APPROXIMATION ALGORITHMS FOR NETWORK DESIGN AND PARTITIONING PROBLEMS

by

Siddharth Barman

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2012

Date of final oral examination: 06/08/12

The dissertation is approved by the following members of the Final Oral Committee:

Shuchi Chawla, Assistant Professor, Computer Sciences
Carl Eric Bach, Professor, Computer Sciences
Stark Draper, Assistant Professor, Electrical and Computer Engineering
Benjamin Recht, Assistant Professor, Computer Sciences
Stephen Wright, Professor, Computer Sciences

To my teacher, Dwivedi ji

Acknowledgments

I would first like to thank my advisor Shuchi Chawla for her guidance and generous support. I am grateful for the independence she provided me to freely pursue research problems that I found interesting. Working with her has been an invaluable experience and it has profoundly shaped my approach to research. I will always admire the speed with which she grasps the essence of things.

I owe many thanks to the faculty at the University of Wisconsin–Madison; in particular, Eric Bach, Jin-Yi Cai, Shuchi Chawla, Stark Draper, Benjamin Recht, Dieter van Melkebeek, and Stephen Wright. I have learned a lot from their courses and I thank them for all the time they have spent teaching me and serving as a reference on many occasions. I have benefited profoundly from working with with Eric and Steve. I hope one day I am as knowledgeable and humble as them. I would like to express my gratitude to Eric for his open-door policy and letting me pick his brain, every so often, for ideas and references. I am grateful to have had the opportunity of working with Stark and Ben. I thank them for their continuing advice and providing me with a broader view of research. It is a pleasure to work with Seeun Umboh; he is a valuable collaborator and I thank him for the numerous hours we spent working together. Working with Yeye He and Xishuo Liu has been a fruitful and enjoyable experience.

It am thankful to Matt, Piramanayagam, Seeun for making Madison a fun and exciting place for me. They are great friends and digressing with them has been an integral part of my graduate-school experience. I am grateful to Matt for all the useful discussions and enormous help along the way. His attention to detail and diligence is inspiring. I will certainly miss playing tennis with Piramanayagam. It is hard to find players that are as "good" as me and who, without exception, bring their "B game" to the court. I thank Seeun for introducing me as the weirdest guy he knows; somehow both of us consider this to be high praise. I grateful to Tyson for "organizing" our theory lunches. I have certainly enjoyed having Balu and David as officemates. They have ensured that that a few conversations in our office were not goofy. Rathijit brought some maturity to our group and I enjoyed our discussions on politics, Higgs bosons, and immortal jellyfishes; I hope at least he knew what we were talking about. Special thanks to Anshuman, Sarvagya, and Vinay for their friendship that has spanned many miles and years.

Didi and Marc are a great source of advice and free food. I am grateful to them for taking me along on vacations and hosting me so many times. Now that I have my Ph.D., I hope,

they will ask me to do fewer chores. Above all, I thank my parents for their unconditional love and support. I thank my mother for teaching me the value and importance of hard work.

I dedicate this thesis to my poet and teacher, Dwivedi ji. He continually inspires me to become a better researcher. I realize that mere words will not suffice; to thank him properly I must lead a life befitting his teachings.

Contents

List of Tables

List of Figures

Abstract

We develop new approximation algorithms for three graph-theoretic optimization problems. Two of these problems arise in communication networks and the third comes up in the context of classification and labeling. Below we describe these problems and present an overview of our main results.

Information Network Design: We develop a model for information networks with a cost structure that captures savings obtained by redundant-data elimination. This model presents an expressive and algorithmically interesting framework as it allows us to represent information flow and still maintain the tractable nature of classical network-design. We consider two problems within this framework and develop algorithms that achieve logarithmic and constant-factor approximation for them.

Multi-Route Cuts: A fundamental problem in combinatorial optimization is to find a low-cost cut which disconnects the underlying graph. A natural generalization of finding small cuts is the multi-route cut problem where the goal is to determine a low-cost set of edges or nodes whose removal reduces the connectivity of the graph to below a certain threshold. This problem arises in the context of reliability of service in networks. We provide the first non-trivial approximations for variants of the problem. When the connectivity thresholds are either two or infinity, we obtain polylogarithmic approximations to cost. For arbitrary thresholds, we develop bicriteria approximation algorithms; in particular, we obtain approximations to cost while ensuring that the connectivity drops below a constant times the threshold.

Packing Multiway Cuts: Problems involving classification and labeling of interconnected objects arise in many contexts such as machine learning and computational biology. An important class of labeling problems reduces to packing cuts in a graph where the goal is to

determine nearly-disjoint cuts that satisfy containment constraints. We develop constant-factor approximation algorithms for the *multiway cut packing problem*; where, given a collection of subsets of vertices, the objective is to produce separating cuts (one for each subset) that are as edge disjoint as possible.

1 Introduction

Graph-theoretic optimization problems are ubiquitous and their algorithms play a prominent role in computer science. Given a graph the general goal is to determine a discrete set of objects (edges/vertices) that satisfy some constraints and have minimum cost or maximum value. Historically, these problems were formulated to study transshipment, flow, assignment, and transportation of physical objects. These problems arise in numerous other areas including physical sciences, engineering, and economics. See [Sch03] and [Sch05] for a comprehensive treatment of the subject.

Many natural and practically-relevant instances of graph-theoretic optimization are NP-Hard. Therefore, it is unlikely that efficient methods exist that find exact solutions for them. This drives us to explore algorithms that efficiently determine provably-good approximate solutions. This area of study is called *approximation algorithms*. An algorithm is said to achieve an approximation ratio of $\alpha \geq 1$ if, for any input instance of the problem, it produces a solution of cost no more than α times the optimal. The study of algorithms for graph-theoretic problems has successfully generated many practical techniques and significantly influenced algorithm design in other fields such as online computation, learning theory, and algorithmic game theory.

Despite notable advances in approximation algorithms for graph-theoretic problems, our understanding of optimization in the *digital realm* is far from complete. Classical graph-theoretic models do not represent computational systems that deal with data and information in their entirety. We find stark examples of this limitation when considering design of information networks. For instance, in a network design problem we are given a graph with cost on edges and the goal is to determine a minimum cost subgraph that satisfies certain connectivity constraints (see [GK11] for a survey on algorithms for network design). A common class of network design problems involves multiple source-sink pairs with demands

and the objective of determining a minimum cost subgraph which can route the specified demands. We can use classical flow models if the network design problem is formulated to represent a physical network, like incompressible fluids in pipes. In particular, an optimal solution of the multiple source-sink network design problem with physical flow is simply a collection of shortest paths connecting each source to the corresponding sink. On the other hand, for network design over information networks, we need to address completely different problems. Consider the case in which we can simultaneously transmit the demand of different source-sink pairs along any edge, i.e. an edge of unit capacity can satisfy multiple source-sink pairs each with unit demand. This telecommunication inspired network design problem is called the Steiner network or survivable network design problem [WGMV95, GGP$^+$94, Jai01] and, unlike the physical flow version, is NP-Hard. Jain [Jai01] developed an approximation algorithm for this problem that uses novel techniques and achieves an approximation ratio of two.

Recent past has seen the development of such novel algorithms in multiple areas of graph-theoretic optimization, including in online [MSVV07], stochastic [GKR09], and economic [RT02] settings. The overarching objective of these results is to obtain insights into the vast collection of optimization problems that arise in computational systems. In this dissertation we move towards this goal and develop new approximation algorithms for three graph-theoretic optimization problems that come up in communication networks and data classification.

In Sections 1.2, 1.3, and 1.4 we outline the three problems. The body of the dissertation presents a detailed description of the problems and formally proves our results.

1.1 Approximation Algorithms

The problems considered in this dissertation are NP-Hard. Efficient algorithms that exactly solve NP-Hard problems do not exist unless P = NP, hence we must consider alternatives. One formal and well-established approach to dealing with NP-Hardness is to develop algorithms that determine provably-good approximate solutions in polynomial time; that is, we relax optimality in order to achieve efficiency. Such algorithms are called approximation algorithms and the guarantee they provide on the quality of solution is specified in terms of an approximation ratio. Specifically, we consider worst-case bounds for the ratio between the cost/value of the solution generated by the algorithm and the optimal. Write OPT as the optimal value of a minimization problem with objective function c and the feasible set (implicitly specified by the input in most cases) $\mathcal{F}$. That is, OPT is the minimum cost attained by the following program:

$$\begin{aligned} \text{minimize} \quad & c(S) \\ \text{subject to} \quad & S \in \mathcal{F} \end{aligned}$$

An algorithm achieves an approximation ratio of $\alpha \geq 1$ if, for any instance, it generates a solution in $\mathcal{F}$ of cost no more than α OPT. Along these lines, an α-approximation algorithm for a value-maximization problem is one which always generates a feasible solution of value no less than $\frac{1}{\alpha}$OPT.

Some NP-Hard problems have high approximation thresholds; for example, independent set cannot be approximated within a factor of n^c, for some constant c, unless P = NP. In addition, there are problems for which standard relaxations and approaches do not yield good approximations. A natural course of action in such situations is to relax the feasibility constraint or augment the algorithm with extra resources while keeping the benchmark

as an optimal solution without augmentation. Specifically, say we have a minimization problem and one of its constraints is of the form $f(S) \leq k$, for some function $f()$ and value k. An algorithm is said to achieve an (β, α)-*bicriteria approximation* if for any instance of the minimization problem it produces a solution $\mathcal{A}$ that satisfies $f(\mathcal{A}) \leq \beta k$ and has cost no more than αOPT. Appropriate changes are made to this definition for maximization problems and constraints of the form $f(S) \geq k$.

1.2 Information Network Design

We develop a combinatorial model which captures data flow in information networks. In particular, we consider a cost structure in which the data-redundancy in traffic can be leveraged to save on routing costs. Within this framework we develop algorithms with strong approximation guarantees for network design and facility location problems. This cost structure models information flow and still maintains the tractable nature of commodity (physical-flow carrying) network design; therefore, it presents an expressive and algorithmically interesting class of problems.

The first problem we study is the redundancy aware network design (RAND) problem. We are given an edge-weighted graph containing a single server and many clients. The server owns a number of different data packets and each client desires a subset of the packets. Our goal is to connect every client to the server via a single path, such that the collective cost of the resulting network is minimized. Here the transmission cost over an edge is its weight times times the number of *distinct* packets that it carries. For example, if the edge belongs to two paths that each carry the same packet, then the edge only needs to route the packet once rather than twice. RAND arises in networks that face a lot of data duplication. Anand et al. [AGA+08] show that this kind of traffic redundancy is highly prevalent in the Internet, and it can be eliminated across individual links by routers employing packet

caches. We consider instances where the demand sets form a laminar family, i.e., any two demand sets in the family are either disjoint or one is contained in the other. We develop a combinatorial algorithm that achieves an $O(\log P)$ approximation for RAND, where P is the total number of distinct packets. We note that P is always at most the number of different demand sets desired or the number of clients, and is generally much smaller.

The second problem we study is a facility location problem that we call RAFL (redundancy aware facility location). Here the goal is to find an assignment from clients to facilities such that the total cost of routing packets from the facilities to clients plus the total cost of "producing" one copy of each desired packet at each facility is minimized. Note that in RAFL data redundancy is leveraged at the facilities and not in routing. This problem is motivated by the data allocation requirements of content distribution networks (CDNs) in the Internet. For RAFL, with laminar demands, we presents a multi-phase LP rounding algorithm that achieves a constant approximation ratio.

This is joint work with Shuchi Chawla and appeared in the Symposium on Discrete Algorithms (SODA), 2012 [BC12].

1.3 Multi-Route Cuts

A fundamental problem in combinatorial optimization is to find a low-cost cut which disconnects the underlying graph. A natural generalization of finding small cuts is the multi-route cut problem where the goal is to determine a low-cost set of edges or vertices whose removal reduces the connectivity of the graph to below a certain threshold. This problem arises in the context of reliability of service in networks. Specifically, given a graph $G = (V, E)$ and connectivity thresholds $k_{(u,v)}$ on pairs of nodes, the goal of the minimum multi-route cut problem is to find a minimum cost set of edges or vertices whose removal reduces the connectivity between every pair (u, v) to strictly below its given threshold.

Note that in traditional minimum cut problems the thresholds are either one (we want to completely separate the pair) or ∞ (we do not care about the connectivity for the pair).

We propose a new linear programming formulation for this problem and provide the first non-trivial approximations to several versions of the problem. A main contribution of our work is an extension of the region growing technique for approximating minimum multicuts to the multi-route setting. When the connectivity thresholds are either two or ∞ (the "2-route cut" case), we obtain an $O(\log^2 h)$ approximation, where h is the number source-sink pairs. For arbitrary connectivity thresholds, we present a number of bicriteria-approximation algorithms that achieve different cost-connectivity tradeoffs.

This is joint work with Shuchi Chawla and appeared in the Symposium on Discrete Algorithms (SODA), 2010 [BC10].

1.4 Packing Multiway Cuts

Problems involving classification and labeling of interconnected objects arise in many contexts such as machine learning and computational biology. A common instance is one where we are given a partial labeling of a set of items along with a neighborhood structure over them and the objective is to complete the labeling in the most consistent way possible. These problems naturally translate into a graph-theoretic setting. Finding a consistent labeling is reduces to packing cuts in a graph where the goal is to determine nearly-disjoint subsets (of graph vertices) that satisfy containment constraints.

We consider the multiway cut packing (MCP) problem. In MCP we are given an undirected graph $G = (V, E)$ and k commodities, each corresponding to a set of terminals located at different vertices in the graph; our goal is to produce a collection of cuts $\{C_1, \cdots, C_k\}$ such that C_i is a multiway cut for commodity i (that is, C_i separates all pairs of terminals in commodity i) and the maximum load on any edge is minimized. The load

on an edge is defined to be the number of cuts in the solution crossing the edge. In the capacitated version of the problem edges have capacities c_e and the goal is to minimize the maximum *relative* load on any edge – the ratio of the edge's load to its capacity.

MCP is NP-Hard and we present a constant-factor approximation for it. We consider a linear programming relaxation for the problem and show that every instance of the problem admits a near-optimal fraction solution in which no pair of cuts cross, i.e., the cuts are laminar. We use this insight to develop a rounding algorithm which guarantees a maximum edge load of at most $8\mathrm{OPT} + 4$. For the special case where each commodity has only two terminals and all commodities share a common sink (the "common sink s-t cut packing" problem) we guarantee a maximum load of $\mathrm{OPT} + 2$. Since the special case is also NP-hard this is nearly the best possible approximation unless $\mathrm{P} = \mathrm{NP}$.

This is joint work with Shuchi Chawla and appeared in the Symposium on Discrete Algorithms (SODA), 2009 [BC09].

2 Information Network Design

2.1 Introduction

We consider network design problems for information networks where edges can replicate data but cannot otherwise alter it. In this setting our goal is to exploit the redundancy in the given traffic matrix to save on routing costs. Formally, we are given a graph over a single server and many clients. The server has a universe of data packets available, and each client desires a subset of the packets. The goal is to determine a collection of paths, one from the source to each client, such that the total cost of routing is minimized. Here the cost of routing on an edge is proportional to the total size of the *distinct* packets that the edge carries. For example, if the edge belongs to two paths that each carry the same packet, then the edge only needs to route the packet once and not twice. We call this the traffic-redundancy aware network design problem, or RAND for short.

RAND arises in networks that face a lot of data duplication. Consider for example a Netflix server serving movies to a large and varied clientele. Each client desires a certain subset of the movies. What routing paths should the server use to send the data to the clients so as to minimize its total bandwidth usage? If different movie streams involve disjoint sets of packets this boils down to setting up a single multicast tree, or solving the minimum Steiner tree problem, once per movie. However, the server may want to set up a single routing path for each client regardless of how many different movies the client desires. Moreover, clients desiring the same movie may desire it at different rates or qualities depending on their location or the device they are using. For example, a desktop user with a broadband connection may desire a high definition video, whereas a mobile phone user may be content with a much lower resolution. Then, the data sent to these clients is not identical but has some amount of overlap. The server can exploit this redundancy in traffic

by using common paths for clients with similar demands, thereby saving on the actual amount of traffic routed. Redundancy in data can arise even across different movies when data streams are broken down into small enough packets. Anand et al. [AGA+08] show that this kind of traffic redundancy is highly prevalent in the Internet, and it can be eliminated across individual links by routers employing packet caches.

Given the cost structure that this redundancy generates, it makes sense to try to route the demands of clients desiring similar sets along overlapping paths. An extreme example of the benefit of merging paths is when all clients desire the same set of packets. In this case, the problem becomes equivalent to finding the minimum-cost Steiner tree over the clients and the server. At the other extreme, if all the clients desire disjoint sets of packets, then merging does not help at all, and it is optimal to pick the shortest path from every client to the source. There is thus a trade-off between routing demands along shortest paths and trying to merge the paths of clients with similar demands.

We also study a facility location version of the problem that we call traffic-redundancy aware facility location or RAFL. This problem is motivated by the prevalence of content distribution networks (CDNs) in the Internet. Netflix servers, instead of connecting to clients directly, cache their data at multiple servers spread around the network that are hosted by a CDN such as Akamai; each client then connects to a CDN server individually to obtain its data. The savings in this case comes from assigning clients with similar demand sets to the same CDN server and sending a single copy of the multiply desired packets to the server. Formally we are given a network over potential facilities and clients. Each client, as before, desires a subset of the available packets. Our goal is to assign each client to a facility and route to each facility the union of the demand sets desired by clients assigned to it. The cost of such a solution is the sum of the cost of routing packets from facilities to clients (that are proportional to the size of the respective client's demand set), and the cost of routing packets to facilities (that is proportional to the total size of the distinct packets

being routed to the facility). The savings from redundancy in this case are realized in the facility opening costs where we assign multiple clients with similar demand sets to the same facility and pay for each of the common packets only once.

RAND and RAFL model information networks as opposed to commodity networks in traditional network design problems. They can therefore be considered as intermediate models between traditional network design and network coding. In the latter the information network is allowed to use coding to increase its capacity. In our context the network can eliminate redundant information but cannot otherwise alter the information. See [nwc] and references therein for work on network coding.

2.1.1 Results and Techniques

We study the RAND and RAFL under a laminar demands assumption. In particular, we assume that the sets of packets demanded by clients form a laminar set family. In other words, every pair of demanded packet sets is either disjoint or one is a subset of the other. Such a structure arises, for example, in the Netflix problem described above when layered coding is used; the packets for a lower encoding rate are a subset of the packets for a higher encoding rate.

RAND generalizes the minimum Steiner tree problem and RAFL generalizes metric uncapacitated facility location (MUFL); therefore both problems are NP-hard. We study approximations.

We develop a constant factor approximation for the RAFL based on the natural LP-relaxation of the problem. Our algorithm follows the filtering approach developed by Lin and Vitter [LV92] and later exploited by Shmoys et al. [STA97] in the context of MUFL. In the MUFL setting, in the filtered solution, each client t is associated with a set of facilities, say $\mathcal{F}(t)$, such that the routing cost of any of those facilities is comparable to the routing

cost that the client pays in the LP solution. It is then sufficient to open a set of facilities in such a way that each can be charged to a client with a distinct set of associated facilities. Clients t that are not charged (a.k.a. free riders) are rerouted to the facilities opened for other charged clients t' such that $\mathcal{F}(t)$ and $\mathcal{F}(t')$ overlap, and furthermore the routing cost for t' is smaller than that for t. Then, using the triangle inequality, the cost of rerouting can be bounded and this gives a constant factor approximation. In our setting, it is not sufficient to ensure that the routing cost of t' is smaller than that of t. In addition, we must ensure that the facility opened by t' can support the demand of t, otherwise every time we reroute a client we incur extra facility costs. Ensuring these two properties is tricky because clients with low routing costs may also have small demand sets; so essentially, these properties require us to consider clients according to two distinct and potentially conflicting orderings.

In order to deal with this issue, our algorithm is run in two phases. In the first phase we consider clients in order of increasing routing costs, in order to determine which clients will pay for their facilities and which ones are free riders. In the second phase, we consider free riders in decreasing order of the sizes of their demand sets. Each free rider is associated with a set of paying clients that pay for the facility that this client opens. Every time a free rider opens a facility, we reroute to it all of the other clients whose paying neighbors overlap with those of this facility. In this manner we can ensure that whenever a client is rerouted, it is routed to a facility that already produces the packets it needs. Unfortunately, our algorithm charges each paying client multiple times for different facilities opened. We ensure that the costs that a client pays each time it is charged form a geometrically decreasing sequence, and can therefore be bounded in terms of the LP solution. Overall we obtain a 27-approximation.

For RAND an $O(\log n)$ randomized approximation can be obtained via tree embeddings [FRT03] because the problem is trivially solvable on trees; here n is the number of

nodes in the network. We give a simple deterministic combinatorial $O(\log P)$ approximation, where P is the number of distinct packets to be routed. Note that if two or more packets are essentially identical in that they are desired by exactly the same set of clients, then we can combine them into a single packet (albeit with a larger size). Then, under the laminar demands assumption, P is always at most the number of distinct demand sets or clients, which is at most n, the total number of nodes in the network. In fact in applications such as the Netflix multicast problem described above, we expect n to be much larger than P.

Furthermore, our $O(\log P)$ approximation algorithm is a natural combinatorial algorithm that is simple and fast to implement. It is convenient to represent the laminar family of demand sets in the form of a tree where the demand set at any node is a proper subset of that at its parent and disjoint from those at its siblings. Our algorithm begins with some minor preprocessing of the demand tree to ensure that the tree has small (logarithmic) height. It then traverses the demand tree in a top-down fashion, and at each node of the tree constructs a (approximately optimal) Steiner tree over all the terminals with the corresponding demand set connecting them to the source. We show that the cost of the Steiner trees constructed at every level of the demand tree is bounded by a constant times the cost of the optimal solution, and therefore obtain an overall approximation factor proportional to the height of the tree. The height of the tree can however be much larger than $\log P$ because packets have different sizes. In order to prove an approximation factor of $O(\log P)$ we need to do a more careful bounding of the total cost spent on long chains of degree 2 in the demand tree. Using the Steiner tree algorithm of Robins and Zelikovsky [RZ00] as a subroutine we obtain a $(6.2 \log P)$-approximation.

2.1.2 Related Work

RAND is closely related to single-source uniform buy-at-bulk network design (BaBND) [AA97, GKR03, MMP08] in that both problems involve a trade-off between picking short paths between the source and the clients and trying to merge different paths to avail of volume discounts. However the actual cost structure of the two problems is very different. In BaBND the cost on an edge is a concave function of the total load on the edge; in our setting the cost is a submodular function of the clients using that edge. Neither of the problems is a special case of the other. BaBND admits a constant approximation in the single-source version [GKR03, Tal02] and an $O(\log n)$ approximation is known for the general multi-source multi-sink problem [AA97].

One way of thinking about RAND is to break-up the problem and solution packet-wise: each packet p defines a subset of the terminals, say T_p, that desire that packet; the solution restricted to these terminals is essentially solving a Steiner tree problem over the set $T_p \cup \{s\}$. Our goal is to pick a single collection of paths such that the sum over packets of the costs of these Steiner trees is minimized. In this respect, the problem is related to variants of the Steiner tree problem where the set of terminals is not precisely known before hand. This includes the maybecast problem of Karger and Minkoff [KM00] for which a constant factor approximation is known, as well as the universal Steiner tree problem [JLN$^+$05] for which a randomized logarithmic approximation can again be obtained through tree embeddings and this is the best possible [BCK10].

