c_e = {edges[i]:capacity[edges[i][0]][edges[i][1]] for i in range(v
   **2)}

# setting the variable flow values for the linear program
f_i_vars = {i:opt_model.continuous_var(lb=0, ub=1, name="f_{0}".
   format(i))

for i in range(k)}

f_i_e_vars = {(i,edge):opt_model.continuous_var(lb=0, name="f_{0}_
   {1}".format(i,edge))

for i in range(k) for edge in edges}
end=time.time()
print("Time taken = ",end-start," seconds")

# adding the constraints for the linear program
print("Now adding constraints:")
start=time.time()
constraint1 = {i :
opt_model.add_constraint(
ct=opt_model.sum(f_i_e_vars[(i,(sourcesink[i][0],j))] for j in range
   (v)) == f_i_vars[i],
ctname="constraint1_{0}".format(i))
    for i in range(k)}

constraint2 = {(i,j) :
opt_model.add_constraint(
ct=opt_model.sum(f_i_e_vars[(i,(a,j))] for a in range(v) ) ==
   opt_model.sum(f_i_e_vars[(i,(j,a))] for a in range(v)),
ctname="constraint2_{0}_{1}".format(i,j))
    for i in range(k) for j in range(v) if j is not sourcesink[i][0]
    and j is not sourcesink[i][1]}
```

```
#Constraint 3 for directed graph and varying demands
constraint3 = {(a,b) :
opt_model.add_constraint(
ct=opt_model.sum(f_i_e_vars[(i,(a,b))]*demands[i] + f_i_e_vars[(i,(b
   ,a))]*demands[i] for i in range(k)) <= c_e[(a,b)] ,
ctname="constraint3_{0}_{1}".format(a,b))
   for a in range(v) for b in range(v)}

# Constraint 4 for varying demands
constraint4 = {(i,(a,b)) :
opt_model.add_constraint(
ct=f_i_e_vars[(i,(a,b))]*demands[i] <= c_e[(a,b)]*f_i_vars[i] ,
ctname="constraint4_{0}_{1},{2}".format(i,a,b))
   for i in range(k) for a in range(v) for b in range(v)}

constraint5 = {(i,(a,b)) :
opt_model.add_constraint(
ct=f_i_e_vars[(i,(a,b))] >= 0 ,
ctname="constraint5_{0}_{1},{2}".format(i,a,b))
    for i in range(k) for a in range(v) for b in range(v)}

# objective function for the linear program in presense of varying
   weights
objective = opt_model.sum(weights[i]*f_i_vars[i]
                          for i in range(k))
end=time.time()
print("Time taken = ",end-start," seconds")


# for maximization
opt_model.maximize(objective)

#solving the linear program
print("Now solving:")
start=time.time()
opt_model.solve()
end=time.time()
print("Time taken = ",end-start," seconds")
print("="*150)
#print lp results
print("The flow values assigned by the LP to each commodity are:")
x = {}
[x.update({i:f_i_vars[i].solution_value}) for i in range(k)]
print(x)
print("\n")

print("The flow values for each edge assigned by the LP to each
   commodity are:")
x=[tuple([float(f_i_e_vars[(i,e)].solution_value) for i in range(k)
   ]) for e in edges]
for i in range(v):
    print(x[v*i:v*(i+1)])
print("\n")

print("The objective function value given by the LP is:",objective.
   solution_value)
```

```
print("="*150)

if(objective.solution_value==0):
    print("Objective function value is zero!")
    exit()
flows = []
w_lp = 0

for i in range(k):
    flows.append(f_i_vars[i].solution_value)
    w_lp += weights[i]*(f_i_vars[i].solution_value)

edgeflows = {}

for i in range(k):
    for edge in edges:
        edgeflows[(i,edge)] = f_i_e_vars[(i,edge)].solution_value