Facility location has been extensively studied under various models. The models most closely related to RAFL are the service installation costs model of Shmoys et al. [SSL04] and the heirarchical costs model of Svitkina and Tardos [ST06]. In the former, each client has a production cost associated with it; the cost of opening a facility is equal to a fixed cost associated with the facility plus the production costs of all the clients assigned to that

facility. One way of representing these costs is in the form of a two-level tree for each facility with the fixed cost for the facility at the root of the tree and the client-specific production costs at the next level nodes. The cost of assigning a set of clients to the facility is the total cost of the subtree formed by the unique paths connecting the client nodes to the root of the tree. Svitkina and Tardos generalize this cost model to a tree of arbitrary depth, although in their setting the trees for different facilities are identical. Both the works present constant factor approximations for the respective versions, based on a primal-dual approach and local search respectively. Our model is similar to these models in that our facility costs are also submodular in the set of clients connecting to a facility. In our setting, the costs can again be modeled by a tree in which each node is associated with one or more clients; the cost of a collection of clients is given by the total cost of the union of subtrees rooted at those clients (as opposed to the portion of the tree "above" the clients). Therefore, neither of the two settings generalize the other. Moreover, facility costs in our setting are different for different facilities, although they are related through a multiplier per facility. Finally, while in Shmoys et al. and Svitkina et al. routing costs are given merely by the metric over facilities and clients, in our setting they are given by the distances times the total demand routed.

For modeling information flow, Hayrapetyan et al. [HST05] have studied a single-source network design problem with monotone submodular costs on edges, and a group facility location problem. The network design setting generalizes ours in that edge costs can be arbitrary submodular functions of the clients using the edges; they note that an $O(\log n)$ approximation can be achieved via tree embeddings. In contrast, we obtain an $O(\log P)$ approximation, where P, the number of distinct packets in the system, is always at most n and generally much smaller. In the group facility location problem, edge costs are identical to those in RAND, but neither of the problems subsumes the other: the former assumes there are multiple facilities (sources) with fixed opening costs, but also limits the number of

distinct packets per client to at most one.

In Section 2.2 we formally define the problems and develop notation. We describe our logarithmic-approximation algorithm for RAND in Section 2.4 and our constant-factor approximation for RAFL in Section 2.3. Finally, we conclude with some open questions in Section 2.5.

2.2 Problem Definition and Preliminaries

The traffic-redundancy aware network design (RAND) problem is defined as follows. We are given a graph $G = (V, E)$ with weights $c_e \in \Re^+$ on edges $e \in E$, and a special node s called the source. In addition, we are given a set T of clients or terminals located at different nodes in the graph. The source carries a set Π of packets with $|\Pi| = P$. Each packet $p \in \Pi$ is associated with a weight w_p; we assume that the weights are integral. Each terminal $t \in T$ desires some subset of the packets; this is called the terminal's demand and is denoted by $\mathrm{D}(t)$. We use the convention $\mathrm{D}(s) = \Pi$. Also let $w(S) = \sum_{p \in S} w_p$ denote the total weight of a set S of packets.

We assume that the collection of demand sets $\mathcal{D} = \{\mathrm{D}(t)\}_{t \in T}$ forms a laminar family of sets. In particular, for any two terminals $t_1, t_2 \in T$, $\mathrm{D}(t_1) \cap \mathrm{D}(t_2) \neq \emptyset$ implies that either $\mathrm{D}(t_1) \subseteq \mathrm{D}(t_2)$ or $\mathrm{D}(t_2) \subseteq \mathrm{D}(t_1)$. We use a tree τ to represent the containment relationship between sets in the laminar family. The nodes of τ are sets in $\mathcal{D}$. A demand set X is a parent of another set Y if $Y \subset X$ and there is no set $Z \in \mathcal{D}$ with $Y \subset Z \subset X$. At the root of the tree is the universe Π of packets. For a demand set X in the tree τ, we use T_X to denote the terminals $t \in T$ with $\mathrm{D}(t) = X$.

We denote an instance of RAND by the tuple $(G, T, \mathcal{D}, \tau)$.

The solution to RAND is a collection of paths $\mathcal{P} = \{P_t\}_{t \in T}$, with P_t connecting the terminal t to the source s. Given this solution, an edge $e \in E$ carries the set $S_{\mathcal{P}}(e) =$

$\cup_{t \in T: P_t \ni e} \mathrm{D}(t)$ of packets. The load on the edge is $w_{\mathcal{P}}(e) = w(S_{\mathcal{P}}(e))$. We drop the subscript $\mathcal{P}$ when it is clear from the context. The cost of the solution $\mathcal{P}$ is $\mathrm{cost}(\mathcal{P}) = \sum_{e \in E} c_e w_{\mathcal{P}}(e)$. Our goal is to find a solution of minimum cost.

Let $\mathrm{OPT} = \mathrm{argmin}_{\text{feasible } \mathcal{P}} \mathrm{cost}(\mathcal{P})$ be an optimal solution. For a subset W of terminals, we use $\mathrm{OPT}(W)$ to denote the restriction of OPT to W, that is, the collection of paths $\{P_t \in \mathrm{OPT}\}_{t \in W}$.

In the traffic-redundancy aware facility location problem (RAFL), we are given a set $\mathcal{F}$ of facilities, and a graph over $T \cup \mathcal{F}$ with edge weights c_e. Let $c(u, v)$ denote the shortest path distance between nodes u and v in the graph under the metric c. Furthermore, each facility $f \in \mathcal{F}$ has a cost λ_f associated with it.

A solution to this problem is an assignment A from terminals to facilities. The assignment specifies the set of packets that a facility f needs to produce in order to serve all the terminals connected to it: $S_A(f) = \cup_{t \in T: A(t) = f} \mathrm{D}(t)$. The load on the facility is $w_A(f) = w(S_A(f))$. We drop the subscript A when it is clear from the context. The cost of the solution A is $\mathrm{cost}(A) = \sum_{f \in \mathcal{F}} \lambda_f w_A(f) + \sum_{t \in T} w(\mathrm{D}(t)) c(t, A(t))$. The first component of the cost is called the facility opening cost $C_f(A)$, and the second the routing cost $C_r(A)$ of the solution. Once again our goal is to find a solution of minimum cost.

Both RAND and RAFL are NP-hard because they generalize Steiner tree and metric uncapacitated facility location respectively. Our goal is to find approximation algorithms.

2.3 A Constant Factor Approximation for RAFL

In this section we present a constant factor approximation for the RAFL. For ease of exposition we assume that all packets have unit weight, and write $|S|$ for the total size or weight of a set S of packets. This assumption is without loss of generality. We further assume without loss of generality that $\min_{f \in \mathcal{F}} \lambda_f = 1$.

The following is a natural LP-relaxation of RAFL. Here $x_{t,f}$ is an indicator for whether terminal t is assigned to facility f, and $y_{f,p}$ denotes the extent to which facility f produces packet p.

$$\begin{aligned} \text{minimize} \quad & \sum_{f \in \mathcal{F}} \sum_{p \in \Pi} \lambda_f y_{f,p} + \sum_{t \in T} \sum_{f \in \mathcal{F}} |\mathrm{D}(t)| x_{t,f} c(t,f) & \\ \text{subject to} \quad & \sum_{f \in \mathcal{F}} x_{t,f} \geq 1 & \forall t \in T \\ & y_{f,p} \geq x_{t,f} & \forall t, f, p \in \mathrm{D}(t) \end{aligned}$$

Our approach begins along the lines of the filtering approach developed by Lin and Vitter [LV92] and employs some of the rounding ideas of Shmoys et al. [STA97] developed for metric uncapacitated facility location. In particular, given an optimal solution to the LP, we preprocess the solution at a constant factor loss in performance such that each terminal is assigned to a non-zero extent only to facilities for which the terminal's routing cost is within a small constant factor of the corresponding average amount in the LP. At this point, each terminal can be assigned to any facility to which it is fractionally assigned by the filtered solution, at a low routing cost. The key part of the analysis is bounding the cost for producing packets at facilities.

Let $\mathcal{F}(t)$ denote the set of facilities to which t is assigned fractionally by the filtered solution. In Shmoys et al.'s setting, in order to bound the cost of opening facilities, it is sufficient to find a "paying" terminal for each open facility such that the sets $\mathcal{F}(t)$ for paying terminals are mutually disjoint. For each terminal t that is not paying, there exists at least one representative paying terminal t' such that $\mathcal{F}(t)$ and $\mathcal{F}(t')$ overlap; any such terminal t is assigned to the facility opened by its representative terminal t' at a slight increase in routing cost as long as the average routing cost of t' is no more than that of t. In order to

accomplish this, we process terminals in order of increasing average routing cost.

In our setting, this approach has a basic flaw. The terminal t' may have a much smaller demand set compared to the terminal t. Then, if we assign t to the facility opened by t', the facility needs to produce many more packets and its new larger opening cost can no longer be charged to t'. Unfortunately, it is not possible to ensure that t' has both a small average routing cost than t as well as a larger demand set. Instead, we divide the process of opening facilities into two parts. First we determine which terminals are paying and which ones are not by processing facilities in order of increasing average routing cost. Then we decide which facilities to open by processing the non-paying terminals in the order of decreasing demand set sizes.

The algorithm is described formally below. We first introduce some notation. Let (x^*, y^*) denote the optimal solution to the RAFL LP given above. Let $C_r^*(t) = \sum_{f \in \mathcal{F}} x_{t,f}^* c(t, f)$ and $C_f^*(t) = \sum_{f \in \mathcal{F}} x_{t,f}^* \lambda_f$ denote the average routing and facility opening costs respectively under the solution (x^*, y^*) associated with a terminal t. The total routing and facility opening costs of the solution are given by $C_r^* = \sum_{t \in T} |\mathrm{D}(t)| C_r^*(t)$ and $C_f^* = \sum_{f \in \mathcal{F}} \sum_{p \in \Pi} y_{f,p}^* \lambda_f$ respectively. Likewise, for a feasible solution (x, y), we use $C_r^{(x,y)}(t)$ and $C_f^{(x,y)}(t)$ to denote the average routing and facility opening costs associated with a terminal t respectively. We drop the superscript (x, y) when it is clear from context. Also let $C_r(x, y)$ and $C_f(x, y)$ denote the total routing and facility opening costs of the solution (x, y).

Our algorithm proceeds in three stages. The first is a filtering stage in which we convert the solution (x^*, y^*) into a fractional solution (x, y) which satisfies the following property: for all t, f with $x_{t,f} > 0$, $c(t, f) \leq \alpha C_r^*(t)$. Here α is a parameter that we fix later. For a terminal t, we use $\mathcal{F}(t)$ to denote all the facilities f that are fractionally assigned to t in (x, y), that is, have $x_{t,f} > 0$.

In the second stage of the algorithm, we classify terminals into paying terminals T^p and free terminals T^f. Essentially, a terminal t becomes a free terminal if any of the facilities in

Algorithm 1 Rounding algorithm for RAFL

Given: LP solution (x^*, y^*); Return: Assignment A from terminals to facilities

Phase 1: Filtering

for all $t \in T$
  Let $C_r^*(t) = \sum_{f\in\mathcal{F}} x_{t,f}^* c(t,f)$ and $C_f^*(t) = \sum_{f\in\mathcal{F}} x_{t,f}^* \lambda_f$
  For all $f \in \mathcal{F}$, if $c(t,f) > \alpha C_r^*(t)$ set $x_{t,f} = 0$ else $x_{t,f} = x_{t,f}^*$.
  Renormalize $x_{t,f}$ so that $\sum_f x_{t,f} = 1$.
  Let $\mathcal{F}(t) = \{f : x_{t,f} > 0\}$, $C_f(t) = \sum_{f\in\mathcal{F}} x_{t,f}\lambda_f$, and $C_r(t) = \sum_{f\in\mathcal{F}} x_{t,f} c(t,f)$.
For all $f \in \mathcal{F}$ and $p \in \Pi$, set $y_{f,p} = \max_{t\in T:\mathrm{D}(t)\ni p} x_{t,f}$.

Phase 2: Classification of terminals into paying and free

Initialize paying and free terminal sets: $T^p = \emptyset$ and $T^f = \emptyset$.
For all $t \in T$ initialize temporary assignment $\tilde{A}(t) = \emptyset$, permanent assignment $A(t) = \emptyset$, and cover $\mathrm{Cov}(t) = \emptyset$.
For all $f \in \mathcal{F}$ initialize paying terminal set $\mathrm{Pay}(f) = \emptyset$ and final paying set $\mathrm{FPay}(f) = \emptyset$.
case $T \setminus \left(T^p \cup T^f\right) \neq \emptyset$
  Let $t = \mathrm{argmin}_{j\in T\setminus(T^p\cup T^f)} C_r^*(j)$. {Select terminal with least connection cost}
  if there exists $f \in \mathcal{F}(t)$ such that $\mathrm{Pay}(f)$ covers t **then**
    $T^f = T^f \cup \{t\}$ {t is a free terminal}
    $\tilde{A}(t) = f$
    Assign covering set for t: $\mathrm{Cov}(t) = \{j \in \mathrm{Pay}(f) \mid \mathrm{D}(j) \cap \mathrm{D}(t) \neq \emptyset\}$
  else
    $T^p = T^p \cup \{t\}$ {t is a paying terminal}
    For all $f \in \mathcal{F}(t)$ update $\mathrm{Pay}(f) = \mathrm{Pay}(f) \cup \{t\}$

Phase 3: Opening facilities

For all $t \in T^p$ assign level $\ell(t) = \lceil \log_2 C_f(t) \rceil$; for all $t \in T^f$ assign $\ell(t) = \min_{j\in\mathrm{Cov}(t)} \ell(j)$.

Initialize $T_d^f = \{t \in T^f \mid \ell(t) = d\}$ and for all $t \in T_d^f$ set $\Gamma_d(t) = \{t' \in T_d^f \mid \mathrm{Cov}(t) \cap \mathrm{Cov}(t') \neq \emptyset\}$.
for $d = 0$ to $\max_{j\in T^p} \ell(j)$
  Initialize $W = T_d^f$ {Repeat till all terminals in T_d^f have been assigned a permanent facility}.
  case $W \neq \emptyset$
    Let $t \in \mathrm{argmax}_{j\in W} |\mathrm{D}(j)|$ {Select any terminal with the largest demand set}.
    Let $\bar{t} \in \mathrm{argmin}_{j\in\Gamma_d(t)} C_r^*(j)$.
    Let facility $\phi(\bar{t}) \in \mathrm{argmin}_{f\in\cup_{j\in\mathrm{Cov}(\bar{t})}\mathcal{F}(j)} \lambda_f$.
    Let $A(t) = \phi(\bar{t})$ {We say that t opens the facility $\phi(\bar{t})$.}
    For all $t' \in \Gamma_d(t)$, assign $A(t') = A(t)$. Update $W = W \setminus (\Gamma_d(t) + \{t\})$.
    Assign final paying set for facility $A(t)$: $\mathrm{FPay}(A(t)) = \mathrm{Cov}(t)$.
for all $t \in T^p$
  Let $A(t) \in \mathrm{argmin}_{f\in\mathcal{F}(t)} \lambda_f$.

$\mathcal{F}(t)$ is already expected to produce a large fraction of t's demand. We record this facility as t's temporary assignment $\tilde{A}(t)$. To this end, we say that a set of terminals W covers a terminal t if $|\mathrm{D}(t) \setminus (\cup_{t' \in W} \mathrm{D}(t'))| < \frac{1}{2}|\mathrm{D}(t)|$. If a terminal t is not covered at any of the facilities in $\mathcal{F}(t)$, then it becomes a paying terminal and can potentially pay for any of the facilities in $\mathcal{F}(t)$. $\mathrm{Pay}(f)$ tracks the set of terminals paying for a facility f.

Finally, in the third stage of the algorithm, we pick a permanent assignment from terminals to facilities by considering facilities in decreasing order of the sizes of their demand sets. As a first cut approach, suppose that we assign a free terminal t to the facility at which it is covered ($\tilde{A}(t)$), and pay for that facility using the paying terminals associated with it. To ensure that no paying terminal t' ends up paying for two or more opened facilities, we consider all the free terminals that this paying terminal covers and assign those also to the first facility that the paying terminal pays for. The order in which we assign free terminals to facilities ensures that in this last step we do not increase the facility opening cost of the solution. Here is the catch: which facility is actually opened is decided by the free terminal t that starts this process and may be one of the more expensive facilities in the paying terminal t''s set $\mathcal{F}(t')$. In this case, the paying terminal does not have enough charge in the LP solution to pay for this facility. In order to avoid this situation, we consider all of the facilities that are "close" to t or to other free terminals covered by the paying terminals that cover t. Of these we open the facility with minimum cost and pay for it using the paying terminals associated with f.

While in our algorithm a paying terminal can end up paying for multiple opened facilities, in Lemma 2.4 below we argue that the costs of those facilities decrease geometrically and so the sum can be bounded.

We now formalize this argument. We begin by showing that the fractional solution (x, y) is not too expensive.

Lemma 2.1. *The solution (x, y) is feasible for the RAFL LP. Moreover $C_f(x, y) \leq \frac{\alpha}{\alpha-1} C^*_f$.*

Proof. (x, y) is feasible by construction. Note that for all $t \in T$, $\sum_f x^*_{f,t} = 1$, otherwise the cost of the solution can be improved. Therefore, by Markov's inequality, $\sum_{f:c(f,t)>\alpha C^*_r} x^*_{f,t} \leq 1/\alpha$. Then, in the renormalization step (Step 4) we set $x_{t,f}$ to be no more than $\alpha/(\alpha-1)x^*_{t,f}$.

For all $f \in F$ and $p \in \Pi$ we set $y_{f,p}$ to be $\max_{t\in T:\mathrm{D}(t)\ni p} x_{t,f}$. Hence $y_{f,p}$ is no more than $\max_{t\in T:\mathrm{D}(t)\ni p} \frac{\alpha}{\alpha-1} x^*_{t,f}$, which is no more than $\alpha/(\alpha-1)y^*_{f,p}$. This in turn implies the second part of the claim. □

Next we bound the routing cost of the assignment A.

Lemma 2.2. *For all $i \in T$, $c(i, A(i)) \leq 9\alpha C^*_r(i)$.*

Proof. Let i be a paying terminal that is assigned a facility f in Step 31 of the algorithm. Then, $f \in \mathcal{F}(i)$ and therefore, by the definition of x and $\mathcal{F}(i)$, $c(i, f) \leq \alpha C^*_r(i)$.

Now, let i be a free terminal which is assigned a facility f in Step 27 or Step 28 of the algorithm. Say level $\ell(i) = d$ and let t be the terminal that opened f. Then there are two possibilities: either $i = t$ or $i \in \Gamma_d(t)$. In the former case, $\tilde{A}(t) \in \mathcal{F}(i)$ and so $c(i, \tilde{A}(t)) \leq \alpha C^*_r(i)$.

Next we show that if terminal t opens a facility (in Step 27) then for all $t' \in \Gamma_d(t)$ we have $c(t', \tilde{A}(t)) \leq 3\alpha C^*_r(t')$. Note that $\mathrm{Cov}(t') \cap \mathrm{Cov}(t) \neq \emptyset$ and let j be a paying terminal in $\mathrm{Cov}(t') \cap \mathrm{Cov}(t)$. We have $\tilde{A}(t) \in \mathcal{F}(j)$ and $\tilde{A}(t') \in \mathcal{F}(j)$. Since terminal j was selected in the second phase before t' we have $C^*_r(j) \leq C^*_r(t')$. By triangle inequality we get that $c(t', \tilde{A}(t)) \leq c(t', \tilde{A}(t')) + c(\tilde{A}(t'), j) + c(j, \tilde{A}(t)) \leq \alpha C^*_r(t') + 2\alpha C^*_r(j) \leq 3\alpha C^*_r(t')$.

In Step 27 the opened facility, say f, is selected to be $\phi(\bar{t})$ for the terminal $\bar{t} \in \Gamma_d(t)$ with minimum C^*_r value. Since $f \in \mathcal{F}(j')$ for some $j' \in \mathrm{Cov}(\bar{t})$ we have $c(f, \bar{t}) \leq 3\alpha C^*_r(\bar{t})$. Also $\bar{t}$ is contained in $\Gamma_d(t)$, which implies that $c(\bar{t}, \tilde{A}(t)) \leq 3\alpha C^*_r(\bar{t})$.

Again using triangle inequality we get the desired result: $c(i,f) \leq c(i,\tilde{A}(t)) + c(\tilde{A}(t),\bar{t}) + c(\bar{t},f) \leq 3\alpha C_r^*(i) + 6\alpha C_r^*(\bar{t}) \leq 9\alpha C_r^*(i)$. Here the last inequality follows from the definition of $\bar{t}$. □

Finally we account for the facility opening cost of the solution. Recall that $S_A(f)$ denotes the set of packets produced at f under the assignment A. We note that a facility f may be "opened" multiple times by different free terminals in Step 27 of the algorithm. In this case, we treat each subsequent opening as opening a new copy of f and designate a distinct set of terminals, $\mathrm{FPay}(f)$, to pay for all of the packets to be produced at the new copy freshly (even though some of them may already be assigned to the facility).

Henceforth, for ease of exposition we will assume that each facility is opened at most once.

The following lemma notes that $|S_A(f)|$ can be bounded in terms of the demand sets of the terminals finally paying for this facility.

Lemma 2.3. *Let f be a facility opened by a free terminal t. Then $S_A(f) = \mathrm{D}(t) \cup \cup_{j\in \mathrm{Cov}(t)} \mathrm{D}(j)$. Furthermore, $|S_A(f)| \leq 2|\cup_{j\in \mathrm{Cov}(t)} \mathrm{D}(j)|$.*

Proof. Let $t \in T_d^f$ and consider a terminal $t' \in \Gamma_d(t)$. We claim that $\mathrm{D}(t') \subseteq \mathrm{D}(t) \cup \cup_{j\in \mathrm{Cov}(t)} \mathrm{D}(j)$, and this implies the first part of the lemma. Let $j \in \mathrm{Cov}(t) \cap \mathrm{Cov}(t')$. Then, by the definition of a covering set, $\mathrm{D}(t') \cap \mathrm{D}(j) \neq \emptyset$. By laminarity, either $\mathrm{D}(j) \supset \mathrm{D}(t')$ in which case our claim holds, or $\mathrm{D}(j) \subset \mathrm{D}(t')$. In the latter case, $\mathrm{D}(t) \cap \mathrm{D}(j) \neq \emptyset$ implies $\mathrm{D}(t) \cap \mathrm{D}(t') \neq \emptyset$. Once again by laminarity, either $\mathrm{D}(t') \subseteq \mathrm{D}(t)$, or $\mathrm{D}(t') \supset \mathrm{D}(t)$. The latter case cannot hold because t is considered before t' in phase 3 and therefore $|\mathrm{D}(t')| \leq |\mathrm{D}(t)|$. In the former case our claim holds.

The second part of the lemma follows from the definition of covering. □

Using this lemma we can bound the facility opening cost of the assignment A in terms of the average facility opening costs $C_f(t)$ for the paying terminals $t \in T^p$ in the fractional solution (x, y).

Lemma 2.4. $\sum_f |S_A(f)|\lambda_f \leq 9 \sum_{t \in T^p} |\mathrm{D}(t)| C_f(t)$.

Proof. First we bound the opening cost of facilities that were opened by free terminals. Lemma 2.3 implies that $|S_A(f)| \leq 2|\cup_{j \in \mathrm{FPay}(f)} \mathrm{D}(j)| \leq 2 \sum_{j \in \mathrm{FPay}(f)} |\mathrm{D}(j)|$. For facility f write $\ell(f) = d$ iff the terminal opening it has level d. Note that for facility f with $\ell(f) = d$ we have $\lambda_f \leq 2^d$.

Fix a terminal $i \in T^p$, and write $\mathrm{Cov}^{-1}(i) = \{j \in T^f \mid i \in \mathrm{Cov}(j)\}$. Say $\ell(i) = \ell$. Then $C_f(i) \in (2^{\ell-1}, 2^\ell]$. By definition, for all $j \in \mathrm{Cov}^{-1}(i)$ we have $\ell(j) \leq \ell$.

We claim that i pays for at most one facility at level d for $d \leq \ell$, and does not pay for any facilities at level $d > \ell$. We prove the first part by contradiction. Say there exits $f \neq f'$ such that $\ell(f) = \ell(f') = d$ and $i \in \mathrm{FPay}(f) \cap \mathrm{FPay}(f')$. Write t as the terminal that opens f and t' as the terminal that opens f'. Both t and t' are in $\mathrm{Cov}^{-1}(i)$ and have level equal to d. Without loss of generality assume t was processed before t' by the algorithm. Then $t' \in \Gamma_d(t)$ and we would have $A(t') = A(t)$, contradicting the assumption that they are assigned to different facilities.

For the second part of the claim, suppose that $i \in \mathrm{FPay}(f)$ for some facility f. Then f is opened by a terminal $t \in \mathrm{Cov}^{-1}(i)$. As stated above, the level of such a terminal t must be no more than the level of i, hence $\ell(f) \leq \ell(i)$.

For a level d facility f we have $\lambda_f \leq 2^d$ and hence $\sum_{f:\mathrm{FPay}(f) \ni i} \lambda_f \leq \sum_{d=1}^{\ell} 2^d \leq 4C_f(i)$.

We therefore get the following chain of inequalities.

$$\begin{aligned}\sum_f \lambda_f |S_A(f)| &\leq 2 \sum_f \lambda_f \sum_{j \in \mathrm{FPay}(f)} |\mathrm{D}(j)| \\ &= 2 \sum_{j \in T^p} |\mathrm{D}(j)| \sum_{f:\ \mathrm{FPay}(f) \ni j} \lambda_f \\ &\leq 8 \sum_{j \in T^p} C_f(j)\ |\mathrm{D}(j)|\end{aligned}$$

Here the first inequality follows from Lemma 2.3 and the second inequality follows from the bound $\sum_{f:\mathrm{FPay}(f)\ni i} \lambda_f \leq 4C_f(i)$.

To account for facilities that were opened by paying terminals in Step 31 of the algorithm we note that for any such facility h, the set $S_A(h) = \mathrm{D}(t)$ where t is the paying terminal that opened h. Moreover $\lambda_h \leq \sum_f x_{t,f}\lambda_f$ and hence the opening cost incurred by the algorithm is no more than the fractional value, $\lambda_h|\mathrm{D}(t)| \leq C_f(t)\ |\mathrm{D}(t)|$. Hence the total facility opening cost incurred by the algorithm is no more than $9 \sum_{t \in T^p} |\mathrm{D}(t)| C_f(t)$. □

To complete the argument, we relate the costs $C_f(t)$ to the total cost $C_f(x, y)$.

Lemma 2.5. $\sum_{t \in T^p} |\mathrm{D}(t)| C_f(t) \leq 2C_f(x, y)$.

Proof. When a terminal t is added to T^p (Step 17 of the algorithm) it is not covered at any of the facilities in $\mathcal{F}(t)$. For $f \in \mathcal{F}$ let $L(t, f) = \mathrm{D}(t) \setminus \cup_{t' \in \mathrm{Pay}(f)} \mathrm{D}(t')$ denote the set of packets in $\mathrm{D}(t)$ that is not covered at f at the time that t is considered. Here, $\mathrm{Pay}(f)$ denotes the set of terminals that is paying for f at the time that t is considered. Note that $|L(t, f)| \geq 1/2|\mathrm{D}(t)|$ for $f \in \mathcal{F}(t)$. For any facility f the sets $L(t, f)$ are disjoint and partition the support of $y_{f,p}$ and hence the following chain of inequalities hold:

$$\begin{aligned}\sum_p y_{f,p} &\geq \sum_{t\in T^p} \sum_{p\in L(t,f)} y_{f,p} \\ &\geq \sum_{t\in T^p} \sum_{p\in L(t,f)} x_{t,f} \\ &= \sum_{t\in T^p} |L(t,f)| x_{t,f} \end{aligned} \tag{2.1}$$

We can now derive the desired bound:

$$\begin{aligned} C_f(x,y) &= \sum_f \sum_p \lambda_f y_{f,p} \\ &= \sum_f \lambda_f \sum_p y_{f,p} \\ &\geq \sum_f \lambda_f \sum_{t\in T^p} |L(t,f)|\ x_{t,f} \\ &\geq \sum_f \sum_{t\in T^p} \frac{1}{2}\ |\mathrm{D}(t)|\ \lambda_f x_{t,f} \\ &= \sum_{t\in T^p} \frac{1}{2}\ |\mathrm{D}(t)| \sum_f \lambda_f x_{t,f} \\ &= \frac{1}{2} \sum_{t\in T^p}\ |\mathrm{D}(t)| C_f^{(x,y)}(t) \end{aligned}$$

Here the third step follows from inequality (2.1) and the fourth step holds because for any $t \in T^p$ and $f \in \mathcal{F}$, either $x_{t,f} = 0$ or $f \in \mathcal{F}(t)$ and the size of the set $L(t,f)$ is at least half its demand: $|L(t,f)| \geq 1/2|\mathrm{D}(t)|$. □

Putting together the above lemmas we obtain the following theorem:

Theorem 2.1. *Algorithm 1 gives a* 27*-approximation to the RAFL.*

Proof. Lemma 2.2 implies that for any terminal the routing cost is no more than 9α times the optimal. Also from Lemma 2.4 and Lemma 2.5 we get that the total facility opening

cost is no more than 18 times the total facility opening cost of the filtered solution $C_f(x, y)$. Finally from Lemma 2.1 we have $C_f(x, y) \leq \alpha/(\alpha - 1)\ C_f^*$, so the facility opening cost of the generated solution is no more than $18\alpha/(\alpha - 1)$ times the optimal. Hence the algorithm achieves an approximation factor of $\max\{9\alpha, \frac{18\alpha}{\alpha-1}\}$, which is minimized at $\alpha = 3$ to give us a 27-approximation. □

2.4 An $O(\log P)$ Approximation for RAND

In this section we develop an $O(\log P)$-approximation algorithm for the RAND where $P = |\Pi|$. The basic observation that our algorithm hinges on is that if every pair of demand sets in $\mathcal{D}$ is either identical or disjoint, that is, the tree τ is a two-level tree, then the problem becomes easy and can be approximated to within a small constant factor. In particular, then the problem becomes one of finding optimal Steiner trees connecting the terminals in T_X to s for every set $X \in \mathcal{D}$. The cost of these Steiner trees is purely additive because the different sets in $\mathcal{D}$ are disjoint.

In particular, this implies that for any collection $\mathcal{X}$ containing disjoint sets of packets, we can construct a partial solution over terminals in $\cup_{X\in\mathcal{X}} T_X$ at a cost of constant times the cost of the optimal solution. This immediately suggests an algorithm with approximation ratio a constant times the depth of the tree τ: define the level of a node in τ as its distance from the root Π; for every level k, consider the collection $\mathcal{X}_k$ of sets at level k and construct a constant factor approximation over terminals in $\cup_{X\in\mathcal{X}_k} T_X$.

In order to obtain a small approximation ratio through this approach, our algorithm first performs a preprocessing of the demand tree τ to ensure, at small cost, that for any pair of demand sets in the tree where one is a parent of the other, the total weight of the parent is at least twice as large as the total weight of the child. Since the weight of every packet is integral, this implies that the depth of the preprocessed tree is at most $\log w(\Pi)$,

and gives us an $O(\log w(\Pi))$ approximation.

To obtain an $O(\log P)$ approximation, we need to do a more clever analysis. As discussed in the introduction, P denotes the number of effectively distinct packets in the instance. In particular, we can assume without loss of generality that P is equal to the number of nodes in the tree τ. Our next key observation is that we can collectively bound the total cost of Steiner trees for nodes in a long root to leaf chain of nodes by a constant times the cost of the optimal solution (rather than by the length of the chain times the optimal cost). Here we crucially use the fact that each subsequent node in the chain has a weight at most half that of the preceding node, a fact that is ensured through our preprocessing step. Given this, we break up the demand tree into $\log P$ collections of chains, each of which corresponds to disjoint packet sets. The total cost of Steiner trees over each collection of chains can then be bounded to within a constant factor of the optimal cost, and we get an overall $O(\log P)$ approximation.

We now present our algorithm and analysis formally.

Preprocessing the tree τ. Our preprocessing phase is described in Algorithm 2 below. The preprocessing phase makes two changes to the given instance. First, it changes the demand sets of some terminals to supersets of their original demands. Second, after these changes to demand sets, if there are nodes in τ that do not have any terminals associated with them, it merges these nodes with their parents.

Algorithm 2 Preprocessing algorithm

Given: Instance $(G, T, \mathcal{D}, \tau)$; Return: New instance $(G, T, \mathcal{D}', \tau')$

1: Perform depth first search over the tree τ.
2: **for all** nodes $X \in \mathcal{D}$ encountered during DFS
3: Let Y be the parent of X in τ
4: **if** $w(X) > \frac{1}{2}w(Y)$ **then**
5: For all $t \in T_X$, set $\mathrm{D}'(t) = Y$.
6: Merge X with Y. That is, set $T_Y = T_Y \cup T_X$, remove X from τ, and reattach the children of X in τ as the children of Y in the modified tree.

We obtain the following lemma.

Lemma 2.6. *Consider an instance $(G, T, \mathcal{D}, \tau)$ of RAND. Then Algorithm 2 returns an instance $(G, T, \mathcal{D}', \tau')$ with the following properties:*

1. *For every $t \in T$, $\mathrm{D}'(t) \supseteq \mathrm{D}(t)$ and $w(\mathrm{D}'(t)) \leq 2w(\mathrm{D}(t))$.*
2. *$\mathcal{D}'$ is a laminar family.*
3. *For every pair of sets $X, Y \in \mathcal{D}'$ such that X is a child of Y in τ', $w(Y) \geq 2w(X)$.*
4. *The cost of the optimal solution over $(G, T, \mathcal{D}', \tau')$ is at most twice that of the optimal solution over $(G, T, \mathcal{D}, \tau)$.*
5. *Any feasible solution to $(G, T, \mathcal{D}', \tau')$ is also feasible for $(G, T, \mathcal{D}, \tau)$.*

Proof. We first remark that the demand of any terminal t is modified at most once, when its original node $\mathrm{D}(t)$ is encountered in the DFS over τ; after this, the new node $\mathrm{D}'(t)$ is never processed again by DFS. Then, it is easy to see that the first property holds. The second property holds because τ' is a tree over sets in $\mathcal{D}'$ and therefore any two sets in $\mathcal{D}'$ are either disjoint or one is an ancestor of another. The third property holds by construction: when the node X is encountered by the DFS, if the node survives the preprocessing, then it holds that $w(X) \leq \frac{1}{2}w(Y)$. The fifth property follows immediately from the first.

To prove the fourth property, we note that for any two terminals t_1 and t_2, $\mathrm{D}(t_1) \subset \mathrm{D}(t_2)$ implies $\mathrm{D}'(t_1) \subseteq \mathrm{D}'(t_2)$. Now consider any optimal solution $\mathcal{P} = \{P_t\}_{t \in T}$ to $(G, T, \mathcal{D}, \tau)$. We will show that the cost of the solution $\mathcal{P}$ for the new instance $(G, T, \mathcal{D}', \tau')$ is no more than twice its cost for $(G, T, \mathcal{D}, \tau)$. For every edge $e \in E$ consider the set of terminals, say $T(e)$, that use e: $T(e) = \{t : P_t \ni e\}$. Let $T'(e)$ be the subset of $T(e)$ of terminals whose demand is not contained inside the demand of any other terminals in $T(e)$; that is $T'(e)$ denotes the terminals with the maximal demand sets. Then, in the instance

$(G, T, \mathcal{D}, \tau)$, $w(S_{\mathcal{P}}(e)) = \sum_{t \in T'(e)} w(\mathrm{D}(t))$. Moreover by our observation above, in the new instance $(G, T, \mathcal{D}', \tau')$, terminals in $T'(e)$ are still the maximal demand terminals and $w(S_{\mathcal{P}}(e)) = \sum_{t \in T'(e)} w(\mathrm{D}'(t))$. The claim now follows from the first property. □

Main algorithm.

We now proceed to describe the main algorithm. Henceforth we will assume that the given instance of RAND satisfies the properties listed in Lemma 2.6, in particular, property 3.

Algorithm 3 A logarithmic approximation for RAND

Given: Instance $(G, T, \mathcal{D}, \tau)$ satisfying the properties in Lemma 2.6; Return: Collection of paths from each terminal to the source s.

1: Perform depth first search over the tree τ.
2: **for all** nodes $X \in \mathcal{D}$ encountered during DFS
3: Construct an approximately optimal Steiner tree $\mathcal{S}(X)$ in G over $T_X \cup \{s\}$.
4: For every $t \in T_X$, return the unique path in $\mathcal{S}(X)$ from t to s.

We first define some notation. For a demand set Y, let $\mathcal{S}^*(Y)$ denote the optimal Steiner tree over $T_Y \cup \{s\}$. The cost of this Steiner tree is $\mathrm{cost}(\mathcal{S}^*(Y)) = w(Y) \sum_{e \in \mathcal{S}^*(Y)} c_e$. Analogously we define the cost of an arbitrary Steiner tree $\mathcal{S}(Y)$ over $T_Y \cup \{s\}$ as $\mathrm{cost}(\mathcal{S}(Y))$.

In our analysis, we sometimes need to consider sets of nodes in τ and bound their cost collectively. To this end, we define a *chain* $\overline{Y}$ to be a set $\overline{Y} = \{Y_0, Y_1, \cdots, Y_k\}$ where for every $i < k$, Y_i is a parent of Y_{i+1}. Recall that this implies $Y_0 \supset Y_1 \supset \cdots \supset Y_k$ and $w(Y_i) \geq 2w(Y_{i+1})$ for all $i < k$. We call the node Y_0 the start of the chain $\overline{Y}$. We say that two chains $\overline{Y_1}$ and $\overline{Y_2}$ are disjoint if $(\cup_{Y \in \overline{Y_1}} Y) \cap (\cup_{Y \in \overline{Y_2}} Y) = \emptyset$.

The following is the main lemma of this section and allows us to bound the cost of large collections of nodes in τ.

Lemma 2.7. *Let $\mathcal{Y} = \{\overline{Y}_1, \overline{Y}_2, \cdots\}$ be a collection of mutually disjoint chains. That is, for any $\overline{Y}_i, \overline{Y}_j \in \mathcal{Y}$, $\overline{Y}_i$ and $\overline{Y}_j$ are disjoint. Then $\sum_{\overline{Y} \in \mathcal{Y}} \sum_{Y \in \overline{Y}} \mathrm{cost}(\mathcal{S}^*(Y))$ is at most* $2\mathrm{cost}(\mathrm{OPT})$.

Proof. For $\overline{Y} \in \mathcal{Y}$, let $\text{OPT}(\overline{Y})$ denote $\text{OPT}(\cup_{Y \in \overline{Y}} T_Y)$, the restriction of the optimal solution to the terminals associated with sets in $\overline{Y}$. Note that because the chains in $\mathcal{Y}$ are disjoint, $\sum_{\overline{Y} \in \mathcal{Y}} \text{cost}(\text{OPT}(\overline{Y})) \leq \text{cost}(\text{OPT})$. Therefore, for the rest of the proof we will argue that for any $\overline{Y} \in \mathcal{Y}$, $\sum_{Y \in \overline{Y}} \text{cost}(\mathcal{S}^*(Y)) \leq 2\text{cost}(\text{OPT}(\overline{Y}))$, and this will imply the lemma.

To prove the claim, fix a chain $\overline{Y} \in \mathcal{Y}$, and let $\overline{Y} = \{Y_0, Y_1, \cdots, Y_k\}$ where $Y_0 \supset Y_1 \supset \cdots \supset Y_k$, and $w(Y_i) \geq 2w(Y_{i+1})$ for all $i < k$. Recall that $\text{OPT}(\overline{Y})$ contains a path for every terminal in $\cup_{Y \in \overline{Y}} T_Y$. Let $\mathcal{P}_Y$ denote the collection of paths for terminals in T_Y. We say that $e \in \mathcal{P}_Y$ if $e \in \cup_{t \in T_Y} P_t$. Let $\mathcal{S}(Y)$ denote the Steiner tree over $T_Y \cup \{s\}$ defined by $\mathcal{P}_Y$ and note that $\text{cost}(\mathcal{S}^*(Y)) \leq \text{cost}(\mathcal{S}(Y)) = w(Y) \sum_{e \in \mathcal{P}_Y} c_e$. Then,

$$\sum_{Y \in \overline{Y}} \text{cost}(\mathcal{S}^*(Y)) \leq \sum_{e \in E} c_e \sum_{Y \in \overline{Y} : \mathcal{P}_Y \ni e} w(Y)$$

On the other hand, because of the containment structure of sets in $\overline{Y}$,

$$\text{cost}(\text{OPT}(\overline{Y})) = \sum_{e \in E} c_e \max_{Y \in \overline{Y} : \mathcal{P}_Y \ni e} w(Y)$$

To conclude the proof we claim that for every edge e, $\sum_{Y \in \overline{Y} : \mathcal{P}_Y \ni e} w(Y) \leq 2 \max_{Y \in \overline{Y} : \mathcal{P}_Y \ni e} w(Y)$. But this is easy to see because weights of sets $Y \in \overline{Y}$ are geometrically decreasing by a factor of at least two. □

Next we show that the tree τ can be decomposed into at most $\log P$ different collections of mutually disjoint chains. This along with Lemma 2.7 will allow us to prove our desired approximation.

Lemma 2.8. *Any demand tree τ can be decomposed into at most $\log P$ different collections of mutually disjoint chains $\mathcal{Y}_1, \mathcal{Y}_2, \cdots, \mathcal{Y}_k$, such that each node in τ belongs to exactly one collection. Here P is the number of nodes in τ.*

Proof. We decompose the tree by finding a long chain, removing it from the tree, and then recursing on the remaining subtrees. Given a tree τ with root Y_0, we start at Y_0 and follow a path down to a leaf. Let m denote the number of nodes in τ. At any node, we consider the sizes of the subtrees rooted at the node in terms of the number of nodes in the tree; the path then moves to the child corresponding to the largest subtree. Note that all of the other remaining subtrees rooted at the node have at most $m/2$ nodes.

Let $\overline{Y} = \{Y_0, Y_1, \cdots, Y_k\}$ be the path obtained. This collection of nodes is a chain by definition. Consider removing the nodes in $\overline{Y}$ from τ. We then note the following properties: (1) Every remaining connected component of the tree is of size at most $m/2$; (2) Let τ_1 and τ_2 denote any two connected components. Then $(\cup_{X \in \tau_1} X) \cap (\cup_{X \in \tau_2} X) = \emptyset$, that is, the connected components are mutually disjoint.

The second property can be proved by contradiction. If two of the components are not disjoint, then by laminarity, they contain nodes X_1 and X_2 respectively such that X_1 is an ancestor of X_2 in τ. Since X_1 and X_2 belong to different connected components, there must be a node on the path between them in τ that is in the chain $\overline{Y}$. Then, all ancestors of this node are in $\overline{Y}$ including X_1 which contradicts the fact that X_1 belongs to a connected component left after removing the chain.

We output $\{\overline{Y}\}$ as the first collection of mutually disjoint chains. (Note that this first collection has only one chain in it.) Now let $\tau_1, \tau_2, \cdots$ be the connected components left over in τ after removing the nodes in $\overline{Y}$. We recursively find collections of mutually disjoint chains in each of the components. Call these $\mathcal{Y}_{i1}, \mathcal{Y}_{i2}, \cdots$ etc. for the ith connected component. Then, we output the collections $\cup_i \mathcal{Y}_{ij}$ for each j. Since the connected components are mutually disjoint, the collections we output are also mutually disjoint. Furthermore, since the sizes of the connected components decrease by a factor of 2 each time we go down a level of recursion, it is easy to argue that the number of collections we output are bounded by $\log P$ where P is the number of nodes in the tree τ that we started out with. $\square$

We now present the main theorem for this section:

Theorem 2.2. *Let $(G, T, \mathcal{D}, \tau)$ be an instance of RAND that satisfies the conditions in Lemma 2.6. Then Algorithm 3 obtains a $(2\alpha \log P)$-approximation for the RAND over this instance where P is the number of effectively distinct packets in the instance, and α is the approximation factor of the Steiner tree algorithm used in Step 3 of the algorithm.*

Proof. For every set $X \in \mathcal{D}$, let $\mathcal{S}(X)$ denote the Steiner tree built by our algorithm over $T_X \cup \{s\}$. Then we have $\text{cost}(\mathcal{S}(X)) \leq \alpha \text{cost}(\mathcal{S}^*(X))$. We first use Lemma 2.8 to decompose the tree τ into at most $\log P$ collections of mutually disjoint chains. Call these collections $\mathcal{Y}_1, \mathcal{Y}_2, \cdots, \mathcal{Y}_k$.

The total cost of our solution can now be written as

$$\begin{aligned}
\sum_{Y \in \mathcal{D}} \text{cost}(\mathcal{S}(Y)) &= \sum_{i \leq k} \sum_{\overline{Y} \in \mathcal{Y}_i} \sum_{Y \in \overline{Y}} \text{cost}(\mathcal{S}(Y)) \\
&\leq \alpha \sum_{i \leq k} \sum_{\overline{Y} \in \mathcal{Y}_i} \sum_{Y \in \overline{Y}} \text{cost}(\mathcal{S}^*(Y)) \\
&\leq \alpha \sum_{i \leq k} 2\text{cost}(\text{OPT}) \\
&= 2\alpha k \ \text{cost}(\text{OPT})
\end{aligned}$$

Here the second inequality follows by applying Lemma 2.7. The theorem now follows by noting that $k \leq \log P$. □

Combining Theorem 2.2 with Lemma 2.6 we get the following result.

Corollary 2.1. *Algorithms 2 and 3 together obtain a $(4\alpha \log P)$-approximation for the RAND where α is the approximation factor of the Steiner tree algorithm used in the algorithm.*

We conclude this section by noting that Algorithm 3 can be implemented in a simple combinatorial fashion in $O(n^3)$ time as a generalization of Prim's algorithm for the minimum spanning tree problem as follows. This version of the algorithm obtains an $(8 \log P)$-approximation.

1. Let $\text{anc}(t)$ denote the set of ancestors of t (those with demands that are strict supersets of $\text{D}(t)$), and $\text{peer}(t)$ denote the set of its peers (those with demands identical to that of t).

2. At any step call a terminal t eligible if all of its ancestors are already connected to the root.

3. Initialize $W = \{s\}$.

4. Let $\Delta(t)$ denote the distance in G from t to its closest node in $W \cap (\text{anc}(t) \cup \text{peer}(t))$.

5. While $W \neq T$, pick the eligible terminal with the smallest $\Delta(t)$ and connect it to its closest node in $W \cap (\text{anc}(t) \cup \text{peer}(t))$. Update $W = W \cup \{t\}$ and update Δ.