f_i_vars_val = deepcopy(flows)
f_i_e_vars_val = deepcopy(edgeflows)
failure_estimate = estimator_alphabeta(f_i_vars_val,f_i_e_vars_val,
   flows,edgeflows,demands,weights,c_e,w_lp)
print(-1,failure_estimate)
failest_csv = open("data/germany_failest.csv","w")
failest_csv.write("commodity,failure_estimate\n")
failest_csv.write(f"-1,{failure_estimate}\n")
for i in tqdm(range(k)):
    tempflows = deepcopy(flows)
    tempedgeflows = deepcopy(edgeflows)
    tempflows[i] = 0
    for edge in edges:
        tempedgeflows[(i,edge)] = 0
    temp_estimate = estimator_alphabeta(f_i_vars_val,f_i_e_vars_val,
   tempflows,tempedgeflows,demands,weights,c_e,w_lp)
    if(temp_estimate <= failure_estimate):
        flows[i] = 0
        for edge in edges:
            edgeflows[(i,edge)] = 0

    else:
        for edge in edges:
            edgeflows[(i,edge)] = edgeflows[(i,edge)]/float(flows[i
   ])
        flows[i] = 1

    failure_estimate = estimator_alphabeta(f_i_vars_val,
   f_i_e_vars_val,flows,edgeflows,demands,weights,c_e,w_lp)
    failest_csv.write(f"{i},{failure_estimate}\n")

failest_csv.close()
print("The flow values assigned by the Derandomized Algorithm to
   each commodity are:")
x = {}
[x.update({i:flows[i]}) for i in range(k)]
print(x)
print("\n")
```

```
print("The flow values for each edge assigned by the Derandomized
    Algorithm to each commodity are:")
x=[tuple([float(edgeflows[(i,e)]) for i in range(k)]) for e in edges
    ]
for i in range(v):
    print(x[v*i:v*(i+1)])
print("\n")

obj_derand = 0

for i in range(k):
    obj_derand += weights[i]*flows[i]

print("The objective function value given by the Algorithm is:",
    obj_derand)
print("="*150)

beta = 0.0
#Calculating Beta
for e in edges:
    f_e=0
    if c_e[e] != 0:
        for i in range(k):
            f_e+=edgeflows[(i,e)]*demands[i]
        if f_e>c_e[e]:
            temp_beta = float(f_e)/float(c_e[e])
            if temp_beta > beta:
                beta = temp_beta

alpha = obj_derand/float(objective.solution_value)

print("Alpha = ",alpha)
print("Beta = ",beta)
```

A.2.1 Estimator Functions

```
import math
import numpy as np

def estimator_alpha(flows,weights,w_lp):

    k = len(weights)
    w_max = np.max(weights)

    delta = (2.0/3)*math.sqrt(float(w_max)/w_lp)
    mu = float(w_lp)/w_max
    theta = math.log(1-delta)

    coeff = math.exp(-theta*(1 - delta)*mu)
    prod = 1

    for i in range(k):
        expectation_i = flows[i]*(math.exp(theta*weights[i]/float(
   w_max)))
        expectation_i += (1-flows[i])
        prod *= expectation_i
```

```
    est_alpha = coeff*prod

    return est_alpha

def estimator_beta(f_i_vars,f_i_e_vars,flows,edgeflows,demands,edge,
    c_uv,v):

    k = len(demands)

    t = 4*math.log(v,2)*c_uv

    a = []
    varsum = 0

    for i in range(k):
            if(f_i_vars[i]==0):
                a.append(0.0)
            else:
                a.append(demands[i]*(f_i_e_vars[(i,edge)])/float(
    f_i_vars[i]))
            varsum += a[i] * (flows[i] - (flows[i]*flows[i]))

    theta_denom = varsum + np.max(a)*float(t)/3
    if(theta_denom==0):
        return 0.0

    theta = float(t)/theta_denom

    coeff = math.exp(-theta*t)
    prod = 1

    for i in range(k):
        expectation_i = flows[i]*(math.exp(theta*(1-flows[i])))
        expectation_i += (1-flows[i])*(math.exp(theta*(-flows[i])))
        prod *= expectation_i

    est_beta = coeff*prod

    return est_beta

def estimator_alphabeta(f_i_vars,f_i_e_vars,flows,edgeflows,demands,
    weights,c_e,w_lp):

    est = estimator_alpha(flows,weights,w_lp)

    v = math.sqrt(len(c_e.keys()))

    for edge in c_e.keys():
        est += estimator_beta(f_i_vars,f_i_e_vars,flows,edgeflows,
    demands,edge,c_e[edge],v)

    return est
```