2.5 Conclusion and Open Questions

In this chapter we consider network design and facility location problems with a cost structure that captures economy of scale. As mentioned in Section 2.1.2, both RAND and buy-at-bulk network design (BaBND) involve a tradeoff between connecting the source to the clients via short paths and trying to merge different paths to avail volume discounts. Therefore, even though the cost structures in BaBND and RAND are different, the fact that the sink-source multiple-sink version of BaBND has constant-factor approximations [Tal06, GKR$^+$07] suggests that we might be able to improve on the logarithmic approximation for RAND.

One possible approach towards a constant-factor approximation for RAND, with laminar demands, is to generalize the framework of Gupta et al. [GKR$^+$07]. In the context of BaBND, they address the aforementioned tradeoff by merging the paths of a randomly selected subset of terminals and individually connecting the remaining terminals to the source via short paths. At a high-level, this framework gives us an approach for RAND where a solution is constructed in multiple sample-augment phases. That is, in each phase first we select a random subset of unassigned terminals and then connect them to the source by constructing a low-cost Steiner tree that spans this subset. We satisfy the demands of the terminals selected in a phase by routing a sufficiently large set of packets through the constructed Steiner tree. In first phase we route all of Π through the constructed tree and in successive phase we route demand sets present at intermediate levels of τ. The idea is to ensure that the demands of the terminals are satisfied and the expected cost of the constructed solution is comparable to the optimal.

The question of eliminating the laminarity assumption requires further study. New ideas are required to tackle this problem, since it likely that there are no approximation preserving reductions from arbitrary instances to the ones that have laminar demands. A naive approach of modifying each demand set, say by adding or removing few packets, to generate a laminar family does not work.

Considering submodular load functions on edges gives us an important generalization of RAND and RAFL. In this setting the routing cost on an edge e is proportional to $f_e(T_e)$, where f_e is a submodular function and T_e is the set of terminals that are connected to the source through e. The class of problems in which the submodular functions across edges are different cannot be approximated within a factor of $(1 - o(1)) \log n$, under standard complexity theoretic assumptions. This inapproximability result can be established by reducing set cover to the nonuniform version of the problem. Note that the reduction at hand does not apply to the uniform version in which function f_e is the same for all

edges. In fact, a logarithmic approximation for the uniform version can be directly achieved via randomized tree embedding [FRT03]. Hence, determining if the uniform version has sublogarithmic approximations remains an interesting open question.

3 Multi-Route Cuts

3.1 Introduction

Finding small cuts in graphs is one of the most fundamental combinatorial optimization problems and there is a large literature on exact and approximate algorithms for various versions of this problem. Cut problems have numerous applications; One of the foremost among these is finding bottlenecks in communication networks. For example, the celebrated max-flow min-cut theorem states that the size of the minimum s-t cut in a network is equal to the maximum flow that can be routed between s and t. Similar (but weaker) duality theorems hold for more general communication patterns, for example, relating the maximum multicommodity flow to the minimum multicut.

From the point of view of reliability of service in the face of edge or node failures, a natural extension to finding the maximum flow in a network is to find a large flow that is spread out across multiple disjoint paths. Such a flow is called a *multi-route flow*. Multi-route flows can be related back to cuts via Menger's theorem [men]: a pair of terminals in a network admits a k-route flow (i.e. is k-edge-connected) if and only if the minimum cut between the terminals contains at least k edges. This suggests the following natural question: what is the minimum cost set of edges or vertices the removal of which reduces the connectivity of terminal pairs in the network to below a certain threshold? This is the *minimum multi-route cut problem*.

We provide approximation algorithms for multi-route cut problems. Like traditional cut problems, multi-route cut problems come in multiple flavors depending on whether we are allowed to remove edges or vertices, the desired connectivity (s-t cut, multiway cut, multicut, etc.), or whether the connectivity is in terms of edge-disjoint or node-disjoint paths. We provide constant and logarithmic approximations to several of these variants.

It is easy to see that multi-route cut problems are at least as hard as their 1-route counterparts, but they can sometimes be much harder. For example, as noted in [CK08], a reduction from (1-route) multiway cut shows that single-source multi-sink 2-route cut is APX-hard, whereas the corresponding 1-route version is equivalent to minimum s-t cut and is poly-time solvable. Likewise, we show in Section 3.5.1 that the following "red-blue" version of s-t k-route cut is NP-hard for large k.[1] In the red-blue s-t cut problem, the edge set is divided into red edges and blue edges; The red edges are associated with certain connectivities and the blue edges with certain costs; The goal is to find an s-t with total connectivity below a certain threshold and total cost minimized. This version is equivalent to k-route s-t cut when all the edge connectivities are polynomially bounded.

Multi-route flows were introduced by Kishimoto [Kis96], and have found a number of applications in communication networks [ACKN07, BCK05, BCSK07]. In a series of papers Kishimoto and others [Kis96, KT93, AO02] developed efficient algorithms for finding multi-route flows, as well as explored approximate max-flow min-cut theorems in this setting. For example, Bagchi et al. [BCKS04] showed a strong duality theorem for multi-route flows and cuts in the single-source single-sink case under a non-standard definition of the cost of a cut. More recently, Bruhn et al. [BČHK07] considered the single-source uniform costs version of the problem, that is where each edge has a cost of 1. They showed that the gap between a maximum k-route flow and a traditional (1-route) maximum flow is at most a factor of $2(1 - 1/k)$. This in turn implies a simple $2(k - 1)$ approximation for the single-source k-route cut problem. Bruhn et al. left open the question of designing sub-polynomial approximation algorithms for multi-route cut problems. Note that unlike for 1-route cut problems, in the multi-route case, the uniform cost assumption is not without loss of generality. In particular, replacing an edge of cost c with c parallel edges of cost 1 each can potentially change connectivity between terminal pairs. Therefore Bruhn et al.'s

[1] The problem is polynomial time solvable for constant k.

approximation does not extend to a general single-source multi-route cut problem.

The first non-trivial approximations for general multi-route cut problems were developed by Chekuri and Khanna [CK08]. Chekuri et al. gave LP-rounding based polylogarithmic approximations for the special case of 2-route cuts. In addition to improving upon their approximation factors we solve the two main open problems mentioned in their work—obtaining a polylogarithmic approximation for the 2-route node-disjoint multicut problem, as well as the first non-trivial approximations for k-route cuts with $k \geq 3$. Moreover, while Chekuri and Khanna's algorithms are based on a specialized rounding scheme, a main contribution of our work is to develop a general approach based on region growing to solve multi-route cut problems.

3.1.1 Results and Techniques

We consider a natural LP relaxation for multi-route cut problems and extend the "region growing" technique of Garg, Vazirani, and Yannakakis [GVY96] (see also [Shm97]) to this case, providing improved approximations for several versions of the 2-route cut problem and the first non-trivial approximations for k-route cut problems.

In a traditional multicut problem the region growing technique guarantees the existence of a cut around every terminal of cost no more than a logarithmic factor larger than the total contribution to the LP objective of edges strictly inside the cut; a logarithmic bound on the approximation then follows from the disjointness of the cuts constructed. Consider a version of the multi-route cut problem in which every edge has cost either 1 or ∞.[2] Then our region growing lemma guarantees the existence of a cut around every terminal that has few infinity-cost edges crossing it, while having cost at most a logarithmic factor larger than the contribution to the LP objective of the 1-cost edges inside the cut.

[2]This version in fact captures arbitrary cost multi-route cut problems without loss of generality.

In a traditional multicut setting, an approximation can be obtained by applying region growing successively at each terminal until all terminal pairs are disconnected; In particular, every region has diameter less than 1 and therefore cannot contain more than one terminal belonging to the same terminal pair. In the multi-route setting there are two problems with this approach. First, our LP relaxation defines h different metrics over the graph, one for each terminal pair. Regions are grown with respect to the metric corresponding to the terminal under consideration. Therefore, we can no longer ensure that no terminal pairs survive within a region, and are forced to recurse within regions. This leads to a further logarithmic loss in the approximation factor. Second, as we remove successive regions from the graph, since we do not remove all the boundary edges (specifically, the infinite cost ones), some paths through these regions survive and it becomes tricky to analyze the final connectivity between terminal pairs.

We are able to overcome all of these difficulties for the case of 2-route cuts, and provide $O(\log^2 h)$ approximations to 2-route multicut and multiway cut, where the previous best known approximations due to Chekuri et al. [CK08] were $O(\log^2 n \log h)$ and $O(\log n \log h)$ respectively. Here h is the number of terminals, and n is the number of vertices in the graph. Furthermore, while Chekuri et al.'s technique does not extend to the node-disjoint version of 2-route multicut, ours extends easily and naturally giving the same approximation factors.

While our region growing lemma extends to the case of k-route problems with arbitrary k, overcoming the difficulties outlined above appears to require significantly new techinques. In fact, for general connectivity thresholds $k > 2$, the integrality gap of our LP relaxation can be as large as k (see Section 3.5.2). We therefore explore bicriteria approximations. Straightforward applications of region growing lead to a $(2, 2h)$ and a $(2h, 2)$ bicriteria approximation, where the first factor refers to the approximation in thresholds, and the second to the approximation in cost. By avoiding overlap between successive cuts more carefully, we show how to obtain a $(6, O(\sqrt{h} \log h))$ approximation. These are the first

non-trivial approximations in the k-route cut case, for $k \geq 3$. We also consider some special cases of the problem. When h is constant or when all the edges have equal cost, we can obtain a $(4, 4)$ and a $(2, 4)$ approximation respectively. The last result holds even when different terminals have different connectivity thresholds.

While we focus on edge-weighted multi-edge-disjoint-route cuts, all of our algorithms and analyses extend with little effort to the node weighted and node-disjoint versions as well. We detail the changes required for the node-weighted node-disjoint version in Section 3.3.2; The other two combinations are identical.

We summarize our main results in Table 3.1 below. See Section 3.2 for precise definitions of the various instances of multi-route cut.

Problem	Previous best result	Our result
SS-2-EDRC, SS-2-NDRC	$O(\log n)$ [CK08]	$O(\log h)$
MW-2-EDRC, MW-2-NDRC	$O(\log n \log h)$ [CK08]	$O(\log^2 h)$
MC-2-EDRC	$O(\log^2 n \log h)$ [CK08]	$O(\log^2 h)$
MC-2-NDRC	–	$O(\log^2 h)$
SS-k-EDRC	–	$(6, O(\sqrt{h} \ln h))$
SS-k-EDRC-Uniform	–	$(2, 4)$
SS-k-EDRC (constant h)	–	$(4, 4)$

Table 3.1: A summary of our main results.

We consider several versions of the multi-cut problem and they are formally defined in Section 3.2. This section also describes the notation and presents a new linear programming relaxation for the problem. Our main technical tool, a region growing lemma, for approximating multi-route cuts is developed in Section 3.3. Using the lemma, in Section 3.4 we design approximation algorithms for the special case in which the connectivity thresholds are equal to two. For single-source multiple-sink cut instances with arbitrary connectivity requirements, Section 3.5 presents bicriteria approximations where the connectivity requirement is violated by a constant factor. We conclude and discuss subsequent results in Section 3.6.

3.2 Problem Definition and Preliminaries

Given a graph $G = (V, E)$, a pair of nodes $u, v \in V$ are called k-edge-connected if there are k edge-disjoint paths between u and v in G, and are called k-node-connected if there are k node-disjoint paths between u and v in G. In multi-route cut problems our goal is to remove a small number (or more generally a low cost set) of edges or nodes from a given graph so as to reduce the connectivity of given pairs of nodes to below certain thresholds.

Like traditional cut problems multi-route cut problems come in different flavors. We begin by formally defining the most general versions we consider. The input to the multicut version of the edge-disjoint-route-cut problem (MC-EDRC) is a graph G with costs c_e on edges, h pairs of vertices called terminals, $\{(s_1, t_1), (s_2, t_2), \cdots, (s_h, t_h)\}$, and connectivity thresholds, k_i for pair (s_i, t_i). The goal is to produce a minimum cost set of edges $E' \subseteq E$, such that for each i, s_i and t_i are at most $(k_i - 1)$-edge-connected in the graph $(V, E \setminus E')$. Note that in the traditional multicut problem $k_i = 1$ for all i. In the node-disjoint-route multicut (MC-NDRC) problem the goal is to produce a set of edges $E' \subseteq E$, such that for each i, s_i and t_i are at most $(k_i - 1)$-node-connected in the graph $(V, E \setminus E')$. Note that although we will mostly talk about edge weighted versions of the problem, our techniques and analyses extend to the node weighted versions as well.

We further study the following special cases:

- k-EDRC or k-NDRC: here all the connectivity thresholds are equal to a common value k.
- 2-EDRC or 2-NDRC: a special case of the above with $k = 2$.
- MW-EDRC or MW-NDRC (MultiWay multi-route cut): we are given a set $T = \{t_1, \cdots, t_h\}$ of terminals with a common connectivity threshold k for every pair $(t_i, t_j) \in T \times T$.

- SS-EDRC or SS-NDRC (Single Source multiple sink multi-route cut): we are given a single source s and a set $T = \{t_1, \cdots, t_h\}$ of terminals with connectivity thresholds k_i for the pair (s, t_i).
- SS-EDRC-Uniform: the version of SS-EDRC where every edge has a cost of 1.

LP Relaxation

The following LP is a relaxation of the MC-EDRC. Other edge-disjoint cut problems have similar LP relaxations. In any integral solution to this LP, edges with $x_e = 1$ are cut, and the (at most) $(k_i - 1)$ edges with $y_e^i = 1$ represent an s_i-t_i cut of size at most $(k_i - 1)$ in the residual graph. Note that the LP defines h different shortest path metrics on the graph.

$$\tilde{z} = \min \sum_{e \in E} x_e c_e \qquad \textbf{(ED-LP)}$$

$$\begin{aligned} \text{subject to} \quad & \sum_{e \in E} y_e^i \leq k_i - 1 && \forall i \in [h] \\ & d^i(u, v) = x_e + y_e^i && \forall i \in [h], e = (u, v) \in E \\ & d^i \text{ is a metric} && \forall i \in [h] \\ & d^i(s_i, t_i) \geq 1 && \forall i \in [h] \end{aligned}$$

We remark that the algorithms developed by Chekuri et al. [CK08] were based on a similar but weaker LP.

The LP relaxation for the node-disjoint version MC-NDRC is similar (see Section 3.3.2).

Notation

We now develop some notation useful in our analysis.

- For a given subset of vertices, $S \subseteq V$, $G[S]$ denotes the subgraph induced by S.
- $\mathbf{d}^{\ell}$ denotes the shortest path metric obtained when edge lengths are given by ℓ_e. We use $\mathbf{d}^i$ as short-hand for the metric $\mathbf{d}^{x+y^i}$.
- $\mathcal{B}^{\mathbf{d}}(u,r) = \{v \mid \mathbf{d}(u,v) \leq r\}$ denotes a ball of radius r around u under metric $\mathbf{d}$. We use $\mathcal{B}^i$ as short-hand for $\mathcal{B}^{\mathbf{d}^i}$.
- For a set $S \subset V$, $\delta(S) = \{(u,v) \mid (u,v) \in E, \ \ |S \cap \{u,v\}| = 1\}$ is the set of boundary edges of S.
- For $S \subset V$, $E(S) = \{(u,v) \mid (u,v) \in E, \ \ |S \cap \{u,v\}| \geq 1\}$ is the set of all edges incident on S. We use $E^i(u,r)$ as short-hand for $E(\mathcal{B}^i(u,r))$.
- For a set S, the "k-cost" of S, denoted $\Gamma^k(S)$, is the total cost of all but the $k-1$ most expensive edges in $\delta(S)$: $\Gamma^k(S) = \min_{F \subseteq \delta(S); |F| \leq k-1} \sum_{e \in \delta(S) \setminus F} c_e$.
- Finally, for $\beta > 0$, the "(β, x)-volume" of a set S measures the total contribution of all the edges incident on the set to the objective function: $\mathcal{V}^{\beta,x}(S) = \beta + \sum_{e \in E(S)} x_e c_e$.

3.3 Region Growing for Multi-Route Cuts

Our main tool for constructing approximations to multi-route cut problems is a region growing lemma. The lemma states that given a feasible solution to the program (**ED-LP**) above, we can find a cut with low $2k$ cost.

We begin by presenting the lemma for the edge-disjoint version of the problem. The following subsection shows the modifications necesary to obtain a version of the lemma for the node-disjoint case.

3.3.1 The Edge-Disjoint Case

Lemma 3.1. *Let $G = (V, E)$ be a graph with costs c_e on edges and terminals s and t, and x and y be vectors of lengths on edges, such that $\boldsymbol{d}^{x+y}(s,t) \geq 1$ and $\sum_e y_e \leq k-1$. Then there exists a radius $r < 1$ such that for $S = \mathcal{B}^{x+y}(s,r)$, the $2(k-1)$-cost of S, $\Gamma^{2(k-1)}(S)$, is no more than $\alpha\mathcal{V}^{\beta,x}(S)$, where $\alpha = 2\ln\left(\mathcal{V}^{\beta,x}(V)/\beta\right)$.*

Proof. For ease of exposition, we assume without loss of generality that there exists a small constant ϵ such that for every edge e, x_e and y_e are multiples of ϵ, and $M = 1/\epsilon$ is an integer. We first modify the graph G such that for every edge e, $x_e + y_e = \epsilon$ and only one of these values is non-zero. Specifically we break every edge e into $(x_e + y_e)/\epsilon$ parts with costs c_e each; We assign an x value of ϵ and a y value of 0 to x_e/ϵ of these parts, and assign a y value of ϵ and x value of 0 to the remaining parts. It is clear that the new instance still satisfies the constraints in the theorem statement. Also note that while costs $\Gamma^{2(k-1)}$ stay the same as before, volumes decrease, and so it suffices to prove the lemma for this new fragmented version of the graph.

We will consider M balls centered at s and show that one of these satisfies the criteria in the theorem. For $0 \leq i \leq M$, let $B_i = \mathcal{B}^{x+y}(s, i\epsilon)$, $\mathcal{V}_i^x = \mathcal{V}^{\beta,x}(B_i)$, and $\Gamma_i = \Gamma^{2(k-1)}(B_i)$. We also define the "change in volume", $\Delta\mathcal{V}_i^x$, for the ith ball as $\Delta\mathcal{V}_i^x = \sum_{e\in\delta(B_i)} x_e c_e$, with $\Delta\mathcal{V}_0^x = \beta$. Note that $\mathcal{V}_i^x \geq \sum_{a=0}^{i} \Delta\mathcal{V}_a^x$. Also the sets $\delta(B_i)$ are disjoint.

We now prove a few statements about how the change in volume relates to the $2(k-1)$-cost of a ball. Let index set Ω be defined as follows: $\Omega = \{i \mid i \in [M], \ \ \epsilon\Gamma_i \leq \Delta\mathcal{V}_i^x\}$. The following lemmas show that Ω is large.

Lemma 3.2. *For any $i \in [M]$, $\epsilon\Gamma_i > \Delta\mathcal{V}_i^x$ implies $\sum_{e\in\delta(B_i)} y_e \geq 2(k-1)\epsilon$.*

Proof. We prove the contrapositive statement. Say we have $\sum_{e\in\delta(B_i)} y_e < 2(k-1)\epsilon$, thus edges e in $\delta(B_i)$ with $y_e = \epsilon$ is strictly less than $2(k-1)$. Let E_y be the set of such edges.

For the rest of the edges in $\delta(B_i)$ the x value in turn is ϵ. Therefore $\Delta\mathcal{V}_i^x = \epsilon \sum_{e\in\delta(B_i)\setminus E_y} c_e$. Since $|E_y| < 2(k-1)$ and therefore $\sum_{e\in\delta(B_i)\setminus E_y} c_e \ \geq \ \Gamma_i$, we have $\Delta\mathcal{V}_i^x \geq \epsilon\Gamma_i$. □

Lemma 3.3. $|\Omega| \geq M/2$.

Proof. Recall that $\sum_e y_e \leq k-1$. Now consider an index $i \in [M] \setminus \Omega$. Lemma 3.2 shows that for such an index $\sum_{e\in\delta(B_i)} y_e \geq 2(k-1)\epsilon$. If the number of indices in $[M] \setminus \Omega$ is strictly more than $M/2$, we would have

$$\sum_e y_e \geq \sum_{i\in[M]\setminus\Omega} \sum_{e\in\delta(B_i)} y_e > \frac{M}{2} 2(k-1)\epsilon = k-1$$

which gives us a contradiction. □

We also require the following inequality for the cost analysis below.

Fact 3.1. *For any sequence of positive numbers: $a_0, a_1, ..a_N$, the following bound holds*

$$\frac{a_1}{a_0+a_1} + \frac{a_2}{a_0+a_1+a_2} + ... + \frac{a_N}{a_0+a_1+....+a_{N-1}+a_N} \leq \ln\left(\frac{a_0+a_1+...+a_{N-1}+a_N}{a_0}\right)$$

Before proving the fact we show how it leads to the theorem. We focus on the set Ω. Let $\sigma(1), \cdots, \sigma(N)$ be the sequence of indices in Ω with $N = |\Omega|$. For $\sigma(i) \in \Omega$, let $\tilde{\mathcal{V}}^x_{\sigma(i)} = \sum_{a=0}^{i} \Delta\mathcal{V}^x_{\sigma(a)}$. Note that $\tilde{\mathcal{V}}^x_{\sigma(i)} \leq \mathcal{V}^x_{\sigma(i)}$ for all i.

Recall that for all $\sigma(i) \in \Omega$ we have $\epsilon\Gamma_{\sigma(i)} \leq \Delta\mathcal{V}^x_{\sigma(i)}$. Suppose there does not exist an index satisfying the required property, that is, $\forall i \in \Omega$ we have $\Gamma_i > \alpha\mathcal{V}_i^x \geq \alpha\tilde{\mathcal{V}}_i^x$. Thus for all $i \in \Omega$ we have $\frac{1}{\epsilon}\Delta\mathcal{V}_i^x \geq \alpha\tilde{\mathcal{V}}_i^x$, or

$$\frac{\Delta\mathcal{V}_i^x}{\tilde{\mathcal{V}}_i^x} > \alpha\epsilon$$

Summing the above inequality for all the indices in Ω we get

$$\frac{\Delta\mathcal{V}_1^x}{\tilde{\mathcal{V}}_1^x} + \frac{\Delta\mathcal{V}_2^x}{\tilde{\mathcal{V}}_2^x} + ... + \frac{\Delta\mathcal{V}_N^x}{\tilde{\mathcal{V}}_N^x} > \alpha\epsilon N = \frac{\alpha}{2}$$

Here the last statement follows from Lemma 3.3 by noting that $N \geq M/2$.

On the other hand, setting $a_0 = \tilde{\mathcal{V}}_0^x = \beta$ and $a_i = \Delta\mathcal{V}_i^x$, we can apply Fact 3.1 to get the following which gives us a contradiction.

$$\frac{\Delta\mathcal{V}_1^x}{\tilde{\mathcal{V}}_1^x} + \frac{\Delta\mathcal{V}_2^x}{\tilde{\mathcal{V}}_2^x} + ... + \frac{\Delta\mathcal{V}_N^x}{\tilde{\mathcal{V}}_N^x} \leq \ln\left(\frac{\tilde{\mathcal{V}}_N^x}{\tilde{\mathcal{V}}_0^x}\right) \leq \ln\left(\frac{\mathcal{V}^{\beta,x}(V)}{\beta}\right) = \frac{\alpha}{2}$$

It remains to prove Fact 3.1.

Proof of Fact 3.1 We prove the fact by induction over N. For the base case, with positive integers it is true that $\frac{a_1}{a_0+a_1} \leq \ln\left(\frac{a_0+a_1}{a_0}\right)$. This follows from the fact that for positive x we have $\ln(1+x) \geq \frac{x}{1+x}$ (the function values are equal at $x = 0$ and rate of growth of $\ln(1+x)$ is more than the other). We set $x = a_1/a_0$ here.

By induction hypothesis we assume the inequality to fold for $N-1$. Denoting by S_k the sum $\sum_{i=0}^{k} a_k$, we have

$$\frac{a_1}{S_1} + \frac{a_2}{S_2} + ... + \frac{a_{N-1}}{S_{N-1}} \leq \ln\left(\frac{S_{N-1}}{a_0}\right)$$

Now using the fact that $\ln(1+x) \geq \frac{x}{1+x}$ again with $x = a_N/S_{N-1}$ we have $\ln\left(\frac{S_N}{S_{N-1}}\right) = \ln\left(\frac{a_N+S_{N-1}}{S_{N-1}}\right) \geq \frac{a_N}{S_N}$. Adding the two inequalities we get,

$$\ln\left(\frac{S_N}{S_{N-1}}\right) + \ln\left(\frac{S_{N-1}}{a_0}\right) \geq \frac{a_1}{S_1} + \frac{a_2}{S_2} + ... + \frac{a_{N-1}}{S_{N-1}} + \frac{a_N}{S_N}$$

And hence the claim follows,

$$\frac{a_1}{S_1} + \frac{a_2}{S_2} + ... + \frac{a_{N-1}}{S_{N-1}} + \frac{a_N}{S_N} \leq \ln\left(\frac{S_N}{a_0}\right)$$

This concludes the proof of the region growing lemma. □

Note that in the special case of $k = 2$, the above lemma gives a bound on the 2-cost of the region, which is equivalent to leaving out exactly one edge. Therefore we incur no loss in the connectivity threshold in this case.

While the above lemma suffices to construct approximate solutions to the SS-EDRC, for the multicut version we require additional properties from cuts in our algorithms and so need to consider cuts around both s_i and t_i for a terminal pair (s_i, t_i). We therefore develop the following "two-sided" region growing lemma which shows that we can simultaneously find good disjoint cuts for both s_i and t_i.

Lemma 3.4. *Let $G = (V, E)$ be a graph with costs c_e on edges and terminals s and t, and x and y be vectors of lengths on edges, such that $\boldsymbol{d}^{x+y}(s, t) \geq 1$ and $\sum_e y_e \leq k - 1$. Then there exist radii $r_1 < 1$ and $r_2 > r_1$ such that for $S_1 = \mathcal{B}^{x+y}(s, r_1)$, and $S_2 = V \setminus \mathcal{B}^{x+y}(s, r_2)$, we have for $\alpha = 2 \ln \mathcal{V}^{\beta,x}(V)/\beta$:*

- $\Gamma^{2(k-1)}(S_1) \leq 2\alpha \mathcal{V}^{\beta,x}(S_1)$, *and,*
- $\Gamma^{2(k-1)}(S_2) \leq 2\alpha \mathcal{V}^{\beta,x}(S_2)$.

Proof. The proof is nearly identical to that of Lemma 3.1. Once again we consider balls with radii ϵi centered at s, and let Ω denote the index set of balls with few ($< 2(k-1)$) "y-edges". As before, the cardinality of this set, N, is at least $M/2$. Consider the balls corresponding to the first $N/2$ indices in Ω. A volume argument identical to the one used previously shows that for one of these balls, say B_i, we must have $\Gamma_i \leq 2\alpha \mathcal{V}_i^x$, so $r_1 = i\epsilon$.

In order to find r_2 we consider the remaining $N/2$ balls in reverse order. That is, set $B'_1 = V \setminus B_N$, $B'_2 = V \setminus B_{N-1}$ and so on. We can again reapply the volume argument to get a set B'_j satisfying the required properties; r_2 would then be $(N - j + 1)\epsilon$. In particular the

$2(k-1)$ cost of the set is no more than (β, x)-volume inside it (or outside the corresponding ball B_{N-j+1}). By construction $r_2 > r_1$, so we are done. □

Finally, we note that if we are allowed to charge the cost of a cut to the volume of the entire graph and not just of the cut itself, then we can obtain a stronger version of the region growing lemma:

Lemma 3.5. *Let $G = (V, E)$ be a graph with costs c_e on edges and terminals s and t, and x and y be vectors of lengths on edges, such that $\boldsymbol{d}^{x+y}(s,t) \geq 1$ and $\sum_e y_e \leq k-1$. Then there exists a radius $r < 1$ such that for $S = \mathcal{B}^{x+y}(t,r)$, the $2(k-1)$-cost of S, $\Gamma^{2(k-1)}(S)$, is no more than $2\mathcal{V}^{\beta,x}(\mathcal{B}^{x+y}(t,1))$.*

Proof. The proof is similar to that of Lemma 3.1. Again we consider balls with radii ϵi centered at t, and let Ω denote the index set of balls with few ($< 2(k-1)$) "y-edges". As before, the cardinality of this set, N, is at least $M/2$, with $M = 1/\epsilon$.

We continue to use the same notation. Thus for $0 \leq i \leq M$, we have $B_i = \mathcal{B}^{x+y}(s, i\epsilon)$, $\Gamma_i = \Gamma^{2(k-1)}(B_i)$ and change in volume, $\Delta\mathcal{V}_i^x$, for the ith ball as $\Delta\mathcal{V}_i^x = \sum_{e\in\delta(B_i)} x_e c_e$.

Note that by disjointness of successive balls we have the following inequality

$$\sum_{a=1}^{M} \Delta\mathcal{V}_a^x \leq \mathcal{V}^{\beta,x}(\mathcal{B}^{x+y}(t,1))$$

Let $\sigma(1), \cdots, \sigma(N)$ be the sequence of indices in Ω. We have established that for all $\sigma(i) \in \Omega$, $\epsilon\Gamma_{\sigma(i)} \leq \Delta\mathcal{V}_{\sigma(i)}^x$. Combining the last two inequalities we have $\sum_{i=1}^{N} \epsilon\Gamma_{\sigma(i)} \leq \mathcal{V}^{\beta,x}(\mathcal{B}^{x+y}(t,1))$. Since $\epsilon = 1/M$ and $N \geq M/2$ by averaging argument there exists an index j such that $\Gamma_j \leq 2\mathcal{V}^{\beta,x}(\mathcal{B}^{x+y}(t,1))$ and so the claim follows. □

3.3.2 Region Growing for Node-Weighted Node-Disjoint-Route Cuts

We next consider the version of multi-route cut where we are required to produce minimum weight node cuts, and satisfy thresholds on node-disjoint paths. The LP relaxation for the node-disjoint version MC-NDRC is very similar to program (**ED-LP**). Here $\mathcal{P}_i$ is the set of all paths between s_i and t_i. Although this LP is exponential in size, it has an equivalent polynomial-size formulation as above.

$$\tilde{z} = \min \sum_{v \in V} x_v c_v \qquad \textbf{(ND-LP)}$$

$$\text{subject to} \quad \sum_{v \in V} y_v^i \leq k_i - 1 \qquad \forall i \in [h]$$

$$\sum_{v \in P} (x_v + y_v^i) \geq 1 \quad \forall P \in \mathcal{P}_i,\ \forall i \in [h]$$

Region growing works almost in the same way for node-disjoint-route cuts as for edge-disjoint-route cuts. Most importantly, we define volumes and volume increments in terms of the boundary vertices of a set rather than in terms of boundary edges. We sketch below the differences in our definitions and argument to incorporate node-disjointness as well as node costs:

- $\mathbf{d}^{x+y^i}$ is the shortest path metric where the length of a path is the sum of the x_v and y_v^i values of vertices present in it (both end points included). As before we use $\mathbf{d}^i$ as short-hand for the metric $\mathbf{d}^{x+y^i}$.
- As before $\mathcal{B}^{\mathbf{d}}(u, r)$ denotes a ball of radius r around u under metric $\mathbf{d}$, and $\mathcal{B}^i$ is short-hand for $\mathcal{B}^{\mathbf{d}^i}$.

- For a set $S \subset V$, the set of boundary vertices of S, $\Delta(S)$ is defined as $\{v \in S \mid \exists (u,v) \in \delta(S)\}$ where $\delta(S)$ are the boundary edges of S.
- For a set S, $\Gamma^k(S)$ denotes the total cost of all but the $k-1$ most expensive vertices in $\Delta(S)$: $\Gamma^k(S) = \min_{F \subseteq \Delta(S); |F| \leq k-1} \sum_{v \in \Delta(S) \setminus F} c_v$.
- For $\beta > 0$, we define the "(β, x)-volume" of a set S to be the total contribution of all the vertices in the set to the objective function: $\mathcal{V}^{\beta,x}(S) = \beta + \sum_{v \in S} x_v c_v$.
- As in the proof of Lemma 3.1 we pick an $\epsilon > 0$ that divides all the x and y values, and fragment the graph by breaking each vertex v into $n_v = (x_v + y_v)/\epsilon$ vertices $v^{(0)}, \cdots, v^{(n_v)}$, each with a cost of c_v, and with edges $(v^{(a)}, v^{(a+1)})$ for all $a \in [n_v]$. We replace an edge (u,v) in the original graph by edge $(u^{(n_u)}, v^{(0)})$ if $\mathbf{d}^{x+y}(s,u) \leq \mathbf{d}^{x+y}(s,v)$ and by $(v^{(n_v)}, u^{(0)})$ otherwise. Again it is easy to see that this transformation preserves the costs $\Gamma^{2(k-1)}$ of balls around s, but decreases volumes.
- We define B_i to be the ball $\mathcal{B}^{x+y}(s, i\epsilon)$.
- Finally we set the incremental volumes $\Delta\mathcal{V}_i^x$ to be $\sum_{v \in \Delta(B_i)} x_v c_v$ and $\mathcal{V}_i^x = \mathcal{V}^{\beta,x}(B_i)$. As before we have $\mathcal{V}_i^x = \sum_{k=0}^{i} \Delta\mathcal{V}_k^x$.

We therefore get the following node-disjoint analog of Lemma 3.4. The other two lemmas have similar analogues.

Lemma 3.6. *Let $G = (V, E)$ be a graph with costs c_v on vertices and terminals s and t, and x and y be vectors of weights on vertices, such that $\boldsymbol{d}^{x+y}(s,t) \geq 1$ and $\sum_v y_v \leq k-1$. Then there exist radii $r_1 < 1$ and $r_2 > r_1$ such that for $S_1 = \mathcal{B}^{x+y}(s, r_1)$, and $S_2 = V \setminus \mathcal{B}^{x+y}(s, r_2)$, we have for $\alpha = 2 \ln \mathcal{V}^{\beta,x}(V)/\beta$:*

- $\Gamma^{2(k-1)}(S_1) \leq 2\alpha \mathcal{V}^{\beta,x}(S_1)$, *and,*
- $\Gamma^{2(k-1)}(S_2) \leq 2\alpha \mathcal{V}^{\beta,x}(S_2)$.

3.4 2-Route Cuts

We now apply the region growing technique to 2-route cut problems. A key difference from how the technique is used to find (1-route) multicuts is that we are now working with h different metrics and grow successive regions under different metrics. Nevertheless, in the single-source multi-sink case we can use region growing in much the same way as it is used to find (1-route) multicuts: we successively find small cuts around terminals, remove them from the graph, and recurse on the remaining graph. Unfortunately this simple approach does not work for the more general multicut version of the problem. In particular, while for a traditional multicut no region contains two terminals belonging to the same pair, in our setting it can. We therefore cannot simply remove subgraphs and ignore them; we must recursively produce cuts within each subgraph. We show how to do this repeated cutting at most $\log h$ times in each subgraph, leading to a final approximation factor of $O(\log^2 h)$.

3.4.1 Single-Source Multiple-Sink 2-Route Cuts

Once again we will focus on the edge-disjoint case; our algorithm and analysis for the node-disjoint case is identical. Recall that program (**ED-LP**) provides a fractional solution (x, y) to the problem with cost $\sum_e c_e x_e$, and $\sum_e y_e^i \leq 1$ for all $i \in [h]$. This fractional solution defines h different shortest-path metrics $\mathbf{d}^i$ with $\mathbf{d}^i(e) = x_e + y_e^i$ for all $i \in [h]$ and $e \in E$.

Our algorithm for SS-2-EDRC is given in Figure 3.1. The algorithm starts with an optimal fractional solution to the program (**ED-LP**). At every step it picks an arbitrary terminal still connected to the source, uses the region growing lemma to find an appropriate cut around the terminal, and removes the entire cut from the graph. It continues until no terminals are left. Then for every cut found, it puts back in the graph the most expensive edge in the cut.

Input: Graph $G = (V, E)$ with costs c_e, source s, terminals $T = \{t_1, \cdots, t_h\}$, fractional solution (x, y) with $\sum_e y_e^i \leq 1$ for all $i \in [h]$ and $\mathbf{d}^{x+y^i}(s, t_i) \geq 1$ for all $i \in [h]$. $\tilde{z} = \sum_e x_e c_e$ and $\alpha = 2\ln(h+1)$.
Output: A set of edges E' of cost at most $\alpha\tilde{z}$ such that for all $i \in [h]$ s and t_i are at most 1-edge-connected in $(V, E \setminus E')$.

1. Initialize $T' \leftarrow T$ and $V' \leftarrow V$.
2. Pick an arbitrary terminal t_i from T'. For the rest of the iteration we consider lengths of edges only under metric $\mathbf{d}^i$ and the ball $\mathcal{B}^i(t_i, r)$ is defined over $G[V']$.
3. Let $\beta = \tilde{z}/h$. Pick a radius $r_i \in [0, 1)$ such that $\Gamma^2(\mathcal{B}^i(t_i, r_i))$ is no more than $\alpha\mathcal{V}^{\beta,x}(\mathcal{B}^i(t_i, r_i))$.
4. Set $S_i \leftarrow \mathcal{B}^i(t_i, r_i)$ and update $V' \leftarrow V' \setminus S_i$.
5. Let T' be the set of terminals that are connected to s in $G[V']$.
6. Repeat steps (2) to (5) until $T' = \emptyset$.
7. Let the partitions generated in the previous steps be S_1 through S_l. Let $\delta'(S_i)$ the set of edges crossing S_i and present in $G[V \setminus \cup_{j=1}^{i-1} S_j]$. Return the set $E' = \bigcup_{i=1}^{l} (\delta'(S_i) \setminus \{e_i^{max}\})$, where e_i^{max} is the maximum cost edge in $\delta'(S_i)$.

Figure 3.1: Algorithm *SS-2EDRC*—Algorithm for single-source multi-sink 2-EDRC

To analyze the algorithm we first note that for all $i \in [h]$ the vectors (x, y^i) together satisfy the conditions in Lemma 3.1 with $k = 2$, and moreover, $2\ln\left(\mathcal{V}^{\beta,x}(V)/\beta\right) \leq 2\ln((\beta+\tilde{z})/\beta) = 2\ln(h+1) = \alpha$. Therefore, we can always find a radius satisfying the conditions of step (3) in the algorithm and the algorithm terminates. It remains to prove that the set E' generated by the algorithm is a legitimate 2-route cut, and analyze its cost. We do this next.

Lemma 3.7. *Given a graph $G = (V, E)$ with terminal set T let E' be the set of edges selected by algorithm* SS-2EDRC *then in the graph $H = (V, E \setminus E')$ the universal source s is at most* 1*-edge-connected to any terminal present in T.*

Proof. We claim that in graph $H = (V, E \setminus E')$, for any a, a path from the sink s to a vertex v, contained in partition S_a, must cross e_a^{max}.

The proof is by induction over a. For the base case we consider a vertex v in S_1. S_1 is a cut separating v and s, so any path from v to s must intersect $\delta(S_1) = \delta'(S_1)$. But $\delta'(S_1) \setminus E' = \{e_1^{max}\}$, therefore our claim holds.

By the induction hypothesis we assume that the claim is true for all vertices in all partitions S_1 to S_{a-1}. Now consider a vertex v in S_a. Consider any path P in $H = (V, E \setminus E')$ from v to s, and let e' be the first edge (starting from v) on P that is contained in $\delta(S_a)$. For the sake of contradiction assume that P does not contain e_a^{max}, so $e' \neq e_a^{max}$. This implies $e' \notin \delta'(S_a)$ because $\delta'(S_a) \setminus E' = \{e_a^{max}\}$. Therefore, $e' \in \delta(S_a) \setminus \delta'(S_a)$. This means that e' got removed from consideration when some partition S_j was removed with $j < a$. One of the vertices of e' survived to be included in S_a thus $e' \in \delta'(S_j)$. But the only edge of $\delta'(S_j)$ present in $E \setminus E'$ (that is in H) is e_j^{max}, so $e' = e_j^{max}$ and P must now go from a vertex inside S_j to s without recrossing e_j^{max}. This is contradicted by the induction hypothesis.

To prove the lemma first note that the above claim immediately implies that any terminal contained in some partition S_j is at most 1-connected to s in H. Finally we consider terminals t in the final subgraph $G[V \setminus \cup_{j=1}^{l} S_j]$ disconnected from s. Consider any path from such a terminal to s in H, say $\overline{e}$ is the first edge (starting at t) on P which has exactly one of its vertices in some partition S_i. Since t is disconnected from s in the final subgraph such an edge must exist. Note that $\overline{e} \in \delta'(S_i)$. The only way that $\overline{e}$ is not in E' is that it is e_i^{max} but for the rest of the path to be in H, P must connect a vertex contained in S_i to s without crossing e_i^{max} which by our claim is not possible. Thus terminals which are present in the final subgraph $G[V \setminus \cup_{j=1}^{l} S_j]$ disconnected from s remain disconnected from s in H. □

Finally we can analyze the cost of the solution. Note that by construction the l edge sets

$E(S_1)$, $E(S_2)$, $\cdots$, $E(S_l)$ are pairwise disjoint. Therefore, $\sum_{i=1}^{l} \mathcal{V}^{\beta,x}(S_i) \leq \beta l + \sum_{e \in E} x_e c_e \leq \beta h + \tilde{z} = 2\tilde{z}$. The cost of the final set E' generated by the algorithm is exactly $\sum_i \Gamma^2(S_i)$, which is at most $\alpha \sum_i \mathcal{V}^{\beta,x}(S_i) \leq 2\alpha\tilde{z}$ by construction. The theorem below now follows from noting that $\tilde{z}$ is no more than the cost of the optimal 2-route cut.

Theorem 3.1. *Algorithm* SS-2EDRC *generates a* 2*-edge-route cut of cost no more than* $4\ln(h+1)$ *times the optimal.*

3.4.2 2-Route Multicuts

We now consider the multicut version of 2-EDRC. As before our algorithm successively uses region growing to construct cuts around terminals. However, instead of recursing only on the remaining graph as in the single-source case, this time we need to recurse on both the components in the graph. We show below that by constructing the cuts appropriately, the depth of recursion is at most $\log h$, and therefore we can find a 2-route cut of cost no more than $O(\log^2 h)$ times the optimal.

The algorithm for 2-EDRC multicut is given in Figure 3.2.

We first note that the vectors (x, y^i) satisfy the conditions of Lemma 3.4 for terminals (s_i, t_i) and therefore we can always find radii r_1 and r_2 satisfying the conditions in Step (3).

Next we show that the cost of the final set E' is not too large. Let $\mathcal{S} = \{S_1, S_2, \cdots, S_l\}$, where l is the total number of cuts formed. We claim that every cut in $\mathcal{S}$ is contained in no more than $\log h$ other cuts in $\mathcal{S}$. This follows by noting that, by construction, for any two sets $S_a \subset S_b$ in $\mathcal{S}$, the number of terminal pairs in $G[S_a]$ is no more than half the number of terminal pairs in $G[S_b]$. We therefore have the following lemma.

Lemma 3.8. *For a given edge* $e \in E$ *there are at most* $\log h$ *cuts in* $\mathcal{S}$ *such that* $e \in E(S_i)$.

Proof. Note that cuts in $\mathcal{S}$ form a laminar family that is for $S_i, S_j \in \mathcal{S}$ either one is contained in the other or they do not intersect at all. Now consider the following collection of cuts

Input: Graph $G = (V, E)$ with costs c_e, a set of source-sink pairs $T = \{(s_i, t_i)\}$ along with metric weights on edges: x_e and y_e^i (one for each source sink pair in T). $\tilde{z} = \sum_e x_e c_e$, $\beta = \tilde{z}/h$, and $\alpha = 2\ln(h+1)$. Also given are global variables p and E'. (Initially $p = 0$ and $E' = \emptyset$.)
Output: A set of edges E' such that for all $(s_i, t_i) \in T$, s_i and t_i are at most 1-edge-connected in $(V, E \setminus E')$.

1. If T is empty, stop.
2. Pick a source-sink pair (s_j, t_j) from T.
3. Find radii $r_1 \in [0, 1)$ and $r_2 \in (r_1, 1]$ such that $\Gamma^2(\mathcal{B}^j(s_j, r_1)) \leq 2\alpha \mathcal{V}^{\beta,x}(\mathcal{B}^j(s_j, r_1))$ and $\Gamma^2(\mathcal{B}^j(s_j, r_2)) \leq 2\alpha \mathcal{V}^{\beta,x}(V \setminus \mathcal{B}^j(s_j, r_2))$. Note that $\mathcal{B}^j(s_j, r_1)$ and $V \setminus \mathcal{B}^j(s_j, r_2)$ do not intersect.
4. Increment the global index count: $p \leftarrow p + 1$.
5. If the number of connected source-sink pairs in $G[\mathcal{B}^j(s_j, r_1)]$ is less than the number of connected source-sink pairs in $G[V \setminus \mathcal{B}^j(s_j, r_2)]$ then the pth cut, S_p, is chosen to be $\mathcal{B}^j(s_j, r_1)$, otherwise it is chosen to be $V \setminus \mathcal{B}^j(s_j, r_2)$.
6. Let $e_p^{max} = \text{argmax}_{e \in \delta'(S_p)} c_e$, where $\delta'(S_p)$ is defined to be the set of boundary edges of S_p present in the graph in the current recursive call.
7. Update the global set of edges, $E' \leftarrow E' \cup (\delta'(S_p) \setminus \{e_p^{max}\})$.
8. Recurse on $G[S_p]$ with terminal set being the source-sink pairs connected in $G[S_p]$ and on $G[V \setminus S_p]$ with terminal set being the of source-sink pairs connected in it.

Figure 3.2: Algorithm *MC-2EDRC*—Algorithm for 2-EDRC Multicut

$\mathcal{S}_e = \{S \mid S \in \mathcal{S},\ e \in E(S)\}$ also write $l_e = |\mathcal{S}_e|$. Since all the cuts in $\mathcal{S}_e$ intersect we have the following chain of containments over them: $S_{\pi(l_e)} \subseteq S_{\pi(l_e-1)} \ldots \subseteq S_{\pi(1)}$, where $\pi(i)$ is the cut index of the ith cut. By our earlier argument the length of such a containment chain can be no more than $\log h$. □

Finally, in order to bound the cost, as before we have $\sum_p \Gamma^2(S_p) \leq 2\alpha \sum_p \mathcal{V}^{\beta,x}(S_p) = 2\alpha(\beta h + \sum_p \sum_{e \in E(S_p)} x_e c_e)$. Unlike in the single-source case, the edges sets $E(S_p)$ are not disjoint, however, by Lemma 3.8 we have $\sum_p \mathcal{V}^{\beta,x}(S_p) \leq \beta h + \log h \sum_{e \in E} x_e c_e \leq (\log h + 1)\tilde{z}$.

Therefore, the cost of our cut is bounded by $O(\log^2 h)$ times $\tilde{z}$.

It remains to prove that we obtain the desired connectivity among terminal pairs; For this we establish the following useful lemma. We say that a pair of vertices u, v are *first separated* by a cut $S_i \in \mathcal{S}$, if $|S_i \cap \{u, v\}| = 1$ and for all $j < i$, $|S_j \cap \{u, v\}| \neq 1$.

Lemma 3.9. *Given $S_i \in \mathcal{S}$, let u, v be a pair of vertices first separated by S_i. Then any u-v path P in $H = (V, E \setminus E')$ must contain e_i^{max}.*

Proof. The proof is by induction over the cut index i. For the base case suppose that $u \in S_1$ and $v \notin S_1$. Now any path P from u to v must contain an edge from $\delta(S_1)$, say e is the first such edge (starting from u). When S_1 is constructed by MC-2EDRC, the subgraph under consideration is G itself, so $\delta'(S_1) = \delta(S_1)$. The only edge of $\delta'(S_1)$ present in $E \setminus E'$ is e_1^{max} so e must be e_1^{max}.

Next we prove the claim for S_i. Let u, v be a pair of vertices first separated by S_i such that $u \in S_i$ and $v \notin S_i$. Now for contradiction assume that there exists a path P from u to v in $H = (V, E \setminus E')$ such that $e_i^{max} \notin P$. Now P must contain an edge in $\delta(S_i)$; Say $e = (u', v')$ is the first such edge in P (starting from u), with $u' \in S_i$ and $v' \notin S_i$. Again it is easy to see that $e \in \delta(S_i) \setminus \delta'(S_i)$. This implies that by the time S_i was constructed e' had been removed from the graph. Thus $e \in \delta'(S_j)$ for some $j < i$. Since P is in H we have $e = e_j^{max}$.

Next we show that S_j first separates v' and v but by (strong) induction hypothesis there is no path from v' to v that does not contain e_j^{max}, which implies that P can not proceed from v' to v in H. Say we label the vertices in P as follows $P = u \to u_1 \to u_2 \to ...u' \to v' \to ....v$. Here u through u' are in S_i and $v' \notin S_i$. Now consider the point of time at which the algorithm constructed S_j. The graph under consideration at that time was $\overline{G} = (V, E \setminus (\cup_{k=1}^{j-1} \delta'(S_k))$. We know that $e = (u', v') \in E \setminus (\cup_{k=1}^{j-1} \delta'(S_k))$ since it is in $\delta'(S_j)$. Also the path $u \to u_1 \to ... \to u'$ is present in $\overline{G}$; This follows from the fact that all cuts in $\mathcal{S}$ constructed before S_i either

contain no vertex of S_i or contain all the vertices in S_i. Moreover there is a path between u and v until S_i is constructed (it is the lowest index cut separating u and v), so there is path from v' to v in $\overline{G}$. In other words no cut before j separates v' and v. Moreover until we get to the construction of S_i the path between u' and v is intact. But $e \in \delta'(S_j)$, so S_j separates u' and v'. Thus S_j separates v' and v. This implies that S_j first separates v' and v, and we are done. □

Corollary 3.1. *All source-sink pairs (s_i, t_i) in T at most 1-connected in $H = (V, E \setminus E')$.*

Proof. If we have some cut in $\mathcal{S}$ separating source-sink pair (s_i, t_i), then we consider the lowest index cut separating s_i and t_i; Say it is S_j, that is, S_j *first separates* s_i and t_i. Then by the previous lemma any path from s_i to t_i must pass through e_j^{max}. This by Menger's Theorem implies that s_i and t_i are 1-connected.

Note that there might be a source-sink pair (s_i, t_i) that gets disconnected (MC-2EDRC continues till all the source-sink pair get disconnected) but no cut in $\mathcal{S}$ separates them. We show that such a pair remains disconnected in H hence proving the corollary.

For contradiction assume there is a path P from s_i to t_i in H. Since s_i and t_i are disconnected at the end of algorithm's execution, P must contain an edge from $\delta'(S_j)$ for some $j \in [l]$. Say $\tilde{e} = (\tilde{u}, \tilde{v})$ is the first edge (starting from s_i) on P such that $\tilde{e} \in \delta'(S_j)$. P is in H so for $\tilde{e}$ to be in $E \setminus E'$ we must have $\tilde{e} = e_j^{max}$. Next we show that S_j first separates $\tilde{v}$ and t_i so by the previous lemma, P can not proceed from $\tilde{v}$ to t_i without crossing e_j^{max} again, giving rise to a contradiction.

Say we label the path as follows $P = s_i \to w_1 \to w_2 \to ... \to \tilde{u} \to \tilde{v} \to ..t_i$. Note that all the edges on P before $\tilde{e} = (\tilde{u}, \tilde{v})$ are present in H. Thus none of these edges belong to any $\delta'(S_i)$ for $i \in [l]$. Since $\tilde{e} \in \delta'(S_j)$ and all edges on P between s_i and $\tilde{u}$ are present in H we have that S_j separates s_i and $\tilde{v}$. Then since no cut in $\mathcal{S}$ separates s_i and t_i, we have that S_j separates $\tilde{v}$ and t_i. Moreover we claim that no cut with a smaller index separates $\tilde{v}$ and

t_i. To see this, suppose that S_k for $k < j$ separates $\tilde{v}$ and t_i. Since S_k does not separate s_i and t_i (no cut does) we have that S_k separates s_i and $\tilde{v}$. But this implies that we must have an edge of $\delta(S_k)$ on every path between s_i and $\tilde{v}$; In particular the segment of P connecting s_i to $\tilde{v}$ must contain an edge from $\delta'(S_k)$. But this contradicts the assumption that $\tilde{e}$ was the first edge on P contained in some $\delta'()$. Thus no path in H connects s_i and t_i. □

From the cost analysis and Corollary 3.1 we get the following theorem.

Theorem 3.2. *Algorithm* MC-2EDRC *generates a* 2*-edge-disjoint-route multicut of cost no more that* $O(\log^2 h)$ *times the optimal.*

3.5 k-Route Cuts

We now consider the EDRC and NDRC with larger connectivity thresholds. In Subsection 3.5.2 it is shown that (**ED-LP**) has a polynomial integrality gap even for the simple case of an s-t k-EDRC. A similar example can be constructed for (**ND-LP**). Given this large integrality gap, we investigate bicriteria approximations to the EDRC. An (α, β) approximation for the k-EDRC is a cut of cost at most β times the optimal and the removal of which reduces the connectivity between the terminal pair (s_i, t_i) to $\alpha(k_i - 1)$, for every i.

3.5.1 NP-Hardness of k-Route s-t Cut

Unlike MW-EDRC and MC-EDRC, the $k = 1$ case of k-Route s-t cut is not NP-Hard. In fact, for any constant k, we can solve k-route s-t cut optimally in polynomial time. Specifically, we can guess a set E' of $k-1$ "witness" edges and find the minimum s-t cut in $G \setminus E'$. However, in this section we show it is unlikely that the problem is easy for large k by proving that a capacitated version of k-route s-t cut is NP-hard. We call this version the *red-blue k-route s-t cut* and defined as follows. We are given a graph $G = (V, E)$ with a

source s and sink t, and a connectivity threshold k. The edge set E is partitioned into red edges, E_R and blue edges, E_B. Edges e in E_R have connectivities k_e associated with them and edges in E_B have cost c_e associated with them. The problem is to find an s-t cut C such that $\sum_{e \in \delta(C) \cap E_R} k_e \leq k-1$ and the cost $\sum_{e \in E_B \cap \delta(C)} c_e$ is minimized.

We reduce the knapsack problem to the red-blue k route cut problem. In an instance of the knapsack problem we are given a universe of n items along with a size bound B. Here item i has value v_i and size z_i. The objective is to find a subset S of items such that $\sum_{i \in S} z_i \leq B$ and the value $\sum_{i \in S} v_i$ is maximized. We construct the graph G with n *intermediate vertices* numbered 1 to n, one for each item, along with a source s and a sink t. We connect the source s to each of the n intermediate vertices with a red edge (that is, $E_R = \{(s,i)\}_{i=1}^{n}$). Edge (s,i) is associated with connectivity $k_{(s,i)} = z_i$. Similarly we connect the sink t to the intermediate vertices with blue edges with costs $c_{(i,t)} = v_i$.

It follows that finding a B-size bounded set S of items that achieves maximum value is equivalent to finding a min-cost $(B+1)$-route cut in the constructed graph. In particular consider a cut C with $t \in C$ and $s \notin C$. Then it is easy to see that the set $C \setminus \{t\}$ is a valid solution to the Knapsack problem. Furthermore the value achieved by this solution is exactly $\sum_i v_i$ minus the cost of the cut C. Therefore, minimizing the cost of C is equivalent to maximizing the value of a feasible knapsack solution, and we get the following theorem.

Theorem 3.3. *Red-blue k-route s-t cut is NP-hard.*

We note that red-blue k-route s-t cut is equivalent to k-route s-t cut when the connectivities $k_{(u,v)}$ on edges are polynomially bounded. However, the algorithms developed by us apply to this more general version even with arbitrary edge connectivities. In particular, we can formulate a linear program (**ED-LP-RB**) for the red-blue version, that is similar to the one developed for the k-route cut problem in Section 3.2. Primarily here we ensure that only blue edges have non zero x_e values and only red edges have non zero y_e values.

$$\tilde{z} = \min \sum_{e \in E_B} x_e c_e \qquad \textbf{(ED-LP-RB)}$$

$$\text{subject to} \quad \sum_{e \in E_R} y_e k_e \leq k - 1$$

$$x_e = 0 \qquad \forall e \in E_R$$

$$y_e = 0 \qquad \forall e \in E_B$$

$$d(u,v) = x_e + y_e \qquad \forall e = (u,v) \in E$$

$$d \text{ is a metric}$$

$$d(s,t) \geq 1$$

The following lemma is a counter-part to Lemma 3.5 and shows that we can obtain a $(2,2)$-bicriteria approximation for the red-blue k-route s-t cut problem. As before we use $\mathbf{d}^{x+y}$ to denote the shortest-path metric defined by lengths x_e and y_e on edges.

Lemma 3.10. *Let $G = (V, E_B \cup E_R)$ be a graph with cost c_e and connectivity k_e values associated with edges in E_B and E_R respectively. Also let x be vectors of lengths on edges in E_B and y be vectors of lengths on edges in E_R, such that $\boldsymbol{d}^{x+y}(s,t) \geq 1$. Then there exists a radius $r < 1$ such that for $S = \mathcal{B}^{x+y}(s,r)$ we have $\sum_{e \in \delta(S) \cap E_R} k_e \leq 2(k-1)$ and $\sum_{e \in \delta(S) \cap E_B} c_e \leq 2 \sum_{e \in E_B} x_e c_e$.*

Proof. We argue along the lines of the proof of Lemma 3.5. For simplicity, we assume without loss of generality that there exists a small constant ϵ such that for every edge e, x_e or y_e, as the case may be, is a multiple of ϵ, and $M = 1/\epsilon$ is an integer. We first modify the graph G such that for every edge e, $x_e = \epsilon$ if the edge is blue or $y_e = \epsilon$ if the edge is red. Specifically we break every edge $e \in E_B$ into x_e/ϵ parts with costs c_e each and every edge $e \in E_R$ into y_e/ϵ parts with connectivity k_e each. As before we maintain that only

blue edges have non-zero x values and only red edges have non-zero y values. It is clear that the new instance still satisfies the constraints in the lemma statement.

We will consider M balls centered at s and show that one of these satisfies the criteria in the lemma. For $0 \leq i \leq M$, let $B_i = \mathcal{B}^{x+y}(s, i\epsilon)$. Note that the edge sets $\delta(B_i)$ are disjoint. Consider index set Ω of cuts B_i for which connectivity factor is maintained within a factor of two, that is, $\Omega = \{i \mid i \in [M],\ \sum_{e \in E_R \cap \delta(B_i)} k_e \leq 2(k-1)\}$.

We claim that $|\Omega| \geq M/2$. To see this, note that for all indices j in $[M] \setminus \Omega$ we have $\sum_{e \in E_R \cap \delta(B_j)} k_e > 2(k-1)$. Then for such indices, $\sum_{e \in E_R \cap \delta(B_j)} y_e k_e > 2(k-1)\epsilon$. Noting that the edges sets $\delta(B_i)$ are disjoint, we get the following sequence of inequalities.

$$k - 1 \geq \sum_{e \in E_R} y_e k_e \geq \sum_{j \in [M] \setminus \Omega} \sum_{e \in E_R \cap \delta(B_j)} y_e k_e > 2(k-1)\epsilon |[M] \setminus \Omega|$$

That is, $|[M] \setminus \Omega| < M/2$, and therefore, $|\Omega| \geq M/2$.

Next, denote the cost of edges crossing B_i as $\Gamma_i = \sum_{e \in \delta(B_i) \cap E_B} c_e$. Recall that $x_e = \epsilon$ for all edges $e \in \delta(B_i) \cap E_B$, for all $i \in [M]$. Hence $\sum_{e \in \delta(B_i) \cap E_B} x_e c_e = \epsilon \Gamma_i$. Just considering indices in Ω we have

$$\sum_{i \in \Omega} \epsilon \Gamma_i \leq \tilde{z}$$

As shown above, the cardinality of Ω is at least $M/2$. So by an averaging argument there exists an index i^* in Ω such that $\Gamma_{i^*} \leq 2\tilde{z}$. As $i^* \in \Omega$ we also have $\sum_{e \in E_R \cap \delta(B_{i^*})} k_e \leq 2(k-1)$. □

3.5.2 Integrality Gap for LP

We present a graph where the optimal integral solution has cost $\Omega(k)$ times the optimal fractional solution. Consider the chain graph in Figure 3.3 and suppose that we wish to find a $k+1$-route cut separating source s from sink t. The graph has $k+1$ parallel edges

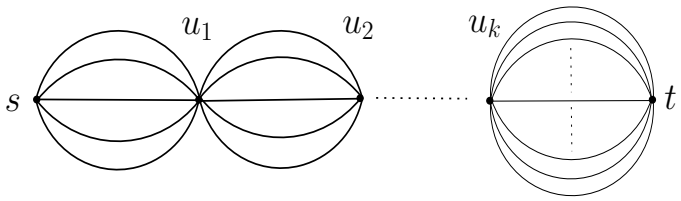

Figure 3.3: Integrality Gap example

between u_i and u_{i+1} for all $i \in [k-1]$, each such edge has infinite cost. Also there are $k+1$ infinite cost edges between s and u_1. Finally we have $2k$ unit cost edges between u_k and t. A feasible fractional solution with cost no more than 2 is obtained as follows: for every edge of infinite cost set $y_e = \frac{1}{k+1}$ and $x_e = 0$, and, for all edges with unit cost, that is edges between u_k and t, we set $x_e = \frac{1}{k+1}$ and $y_e = 0$. Note that $\sum_e y_e$ is no more than k and $\sum_e x_e c_e$ is less than 2. Also under the specified edge lengths distance between s and t_i is 1. Hence we have a feasible fractional solution with cost no more than 2. However any integral solution, with finite cost, in order to ensure that the number of edge disjoint paths between s and t is no more than k can only remove k edges between u_k and t. Hence an optimal integral cut has cost k, giving us an integrality gap of $\Omega(k)$.

3.5.3 The Difficulty of Applying Some Naïve Approximations

As mentioned earlier, although the region growing lemma works in the $k \geq 3$ case as well, applying it successively for different terminals leads to the connectivity thresholds being violated by a large factor. Consider, for example, the following algorithm for the single-source k-EDRC. We solve **ED-LP**; then for each i, we successively apply region growing to the pair (s, t_i) and remove the resulting cut C_i from the graph; our final cut is the collection of all but the k most expensive edges in each C_i. The cost of this cut can be bounded by $O(\log h\ \tilde{z})$ using Lemma 3.1. However, in the final graph, for any terminal t_j with cut C_j there may be several paths to s through cuts C_i for $i < j$ that do not cross C_j.

Therefore, the best bound we can obtain on the connectivity between s and t_j using this approach is $(k-1)h/2$. In other words, we get an $(O(h), O(\log h))$ approximation.

This approach can be modified slightly to obtain an $(O(h), 2)$ approximation. In particular, we solve the **ED-LP** and combine all the h metrics into a single metric. That is, set y_e to be $\sum_{i=1}^{h} y_e^i$. The metric d, defined by setting $d(u,v) = x_e + y_e$ for all edges $e = (u,v)$, separates the source s from all the h terminals — $d(s,t_i) \geq 1$ for all $i \in [h]$. Moreover we have $\sum_{e \in E} y_e \leq kh$. Thus by Lemma 3.5 we can find a cut S which $2kh-1$ separates s from all the terminals and has cost no more than $2\tilde{z}$. This gives us a $(2h, 2)$ approximation.

One way of avoiding this increase in connectivity is to find successive cuts in the original graph itself, instead of throwing away the previously found cuts. This ensures that connectivity thresholds are maintained to within a factor of 2. However, the cost of the solution can blow up to $O(h\tilde{z})$, implying a $(2, 2h)$ approximation. Specifically, the $(2, 2h)$ approximation is obtained by solving **ED-LP** and applying region growing separately to each pair (s, t_i). Lemma 3.5 implies that for each terminal t_i we can find a cut S_i which $2(k-1)$ separates the terminal from the source and for which we have $\Gamma^{2(k-1)}(S_i) \leq 2\mathcal{V}^{\beta,x}(\mathcal{B}^{x+y}(t_i, 1))$. Note that $\mathcal{V}^{\beta,x}(\mathcal{B}^{x+y}(t_i, 1))$ is no more than $\tilde{z}$ and so the total cost of the all such cuts is no more than $2h\tilde{z}$. This gives us a $(2, 2h)$ approximation.

3.5.4 Single-Source Multiple-Sink k-Route Cuts

In the remainder of this section, we focus on the single-source case and present a number of different algorithms. The first is a general $(6, O(\sqrt{h} \ln h))$ approximation that relies on a stronger LP (**ED-LP+**) defined below (see Equation (3.1)). We then consider two special cases — in the first the number of terminals is constant, and in the second all edges in the graph have equal cost. We present a $(4, 4)$ and a $(2, 4)$ approximation for these respectively.

These are the first non-trivial approximations for any variant of the k-route cut problem with $k \geq 3$.

A key observation that we use for each of these algorithms is that the integral solution to SS-EDRC forms a family of laminar cuts. In particular, let E' be the set of edges removed in an integral solution. By Menger's Theorem we know that for each terminal t_i there exists a set of at most $k_i - 1$ edges whose removal disconnects t_i from s in $(V, E \setminus E')$. Consider any such set of edges, and let C_i be the set of vertices in the connected component containing t_i after these edges have been removed. We call this set a witness for t_i. The following lemma shows that for any integral feasible solution we can find a collection of witness sets that are laminar, that is, no two of the sets cross.

Lemma 3.11. *For any integral feasible solution to the SS-EDRC there exists a collection of witness sets that is laminar. When all terminals have equal connectivity thresholds, there exists a family of witness sets such that each pair of sets is either identical or disjoint.*

Proof. Let E' be an integral solution for the given SS-EDRC. Recall the definition of a witness set. By Menger's Theorem we know that for each terminal t_i there exists a set of at most $k_i - 1$ edges whose removal disconnects t_i from s in $(V, E \setminus E')$. A witness set for t_i is the connected component containing t_i that is formed when we remove any such set of edges from $E \setminus E'$.

Let $H = (V, E \setminus E')$, and note that by definition for any $i \in [h]$ the edge connectivity of t_i and s in H is no more than $k_i - 1$. Of the witness sets for t_i that have the fewest edges crossing them, let C_i be a smallest set in terms of cardinality. We now show that no two smallest witness sets C_i and C_j can cross. Suppose for the sake of contradiction that C_i and C_j cross each other, that is, all three sets $C_i \cap C_j$, $C_i \setminus C_j$ and $C_j \setminus C_i$ are non-empty. We define the following mutually disjoint sets of edges, here we have $\delta_H(S) = \{(u, v) \in E \setminus E' \mid |\{u, v\} \cap S| = 1\}$

- $O_i = \{(u,v) \in \delta_H(C_i) \mid \{u,v\} \cap C_j = \phi\}$
- $O_j = \{(u,v) \in \delta_H(C_j) \mid \{u,v\} \cap C_i = \phi\}$
- $I_i = \{(u,v) \in \delta_H(C_i) \mid |\{u,v\} \cap C_j| = 2\}$
- $I_j = \{(u,v) \in \delta_H(C_j) \mid |\{u,v\} \cap C_i| = 2\}$

There are three possible cases:

- Suppose that $t_i \in C_i \setminus C_j$ and $t_j \in C_j \setminus C_i$. Then, if $|I_j| < |I_i|$, then $C_i \setminus C_j$ forms a smaller t_i-s cut than C_i, contradicting the fact that C_i is a witness for t_i. Likewise we cannot have $|I_i| < |I_j|$. Therefore $|I_i| = |I_j|$, but then $C_i \setminus C_j$ is a strictly smaller witness set for t_i, again contradicting our choice of C_i.
- Suppose that $t_i \in C_i \setminus C_j$ and $t_j \in C_j \cap C_i$. This time we must have $|I_i| = |O_j|$ but then $C_i \cap C_j$ forms a strictly smaller witness set for t_j.
- Finally, suppose that $t_i, t_j \in C_i \cap C_j$. As before we have $|I_i| = |O_j|$ and $|I_j| = |O_i|$ but then $C_i \cap C_j$ forms a strictly smaller witness set for both t_i and t_j.

Therefore the witness sets form a laminar family of cuts.

Note that when all the connectivity thresholds are equal, if there are witness sets C_i and C_j with $C_i \subsetneq C_j$, C_j also forms a witness set for t_i. Therefore the lemma holds. □

3.5.4.1 A $(6, O(\sqrt{h}\log h))$ Bicriteria Approximation for Single-Source Cuts

In Figure 3.4 we present a $\left(6, O(\sqrt{h}\ln h)\right)$ bicriteria approximation algorithm for SS-kEDRC with general edge costs. The algorithm requires an optimal solution an augmented version of **ED-LP**. In particular, we add the following constraint to the LP.

$$\mathbf{d}^i(u,t_i) + \mathbf{d}^j(u,t_j) \geq \mathbf{d}^i(t_i,t_j) \;\; \forall i,j \in [h], u \in V \tag{3.1}$$

Input: Graph $G = (V, E)$ with costs c_e, source s, terminals $T = \{t_1, \cdots, t_h\}$, fractional solution (x, y) that is feasible for **ED-LP+** with connectivity thresholds $k_i = k \ \forall i \in [h]$. $\tilde{z} = \sum_e x_e c_e$ and $\alpha = 2\ln(h+1)$.
Output: A set of edges E' of cost at most $O(\alpha\sqrt{h})\tilde{z}$ such that for all $i \in [h]$, s and t_i are at most $6(k-1)$-edge-connected in $(V, E \setminus E')$.

1. Initialize $E' \leftarrow \emptyset$. Let $\beta = \tilde{z}/h$. Set $T' \leftarrow T$.
2. If there is a terminal $t_i \in T'$ such that $|T' \cap \mathcal{B}^i(t_i, 2/3)| \geq \sqrt{|T'|}$, do:
   a) Pick a radius $r_i \in [2/3, 1)$ with $S = \mathcal{B}^i(t_i, r_i)$ such that $\Gamma^{6(k-1)}(S) \leq 3\alpha\mathcal{V}^{\beta,x}(S)$.
   b) Let $F(S)$ be the $6(k-1)$ most expensive edges in $\delta(S)$. Set $E' \leftarrow E' \cup (\delta(S) \setminus F(S))$; $T \leftarrow T \setminus S$; $T' \leftarrow T' \setminus S$.
3. Otherwise, while $T' \neq \emptyset$, do:
   a) Pick a terminal $t_i \in T'$, and a radius $r_i \in [0, 1/3)$ with $S = \mathcal{B}^i(t_i, r_i)$ such that $\Gamma^{6(k-1)}(S) \leq 3\alpha\mathcal{V}^{\beta,x}(S)$.
   b) Let $F(S)$ be the $6(k-1)$ most expensive edges in $\delta(S)$. Set $E' \leftarrow E' \cup (\delta(S) \setminus F(S))$; $T \leftarrow T \setminus S$; $T' \leftarrow T' \setminus \mathcal{B}^i(t_i, 3/4)$.
4. If $T \neq \emptyset$, set $T' \leftarrow T$ and go to Step 2, otherwise return the cut E'.

Figure 3.4: Algorithm *SS-kEDRC*—Algorithm for single-source multi-sink k-EDRC

The augmented program is denoted **ED-LP+**. It is easy to see from Lemma 3.11 that **ED-LP+** is a valid relaxation of the SS-k-EDRC. We note that the integrality gap instance of subsection 3.5.2 applies to this new LP as well. The new constraint is primarily required in Lemma 3.13 to show that the sets S found in Step 3 (that are constructed under different metrics) are disjoint.

Let us now analyze the algorithm. We first note that we can always find the cuts required for Steps 2a and 3a. For the first, note that if we set x_e and y_e^i to be zero inside $\mathcal{B}^i(t_i, 2/3)$ and scale them up by a factor of 3 outside the ball, then the pair (s, t_i) satisfies the requirements of Lemma 3.1, and so we can find the desired cut. For the second, if we

scale x_e and y_e^i by a factor of 3 inside $\mathcal{B}^i(t_i, 1/3)$ and set them to 0 outside the ball, then again the pair (s, t_i) satisfies the requirements of Lemma 3.1, and we can find the desired cut.

Next we claim that the connectivity thresholds are satisfied to within a factor of 6. To see this, consider for any terminal t_i the iteration in which t_i is removed from T and let S be the corresponding cut found. Then, S separates t_i from s and we remove all but $6(k-1)$ edges from $\delta(S)$. Therefore our claim follows. Finally, we present a cost analysis. We first show that the algorithm has few iterations.

Lemma 3.12. *In each iteration of Steps 2 to 3 the size of T decreases by an additive $\sqrt{T}$.*

Proof. If Step 2 is executed the lemma follows immediately. Otherwise, note that in each inner loop of Step 3 we remove at most $\sqrt{T'}$ terminals from T'. So the loop gets executed at least $\sqrt{T'}$ times. Each time we decrease the size of T by at least 1. Therefore the lemma follows. □

A simple consequence of this lemma is that the algorithm has at most $O(\sqrt{h})$ iterations. The following lemma bounds the cost of a single iteration and completes the analysis.

Lemma 3.13. *In any execution of Step 2 or Step 3 of the algorithm the total cost of the edges included in E' is no more than $6\alpha\tilde{z}$, where $\tilde{z}$ is the value of the **ED-LP+**.*

Proof. If Step 2 is executed then the total cost of the edges included is $\Gamma^{6(k-1)}(S)$ which is no more than $3\alpha\mathcal{V}^{\beta,x}(S)$, which in turn is bounded by $3\alpha\tilde{z}$.

Next consider Step 3, and let $S_1, \cdots, S_{h'}$ be the collection of cuts constructed in a single execution of this step. Then we have that the cost of the edges removed in this step is at most $3\alpha\sum_j \mathcal{V}^{\beta,x}(S_j)$. We claim that the sets S_j are disjoint which implies that $\sum_j \mathcal{V}^{\beta,x}(S_j) \leq 2\tilde{z}$, and the total cost for this step is bounded by $6\alpha\tilde{z}$.

Input: Graph $G = (V, E)$ with costs c_e, source s, terminals $T = \{t_1, \cdots, t_h\}$, a partition $\mathcal{P}$ with l sets over terminals along with fractional solution (x, y) satisfying **ED-LP-Part**.
Output: A set of edges E' of cost at most $4\tilde{z}$ such that for all $i \in [h]$ s and t_i are at most $4(k-1)$-edge-connected in $(V, E \setminus E')$.

1. Double the value of x_e and y_e^i for all edges and for all $i \in [l]$.
2. Repeat for all $1 \leq i \leq l$:
   a) Construct meta node v_i by merging all terminals in partition set P_i.
   b) Find a cut S separating v_i from s contained in $\mathcal{B}^i(v_i, 1)$ that satisfies $\Gamma^{4(k-1)}(S) \leq 2\mathcal{V}^{\beta,x}(\mathcal{B}^i(v_i, 1))$. Here $d^i()$ is the metric associated with P_i.
   c) Set $F(S)$ to be the set of $4(k-1)$ most expensive edges in $\delta(S)$. Update $E' \leftarrow E' \cup (\delta(S) \setminus F(S))$.

Figure 3.5: Algorithm *SS-kEDRC-const*—Algorithm for single-source multi-sink k-EDRC with a constant number of terminals.

To prove the claim, suppose that there are two sets S_1 and S_2, corresponding to terminals t_1 and t_2, picked in Step 3 such that $S_1 \cap S_2 \neq \emptyset$. Then for some $u \in S_1 \cap S_2$, $\mathbf{d}^1(u, t_1) \leq 1/3$, and $\mathbf{d}^2(u, t_2) \leq 1/3$, but $\mathbf{d}^1(t_1, t_2) > 2/3$. This directly contradicts constraint (3.1) in **ED-LP+**. □

We therefore get the following theorem.

Theorem 3.4. *Algorithm SS-kEDRC gives a* $\left(6, O(\sqrt{h} \ln h)\right)$ *bicriteria approximation for the SS-k-EDRC.*

3.5.4.2 The Constant h Case

Recall from Lemma 3.11 that the witness sets for terminals in the SS-k-EDRC are disjoint. When the number of terminals is constant, we can guess the "correct" partition of terminals into groups with identical witness sets. Incorporating this information into the linear

program, and finding a good s-t k-route cut for every group gives us a $(4, 4)$ approximation for the SS-k-EDRC.

We know from Lemma 3.11 that for the SS-kEDRC the witness sets of terminals corresponding to any integral solution are laminar. In fact the collection forms a partition, that is there is a collection of l *mutually disjoint* witness sets: $\{C_1, \ldots, C_l\}$, such that each terminal is contained in one of them. The collection imposes a partition on the terminals. Also, in any integral solution, if terminals t_a and t_b are contained in the same witness set, we have $d^a(t_a, t_b) = d^b(t_a, t_b) = 0$; On the other hand if they are in different cuts we have $d^a(t_a, t_b) = d^b(t_a, t_b) = 1$. We denote by $\mathcal{P}$ the induced partition over terminals: $\mathcal{P} = \{P_1, P_2, \ldots, P_l\}$, where P_j is the set of terminals contained in C_j.

Next we present a linear program and the associated algorithm (see Figure 3.5) which if given the partition $\mathcal{P}$ imposed by an integral solution produces a set of edges that $4(k-1)$ separates every terminal from the source and has cost no more than four times the integral solution. When h is constant we can apply the algorithm over all possible partitions and thus achieve a $(4, 4)$ approximation. The linear program essentially determines l metrics, one for each partition in $\mathcal{P}$ and imposes the corresponding separation requirements.

$$\tilde{z} = \min \sum_{e \in E} x_e c_e \qquad \textbf{(ED-LP-Part)}$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{e \in E} y_e^i \leq k - 1 && \forall i \in [l] \\
& d^i(u, v) = x_e + y_e^i && \forall i \in [l], e = (u, v) \in E \\
& d^i \text{ is a metric} && \forall i \in [l] \\
& d^i(s, t_a) \geq 1 && \forall i \in [l], \forall t_a \in P_i \\
& d^i(t_a, t_b) \geq 1 && \forall i \in [l], \forall t_a \in P_i, \forall t_b \notin P_i
\end{aligned}$$

The algorithm is similar to the algorithm for single-source multi-sink 2-EDRC in Section 3.4.1 but in addition exploits the fact that each of the partitions have a distance of 1 between them in the optimal LP solution. In particular, we employ the improved region growing lemma (Lemma 3.5) to argue that the total cost is small.

In order to analyze the algorithm, note that by doubling the x_e and y_e values we have ensured that balls centered at different meta nodes v_i constructed in step (3) of the algorithm are disjoint. Since the x_e values are scaled up by two we have the following: $\sum_{i=1}^{l} \mathcal{V}^{\beta,x}(\mathcal{B}^i(v_i, 1)) \leq 2\tilde{z}$. By Lemma 3.5 we can find a cut S for each meta node v_i, separating the terminals in set P_i from s. This ensures that E' is a legitimate $4(k-1)$ route cut for all the terminals. Finally we have $\Gamma^{4(k-1)}(S) \leq 2\mathcal{V}^{\beta,x}(\mathcal{B}^i(v_i, 1))$ for all $i \in [l]$. Combining the last two inequalities we get that the total cost of E' is no more than $4\tilde{z}$. Hence the algorithm achieves a $(4, 4)$ bicriteria approximation.

3.5.4.3 The Uniform Costs Case

Next we consider single-source instances with general connectivity requirements (that is, different terminals are associated with different k_i), but where every edge has a cost of 1. We give a $(2, 4)$ bicriteria approximation. Our approach is simple: we ignore terminals that are already less than $2(k_i - 1)$ connected to the source; for the rest we use the characterization in Lemma 3.11 to argue that cost of a minimum (1-route) cut separating each terminal from the source is no more than 4 times that of the minimum multi-route cut. We therefore find and output the minimum 1-route cut. Figure 3.6 presents the details.

Theorem 3.5. *Algorithm* SS-EDRC-Uniform *returns a set of edges E' of cost at most* 4OPT *such that for all $i \in [h]$, s and t_i are at most $2(k_i - 1)$-edge-connected in $(V, E \setminus E')$.*

Proof. Terminals with less than $2(k_i - 1)$ edge connectivity do not influence the correctness, while terminals that are more than $2(k_i - 1)$ connected to s are totally disconnected from s.

Input: Graph $G = (V, E)$, set of terminals $T = \{t_1, t_2, ..t_h\}$ with connectivity requirements k_i, and a source vertex s.
Output: A set of edges E' of cost at most 4OPT such that for all $i \in [h]$ s and t_i are at most $2(k_i - 1)$-edge-connected in $(V, E \setminus E')$.

1. Remove all terminals t_i from T that are at most $2(k_i - 1)$ edge connected to s in G.
2. Using a standard mincut algorithm find a set of edges E' that disconnects s from every terminal in T.

Figure 3.6: Algorithm *SS-EDRC-Uniform*—Algorithm for single-source multi-sink EDRC with uniform costs

So the claim about connectivity follows.

Now consider an optimal solution E_{OPT} for the problem, and let $\mathcal{C} = \{C_i\}$ be the collection of witness sets guaranteed by Lemma3.11. Let $\mathcal{C}'$ be the subcollection of sets C_i such that for all $C_j \in \mathcal{C}$, $C_i \not\subset C_j$. We claim that $\cup_{C_i \in \mathcal{C}'} \delta(C_i)$ is a multicut for T of cost no more than 4OPT. The first part of the claim follows immediately by noting that each terminal in T is contained in some set $C_i \in \mathcal{C}'$ whereas $s \notin \cup_i C_i$.

For the second part of the claim, consider any $C_i \in \mathcal{C}'$; t_i is the terminal associated with this set. Let $E_i^* = E_{\mathrm{OPT}} \cap \delta(C_i)$. Since t_i is at least $2(k_i - 1)$ connected to s in G, $|\delta(C_i)| \geq 2(k_i - 1)$. On the other hand, by the feasibility of E_{OPT}, $|\delta(C_i) \setminus E_{\mathrm{OPT}}| \leq k_i - 1 \leq 1/2|\delta(C_i)|$. Therefore, $|E_i^*| \geq 1/2|\delta(C_i)|$. Now, since any two sets C_i and C_j in $\mathcal{C}'$ are disjoint (Lemma 3.11), any edge e belongs to at most two of the sets $\delta(C_i)$. Therefore, $|\cup E_i^*| \geq 1/2 \sum_i |E_i^*| \geq 1/4 \sum_i |\delta(C_i)|$, or $|E_{\mathrm{OPT}}| \geq 1/4 |\cup_{C_i \in \mathcal{C}'} \delta(C_i)|$. □

Multiway Cut and Multicut with Uniform Costs

Finally we note that the approach taken in Algorithm *SS-EDRC-Uniform* does not work in the case of multiway EDRC or multicut EDRC. In particular, there is a family of instances

of the multiway EDRC parameterized by k, containing $\sqrt{k}$ terminals, such that each pair of terminals is $2k+1$ connected, and yet the size of the minimum multiway cut is a factor of $\sqrt{k}$ larger than the size of the minimum multiway k-EDRC.

The family is described as follows. Let $t_0, \cdots, t_{h-1}$ be the terminals with $h = \sqrt{k}$. There are k parallel edges between t_i and $t_{i+1 \mod h}$ for all $i \in [h]$, and an additional edge for each pair of terminals, for a total of $\Theta(k^{3/2})$ edges. Then any multiway cut must remove all the $\Theta(k^{3/2})$ edges, whereas in order to obtain a multiway k-EDRC, it suffices to remove all parallel edges between t_0 and t_{h-1}, as well as the $O(h^2)$ additional edges, (leaving a "path" from t_0 to t_{h-1},) at a cost of $O(k)$.

3.6 Conclusion and Subsequent Results

In this chapter we formulate a new linear-programming relaxation for the multi-route cut problem and extend the region growing lemma to develop polylogarithmic approximations for the 2-route case. In addition, we develop the first non-trivial bicriteria approximations for the single-source multiple-sink case with arbitrary connectivity thresholds.

After the initial publication of this work, improved approximations and inapproximability results have been developed for the multi-route cut problem. Kolman et al. [KS11] achieved an $O(\log^3 h)$ approximation for the 3-route cut ($k = 3$) case by rounding our LP-relaxation via a multi-level ball growing method. The best known approximation and hardness results for multi-route cuts are obtained by Chuzhoy et al. [CMVZ12]. For arbitrary thresholds, they gave a $\big(2, O(\log^{2.5} h \log\log h)\big)$-bicriteria approximation algorithm with running time $n^{O(k)}$ and a polynomial-time $\big(O(\log r), O(\log^3 r)\big)$-bicriteria approximation. Instead of relying on a LP-relaxation, the overall approach in [CMVZ12] is to iteratively reduce the the connectivity between terminals by finding sparse cuts in the graph. In addition, they establish $\Omega(k^\epsilon)$-hardness for the vertex-connectivity version of the multi-route cut problem,

where $\epsilon > 0$ is some constant.

Finding a polynomial time algorithm that violates the connectivity requirement by a constant factor and achieves a polylogarithmic approximation to cost remains an important open question.

4 Packing Multiway Cuts

4.1 Introduction

We study the *multiway cut packing* problem (MCP) introduced by Rabani, Schulman and Swamy [RSS08]. In this problem, we are given k instances of the multiway cut problem in a common graph, each instance being a set of terminals at different locations in the graph. Informally, our goal is to compute nearly-disjoint multiway cuts for each of the instances. More precisely, we aim to minimize the maximum number of cuts that any single edge in the graph belongs to. In the weighted version of this problem, different edges have different capacities; the goal is to minimize the maximum relative load of any edge, where the relative load of an edge is the ratio of the number of cuts it belongs to and its capacity.

The multiway cut packing problem belongs to the following class of graph labeling problems. We are given a partially labeled set of n items along with a weighted graph over them that encodes similarity information among them. An item's label is a string of length k where each coordinate of the string is either drawn from an alphabet Σ, or is undetermined. Roughly speaking, the goal is to complete the partial labeling in the most consistent possible way. Note that completing a single specific entry (coordinate) of each item label is like finding what we call a "set multiway cut"—for $\sigma \in \Sigma$ let S^i_σ denote the set of nodes for which the ith coordinate is labeled σ in the partial labeling, then a complete and consistent labeling for this coordinate is a partition of the items into $|\Sigma|$ parts such that the σ^{th} part contains the entire set S^i_σ. The cost of the labeling for a single pair of neighboring items in the graph is measured by the Hamming distance between the labels assigned to them. The overall cost of the labeling can then be formalized as a certain norm of the vector of (weighted) edge costs.

Different choices of norms for the overall cost give rise to different objectives. Minimizing

the ℓ_1 norm, for example, is the same as minimizing the sum of the edge costs. This problem decomposes into finding k minimum set multiway cuts. Each set multiway cut instance can be reduced to a minimum multiway cut instance by simply merging all the items in the same set S_σ into a single node in the graph, and can therefore be approximated to within a factor of 1.5 [CKR00]. On the other hand, minimizing the ℓ_∞ norm of edge costs (equivalently, the maximum edge cost) becomes the set multiway cut packing problem. Formally, in this problem, we are given k set multiway cut instances $S^1, \cdots, S^k$, where each $S^i = S^i_1 \times S^i_2 \times \cdots \times S^i_{|\Sigma|}$. The goal is to find k cuts, with the ith cut separating every pair of terminals that belong to sets $S^i_{j_1}$ and $S^i_{j_2}$ with $j_1 \neq j_2$, such that the maximum (weighted) cost of any edge is minimized. When $|S^i_j| = 1$ for all $i \in [k]$ and $j \in \Sigma$, this is the multiway cut packing problem.

Note that, unlike the ℓ_1-norm-minimization case, we do not get an approximation preserving reduction from set multiway cut packing to multiway cut packing by merging nodes with the same attribute values. Roughly speaking, if nodes u and v have the same ith attribute, and nodes v and w have the same jth attribute, then this approach merges all three nodes, although an optimal solution may end up separating u from w in some of the cuts. We are not aware of any other approximation preserving reduction between the two problems; therefore finding constant-factor approximation algorithms for the set multiway cut packing problem remains an interesting open question.

To our knowledge Rabani et al. [RSS08] were the first to consider the multiway cut packing problem and provide approximation algorithms for it. They used a linear programming relaxation of the problem along with randomized rounding to obtain an $O(\frac{\log n}{\log \log n})$ approximation, where n is the number of nodes in the given graph. This approximation ratio arises from an application of the Chernoff bounds to the randomized rounding process, and improves to an $O(1)$ factor when the optimal load is $\Omega(\log n)$. When the underlying graph is a tree, Rabani et al. use a more careful deterministic rounding technique to obtain

an improved $O(\log^2 k)$ approximation. The latter approximation factor holds also for a more general multicut packing problem (described in more detail below). One nice property of the $O(\log^2 k)$ approximation by Rabani et al. [RSS08] over trees is that it is independent of the size of the graph, and remains small as the graph grows but k remains fixed. Then, a natural open problem related to their work is whether a similar approximation guarantee independent of n can be obtained even for general graphs.

4.1.1 Results and Techniques

We answer this question in the positive. We employ the same linear programming relaxation for this problem as Rabani et al., but develop a very different rounding algorithm. In order to produce a good integral solution our rounding algorithm requires a fractional collection of cuts that is not only feasible for the linear program but also satisfies an additional good property—the cut collection is laminar. In other words, when interpreted appropriately as subsets of nodes, no two cuts in the collection "cross" each other. Given such an input the rounding process only incurs a small additive loss in performance—the final (absolute) load on any edge is at most 3 more than the load on that edge of the fractional solution that we started out with. Of course the laminarity condition comes at a cost – not every fractional solution to the cut packing LP can be interpreted as a laminar collection of cuts (see, e.g., Figure 4.10). We show that for the multiway cut problem any fractional collection of cuts can be converted into a laminar one while losing only a multiplicative factor of 8 and an additive $o(1)$ amount in edge loads. Therefore, for every edge e we obtain a final edge load of $8\ell_e^{\mathrm{OPT}} + 4$, where ℓ_e^{OPT} is the optimal load on the edge. We only load edges with $c_e \geq 1$ and since the optimal cost is at least 1 our algorithm also obtains a purely multiplicative 12 approximation.

Our laminarity based approach proves even more powerful in the special case of *common-*

sink s-t cut packing problem or CSCP. In this special case every multiway cut instance has only two terminals and all the instances share a common sink t. We use these properties to improve both the rounding and laminarity transformation algorithms, and ensure a final load of at most $\ell_e^{\mathrm{OPT}} + 1$ for every edge e. The CSCP is NP-Hard (see Section 4.3) and so our guarantee for this special case is the best possible.

In converting a fractional laminar solution to an integral one we use an iterative rounding approach, assigning an integral cut at each iteration to an appropriate "innermost" terminal. Throughout the algorithm we maintain a partial integral cut collection and a partial fractional one and ensure that these collections together are feasible for the given multiway cut instances. As we round cuts, we "shift" or modify other fractional cuts so as to maintain bounds on edge loads. Maintaining feasibility and edge loads simultaneously turns out to be relatively straightforward in the case of common-sink s-t cut packing – we only need to ensure that none of the cuts in the fractional or the integral collection contain the common sink t. However in the general case we must ensure that new fractional cuts assigned to any terminal must exclude all other terminals of the same multiway cut instance. This requires a more careful reassignment of cuts.

4.1.2 Related Work

Problems falling under the general framework of graph labeling as described above have been studied in various guises. The most extensively studied special case, called label extension, involves partial labelings in which every item is either completely labeled or not labeled at all. When the objective is to minimize the ℓ_1 norm of edge costs, this becomes a special case of the metric labeling and 0-extension problems [KT02, CKR04, CKNZ04, Kar98]. (The main difference between 0-extension and the label extension problem as described above is that the cost of the labeling in the former arises from an arbitrary metric over the labels,

while in the latter it arises from the Hamming metric.)

When the underlying graph is a tree and edge costs are given by the edit distance between the corresponding labels, this is known as the tree alignment problem. The tree alignment problem has been studied widely in the computational biology literature and arises in the context of labeling phylogenies and evolutionary trees. This version is also NP-Hard, and there are several PTASes known [WJL96, WJG00, WG97]. Ravi and Kececioglu [RJ98] also introduced and studied the ℓ_∞ version of this problem, calling it the bottleneck tree alignment problem. They presented an $O(\log n)$ approximation for this problem. A further special case of the label extension problem under the ℓ_∞ objective, where the underlying tree is a star with labeled leaves, is known as the closest string problem. This problem is also NP-Hard but admits a PTAS [LMW02].

As mentioned above, the multiway cut packing problem was introduced by Rabani, Schulman and Swamy [RSS08]. Rabani et al. also studied the more general multicut packing problem (where the goal is to pack multicuts so as to minimize the maximum edge load) as well as the label extension problem with the ℓ_∞ objective. Rabani et al. developed an $O(\log^2 k)$ approximation for multicut packing in trees, and an $O(\log M \frac{\log n}{\log \log n})$ in general graphs. Here M is the maximum number of terminals in any one multicut instance. For the label extension problem they presented a constant factor approximation in trees, which holds even when edge costs are given by a fairly general class of metrics over the label set (including Hamming distance as well as edit distance).

Another line of research loosely related to the cut packing problems described here considers the problem of finding the largest collection of edge-disjoint cuts (not corresponding to any specific terminals) in a given graph. While this problem can be solved exactly in polynomial time in directed graphs [LY78], it is NP-Hard in undirected graphs, and Caprara, Panconesi and Rizzi [CPR04] presented a 2 approximation for it. In terms of approximability, this problem is very different from the one we study—in the former, the goal is to find as

many cuts as possible, such that the load on any edge is at most 1, whereas in our setting, the goal is to find cuts for all the commodities, so that the maximum edge load is minimized.

We give formal definitions and statements of our results along with a linear programming relaxation for the cut packing problem in Section 4.2. Section 4.4 presents a rounding algorithm which transforms a fractional collection of cuts (i.e, cuts with fractional weights associated with them) that is feasible for the linear programming relaxation into a feasible integral collection of cuts. The algorithm incurs a small additive loss in performance, but requires that no two cuts in the given fractional collection "cross" each other. That is, the input to the rounding algorithm should be a *laminar* cut family. In Section 4.5 we complete the approximation scheme by developing an algorithm which converts any fractional collection of multiway cuts into a laminar one, while losing only a constant factor in edge loads.

4.2 Problem Definitions and Results

Given a graph $G = (V, E)$, a *cut* in G is a subset of edges E', the removal of which disconnects the graph into multiple connected components. A *vertex partition* of G is a pair $(C, V \setminus C)$ with $\emptyset \subsetneq C \subsetneq V$. For a set C with $\emptyset \subsetneq C \subsetneq V$, we use $\delta(C)$ to denote the cut defined by C, that is, $\delta(C) = \{(u, v) \in E : |C \cap \{u, v\}| = 1\}$. We say that a cut $E' \subseteq E$ separates vertices u and v if u and v lie in different connected components in $(V, E \setminus E')$. The vertex partition defined by set C separates u and v if the two vertices are separated by the cut $\delta(C)$. Given a collection of cuts $\mathrm{E} = \{E_1, \cdots, E_k\}$ and capacities c_e on edges, the load ℓ_e^{E} on an edge e is defined as the number of cuts that contain e, that is, $\ell_e^{\mathrm{E}} = |\{E_i \in \mathrm{E} | e \in E_i\}|$. Likewise, given a collection of vertex partitions $\mathcal{C} = \{C_1, \cdots, C_k\}$, the load $\ell_e^{\mathcal{C}}$ on an edge e is defined to be the load of the cut collection $\{\delta(C_1), \cdots, \delta(C_k)\}$ on e.

The input to a *multiway cut packing* problem (MCP) is a graph $G = (V, E)$ with non-zero integral capacities c_e on edges, and k sets $S_1, \cdots, S_k$ of terminals (called "commodities"); each terminal $i \in S_a$ resides at a vertex r_i in V. The goal is to produce a collection of cuts $\mathrm{E} = \{E_1, \cdots, E_k\}$, such that (1) for all $a \in [k]$, and for all pairs of terminals $i, j \in S_a$, the cut E_a separates r_i and r_j, and (2) the maximum "relative load" on any edge, $\max_e \ell_e^{\mathrm{E}}/c_e$, is minimized.

In a special case of this problem called the *common-sink s-t cut packing* problem (CSCP), the graph G contains a special node t called the sink and each commodity set has exactly two terminals, one of which resides at t. Again the goal is to produce a collection of cuts, one for each commodity such that the maximum relative edge load is minimized.

Both of these problems are NP-Hard to solve optimally (see Section 4.3), and we present LP-rounding based approximation algorithms for them. We assume without loss of generality that the optimal solution has a relative load of 1. The integer program **MCP-IP** below encodes the set of solutions to the MCP with relative load 1.

Here $\mathcal{P}_a$ denotes the set of all paths between any two vertices r_i, r_j with $i, j \in S_a$, $i \neq j$. In order to be able to solve this program efficiently, we relax the final constraint to $x_{a,e} \in [0, 1]$ for all $a \in [k]$ and $e \in E$. Although the resulting linear program has an exponential number of constraints, it can be solved efficiently; in particular, the polynomial-size program **MCP-LP** below is equivalent to it. Given a feasible solution to this linear program, our algorithms round it into a feasible integral solution with small load.

$$\sum_{e \in P} x_{a,e} \geq 1 \qquad \forall a \in [k], P \in \mathcal{P}_a \qquad \textbf{(MCP-IP)}$$

$$\sum_{a} x_{a,e} \leq c_e \qquad \forall e \in E$$

$$x_{a,e} \in \{0,1\} \quad \forall a \in [k], e \in E$$

$$d_a(u,v) \leq d_a(u,w) + d_a(w,v) \quad \forall a \in [k],\ u,v,w \in V \qquad \textbf{(MCP-LP)}$$

$$d_a(r_i, r_j) \geq 1 \qquad \forall a \in [k], i,j \in S_a$$

$$\sum_{a} d_a(e) \leq c_e \qquad \forall e \in E$$

$$d_a(e) \in [0,1] \qquad \forall a \in [k], e \in E$$

In the remainder of this chapter we focus exclusively on solutions to the MCP and CSCP that are collections of vertex partitions. This is without loss of generality (up to a factor of 2 in edge loads for the MCP) and allows us to exploit structural properties of vertex sets such as laminarity that help in constructing a good approximation. Accordingly, in the rest of the chapter we use the term "cut" to denote a subset of the vertices that defines a vertex partition.

A pair of cuts $C_1, C_2 \subset V$ is said to "cross" if all of the sets $C_1 \cap C_2$, $C_1 \setminus C_2$, and $C_2 \setminus C_1$ are non-empty. A collection $\mathcal{C} = \{C_1, \cdots, C_k\}$ of cuts is said to be *laminar* if no pair of cuts $C_i, C_j \in \mathcal{C}$ crosses. All of our algorithms are based on the observation that both the MCP and the CSCP admit near-optimal solutions that are laminar. Specifically, there is a polynomial-time algorithm that given a fractional feasible solution to MCP or CSCP (i.e. a

feasible solution to **MCP-LP**) produces a laminar family of fractional cuts that is feasible for the respective problem and has small load. This is formalized in Lemmas 4.1 and 4.2 below. We first introduce the notion of a fractional laminar family of cuts.

Definition 4.1. *A fractional laminar cut family $\mathcal{C}$ for terminal set T with weight function w is a collection of cuts with the following properties:*

- *The collection is laminar*
- *Each cut C in the family is associated with a unique terminal in T. We use $\mathcal{C}_i$ to denote the sub-collection of sets associated with terminal $i \in T$. Every $C \in \mathcal{C}_i$ contains the node r_i.*
- *For all $i \in T$, the total weight of cuts in $\mathcal{C}_i$, $\sum_{C \in \mathcal{C}_i} w(C)$, is 1.*

Next we define what it means for a fractional laminar family to be feasible for the MCP or the CSCP. Note that for a terminal pair $i \neq j$ belonging to the same commodity, condition (2) below is weaker than requiring cuts in *both* C_i and C_j to separate r_i from r_j.

Definition 4.2. *A fractional laminar family of cuts $\mathcal{C}$ for terminal set T with weight function w is feasible for the MCP on a graph G with edge capacities c_e and commodities $S_1, \cdots, S_k$ if (1) $T = \cup_{a \in [k]} S_a$, (2) for all $a \in [k]$ and $i, j \in S_a$, $i \neq j$, either $r_j \notin \cup_{C \in \mathcal{C}_i} C$, or $r_i \notin \cup_{C \in \mathcal{C}_j} C$, and (3) for every edge $e \in E$, $\ell_e^{\mathcal{C}} \leq c_e$.*

The family is feasible for the CSCP on a graph G with edge capacities c_e and commodities $S_1, \cdots, S_k$ if (1) $T = \cup_{a \in [k]} S_a \setminus \{t\}$, (2) $t \notin \cup_{C \in \mathcal{C}} C$, and (3) for every $e \in E$, $\ell_e^{\mathcal{C}} \leq c_e$.

Lemma 4.1. *Consider an instance of the CSCP with graph $G = (V, E)$, common sink t, edge capacities c_e, and commodities $S_1, \cdots, S_k$. Given a feasible solution d to **MCP-LP**, algorithm* Lam-1 *produces a fractional laminar cut family $\mathcal{C}$ that is feasible for the CSCP on G with edge capacities $c_e + o(1)$.*

Lemma 4.2. *Consider an instance of the MCP with graph $G = (V, E)$, edge capacities c_e, and commodities $S_1, \cdots, S_k$. Given a feasible solution d to* ***MCP-LP****, algorithm* Lam-2 *produces a fractional laminar cut family $\mathcal{C}$ that is feasible for the MCP on G with edge capacities $8c_e + o(1)$.*

Lemmas 4.1 and 4.2 are proven in Section 4.5. In Section 4.4 we show how to deterministically round a fractional laminar solution to the CSCP and MCP into an integral one while increasing the load on every edge by no more than a small additive amount. These rounding algorithms are the main contributions of our work, and crucially use the laminarity of the fractional solution.

Lemma 4.3. *Given a fractional laminar cut family $\mathcal{C}$ feasible for the CSCP on a graph G with* integral *edge capacities c_e, the algorithm* Round-1 *produces an integral family of cuts $\mathcal{A}$ that is feasible for the CSCP on G with edge capacities $c_e + 1$.*

For the MCP, the rounding algorithm loses an additive factor of 3 in edge load.

Lemma 4.4. *Given a fractional laminar cut family $\mathcal{C}$ feasible for the MCP on a graph G with* integral *edge capacities c_e, the algorithm* Round-2 *produces an integral family of cuts $\mathcal{A}$ that is feasible for the MCP on G with edge capacities $c_e + 3$.*

Combining these lemmas together we obtain the following theorem.

Theorem 4.1. *There exists a polynomial-time algorithm that given an instance of the MCP with graph $G = (V, E)$, edge capacities c_e, and commodities $S_1, \cdots, S_k$, produces a family $\mathcal{A}$ of multiway cuts, one for each commodity, such that for each $e \in E$, $\ell_e^{\mathcal{A}} \leq 8c_e + 4$.*

There exists a polynomial-time algorithm that given an instance of the CSCP with graph $G = (V, E)$, edge capacities c_e, and commodities $S_1, \cdots, S_k$, produces a family $\mathcal{A}$ of multiway cuts, one for each commodity, such that for each $e \in E$, $\ell_e^{\mathcal{A}} \leq c_e + 2$.

4.3 NP-Hardness

We will now prove that CSCP and MCP are NP-Hard. Since edge loads for any feasible solution to these problems are integral, the result of Theorem 4.1 is optimal for the CSCP assuming P≠NP. The reduction in this theorem also gives us an integrality gap instance for the CSCP.

Theorem 4.2. *CSCP and MCP are* NP*-Hard. Furthermore the integrality gap of* ***MCP-LP*** *is at least* 2 *for both the problems.*

Proof. We reduce independent set to CSCP. In particular, given a graph G and a target k, we produce an instance of CSCP such that the load on every edge is at most 1 if and only if G contains an independent set of size at least k. Let n be the number of vertices in G. We construct G' by adding a chain of $n-k+1$ new vertices to G. Let the first vertex in this chain be t (the common sink) and the last be v. We connect every vertex of G to the new vertex v, and place a terminal i at every vertex r_i in G (therefore, there are a total of n sources). We claim that there is a collection of n edge-disjoint $r_i - t$ cuts in this new graph G' if and only if G contains an independent set of size k.

One direction of the proof is straightforward: if G contains an independent set of size k, say S, then for each vertex $r_i \in S$, consider the cut $\{r_i\}$, and for each of the $n-k$ source not in S, consider the cuts obtained by removing one of the $n-k$ chain edges in G'. Then all of these n cuts are edge-disjoint.

Next suppose that G' contains a collection of edge-disjoint cuts C_i, with $r_i \in C_i$ and $t \notin C_i$ for all i. Note that the number of cuts C_i containing any chain vertex is at most $n-k$ because each of them cuts at least one chain edge. Next consider the cuts that do not contain any chain vertex, specifically v, and let T' be the collection of terminals for such cuts. These are at least k in number. Note that any cut C_i, $i \in T'$, cuts the edges (u, v) for $u \in C_i$. Therefore, in order for these cuts to be edge-disjoint, it must be the case

that $C_i \cap C_j = \emptyset$ for $i, j \in T'$, $i \neq j$. Finally, for two such cuts C_i and C_j, edge-disjointness again implies that r_i and r_j are not connected. Therefore the vertices r_i for $i \in T'$ form an independent set in G of size at least k.

For the integrality gap, let G be the complete graph and k be $n/2$. Then, there is no integral solution with load 1 in G'. However, the following fractional solution is feasible and has a load of 1: let the chain of vertices added to G be $v = v_1, v_2, \cdots, v_{n/2+1} = t$; assign to every terminal i, $i \in [n]$, the cut $\{r_i\}$ with weight $1/2$, and the cut $V \cup \{v_0, \cdots, v_{\lfloor i/2 \rfloor}\}$ with weight $1/2$. □

4.4 Rounding Fractional Laminar Cut Families

In this section we develop algorithms for rounding feasible fractional laminar solutions to the MCP and the CSCP to integral ones while increasing edge loads by a small additive amount. We first demonstrate some key ideas behind the algorithm and the analysis for the CSCP, and then extend them to the more general case of multiway cuts. Throughout the section we assume that the edge capacities c_e are integral.

4.4.1 The Common Sink Case (proof of Lemma 4.3)

Our rounding algorithm for the CSCP rounds fractional cuts roughly in the order of innermost cuts first. The notion of an innermost terminal is defined with respect to the fractional solution. After each iteration we ensure that the remaining fractional solution continues to be feasible for the unassigned terminals and has small edge loads. We use $\mathcal{C}$ to denote the fractional laminar cut family that we start out with and $\mathcal{A}$ to denote the integral family that we construct. Recall that for an edge $e \in E$, $\ell_e^{\mathcal{C}}$ denotes the load of the fractional cut family $\mathcal{C}$ on e, and $\ell_e^{\mathcal{A}}$ denotes the load of the integral cut family $\mathcal{A}$ on e. We call the former the fractional load on the edge, and the latter its integral load.

We now formalize what we mean by an "innermost" terminal. For every vertex $v \in V$, let K_v denote the set of cuts in $\mathcal{C}$ that contain v. The "depth" of a vertex v is the total weight of all cuts in K_v: $d_v = \sum_{C \in K_v} w(C)$. The depth of a terminal is defined as the depth of the vertex at which it resides. Terminals are picked in order of decreasing depth.

Before we describe the algorithm we need some more notation. At any point during the algorithm we use S_e to denote the set of cuts crossing an edge e. As the algorithm proceeds, the integral loads on edges increase while their fractional loads decrease. Whenever the fractional load of an edge becomes 0, we merge its end-points to form "meta-nodes". At any point of time, we use $M(v)$ to denote the meta-node containing a node $v \in V$.

Finally, for a set of fractional cuts $L = \{L_1, \cdots, L_l\}$ with $L_1 \subseteq L_2 \subseteq \cdots \subseteq L_l$ and weight function w, we use L^x to denote the subset of L containing the innermost cuts with weight exactly x. That is, let l' be such that $\sum_{a<l'} w(L_a) < x$ and $\sum_{a \leq l'} w(L_a) \geq x$. Then L^x is the set $\{L_1, \cdots, L_{l'}\}$ with weight function w' such that $w'(L_a) = w(L_a)$ for $a < l'$ and $w'(L_{l'}) = x - \sum_{a<l'} w(L_a)$.

The algorithm *Round-1* is given in Figure 4.1. At every step, the algorithm picks a terminal, say i, with the maximum depth and assigns an integral cut to it. This potentially frees up capacity used up by the fractional cuts of i, but may use up extra capacity on some edges that was previously occupied by fractional cuts belonging to other terminals. In order to avoid increasing edge loads, we reassign to terminals in the latter set, fractional cuts of i that have been freed up.

Our analysis has two parts. Lemma 4.5 shows that the family $\mathcal{C}$ continues to remain feasible, that is it always satisfy the first two conditions in Definition 4.2 for the unassigned terminals. Lemma 4.6 analyzes the total load of the fractional and integral families as the algorithm progresses.

Lemma 4.5. *Throughout the algorithm, the cut family $\mathcal{C}$ is a fractional laminar family for*

Input: Graph $G = (V, E)$ with capacities c_e, terminals T with a fractional laminar cut family $\mathcal{C}$, common sink t with $t \notin \cup_{C \in \mathcal{C}} C$.
Output: A collection of cuts $\mathcal{A}$, one for each terminal in T.

1. Initialize $T' = T$, $\mathcal{A} = \emptyset$, and $M(v) = \{v\}$ for all $v \in V$. Compute the depths of vertices and terminals.
2. While there are terminals in T' do:
   a) Let i be a terminal with the maximum depth in T'. Let $A_i = M(r_i)$. Add A_i to $\mathcal{A}$ and remove i from T'.
   b) Let $K = K^1_{r_i}$. Remove cuts in $K \cap \mathcal{C}_i$ from K, $\mathcal{C}_i$ and $\mathcal{C}$. While there exists a terminal $j \in T'$ with a cut $C \in K \cap \mathcal{C}_j$, do the following: let $w = w(C)$; remove C from K, $\mathcal{C}_j$ and $\mathcal{C}$; remove cuts in $\mathcal{C}^w_i$ from $\mathcal{C}_i$ and add them to $\mathcal{C}_j$ (that is, these cuts are reassigned from terminal i to terminal j).
   c) If there exists an edge $e = (u, v)$ with $\ell^{\mathcal{C}}_e = 0$, merge the meta-nodes $M(u)$ and $M(v)$ (we say that the edge e has been "contracted").
   d) Recompute the depths of vertices and terminals.

Figure 4.1: Algorithm *Round-1*—Rounding algorithm for common-sink s-t cut packing

terminals in T' with $t \notin \cup_{C \in \mathcal{C}} C$.

Proof. We prove this by induction over the iterations of the algorithm. The claim obviously holds at the beginning of the algorithm. Consider a step at which some terminal i is assigned an integral cut. The algorithm removes all the cuts in $K = K^1_{r_i}$ from $\mathcal{C}$. Some of these cuts belong to other terminals; those terminals are reassigned new cuts. Specifically, we first remove cuts in $K \cap \mathcal{C}_i$ from the cut family. The total weight of the remaining cuts in K as well as the total weight of those in $\mathcal{C}_i$ is equal at this time. Subsequently, we successively consider terminals j with a cut $C \in K \cap \mathcal{C}_j$, and let $w = w(C)$. Then we remove C from the cut family, and reassign cuts of total weight w in $\mathcal{C}^w_i$ to j. Therefore, the total weight of cuts assigned to j remains 1. Furthermore, the newly reassigned cuts contain the cut C, and therefore the vertex r_j, but do not contain the sink t. Therefore, $\mathcal{C}$ continues to be a

fractional laminar family for terminals in T'. □

Lemma 4.6. *At any point of time for every edge $e \in E$, $\ell_e^{\mathcal{A}} \leq c_e - 1$ implies $\ell_e^{\mathcal{A}} + \ell_e^{\mathcal{C}} \leq c_e$, $\ell_e^{\mathcal{A}} = c_e$ implies $\ell_e^{\mathcal{C}} \leq 1$, and $\ell_e^{\mathcal{A}} = c_e + 1$ implies $\ell_e^{\mathcal{C}} = 0$. Furthermore, for $e = (u, v)$, $\ell_e^{\mathcal{A}} = c_e$ implies that either $K_u \cap S_e$ or $K_v \cap S_e$ is empty.*

Proof. Let $e = (u, v)$. We prove the lemma by induction over time. Note that in the beginning of the algorithm, we have for all edges $\ell_e^{\mathcal{C}} \leq c_e$ and $\ell_e^{\mathcal{A}} = 0$, so the inequality $\ell_e^{\mathcal{A}} + \ell_e^{\mathcal{C}} \leq c_e$ holds.

Let us now consider a single iteration of the algorithm and suppose that the integral load of the edge increases during this iteration. (If it doesn't increase, since $\ell_e^{\mathcal{C}}$ only decreases over time, the claim continues to hold.) Let i be the commodity picked by the algorithm in this iteration, then $M(r_i)$ is the same as either $M(u)$ or $M(v)$. Without loss of generality assume that $r_i \in M(u)$. Let α denote the total weight of cuts in $K_u \cap S_e$ and β denote the total weight of cuts in $K_v \cap S_e$ prior to this iteration. Then, $\alpha + \beta = \ell_e^{\mathcal{C}}$. Moreover, all cuts in $\mathcal{C} \setminus S_e$ either contain both or neither of u and v. So we can relate the depths of v and u in the following way: $d_v = d_u - \alpha + \beta$. Since i is the terminal picked during this iteration, we must have $d_u \geq d_v$, and therefore, $\alpha \geq \beta$.

We analyze the final edge load depending on the value of α. Two cases arise: suppose first that $\alpha \geq 1$. Then $K_u^1 \subseteq K_u \cap S_e$, and the fractional weight of e reduces by exactly 1. On the other hand, the integral load on the edge increases by 1, and so the total load continues to be the same as before. On the other hand, if $\alpha \leq 1$, then $K_u \cap S_e \subseteq K_u^1$, and all the cuts in $K_u \cap S_e$ get removed from S_e in this iteration. Therefore the final fractional load is at most $\beta \leq \alpha \leq 1$, and at the end of the iteration, $K_u \cap S_e = \emptyset$. If $\ell_e^{\mathcal{A}} \leq c_e - 1$, we immediately get that the total load on the edge is at most c_e.

If $\ell_e^{\mathcal{A}} = c_e$, then prior to this iteration $\ell_e^{\mathcal{A}} = c_e - 1$, and so $\ell_e^{\mathcal{C}} \leq 1$ by the induction hypothesis. Then, as we argued above, $\alpha \leq \ell_e^{\mathcal{C}} \leq 1$ implies that the new fractional load on

the edge is at most 1 and at the end of the iteration, $K_u \cap S_e = \emptyset$.

Finally, if $\ell_e^{\mathcal{A}} = c_e + 1$, then prior to this iteration, $\ell_e^{\mathcal{A}} = c_e$ and by the induction hypothesis, β is zero (as $\alpha \geq \beta$ and either $K_u \cap S_e$ or $K_v \cap S_e$ is empty). Along with the fact that $\alpha \leq 1$ (by the inductive hypothesis), the final fractional load on the edge is $\beta = 0$. □

The two lemmas together give us a proof of Lemma 4.3. We restate the lemma for completeness.

Lemma 4.3. *Given a fractional laminar cut family $\mathcal{C}$ feasible for the CSCP on a graph G with* integral *edge capacities c_e, the algorithm* Round-1 *produces an integral family of cuts $\mathcal{A}$ that is feasible for the CSCP on G with edge capacities $c_e + 1$.*

Proof. First note that for every i, A_i is set to be the meta-node of r_i at some point during the algorithm, which is a subset of every cut in $\mathcal{C}_i$ at that point of time. Then $r_i \in A_i$, and by Lemma 4.5, $t \notin A_i$. Second, for any edge e, its integral load $\ell_e^{\mathcal{A}}$ starts out at being 0 and gradually increases by at most an additive 1 at every step, while its fractional load decreases. Once the fractional load of an edge becomes zero, both its end points belong to the same meta-node, and so the edge never gets loaded again. Therefore, by Lemma 4.6, the maximum integral load on any edge e is at most $c_e + 1$. □

4.4.2 The General Case (proof of Lemma 4.4)

As in the common-sink case, the rounding algorithm for the MCP proceeds by picking terminals according to an order suggested by the fractional solution and assigning the smallest cuts possible to them subject to the availability of capacity on the edges. In the algorithm *Round-1*, we reassign cuts among terminals at every iteration so as to maintain the feasibility of the remaining fractional solution. In the case of MCP, this is not sufficient—a simple reassignment of cuts as in the case of algorithm *Round-1* may not ensure separation

among terminals belonging to the same commodity. We use two ideas to overcome this difficulty: first, among terminals of equal depth, we use a different ordering to pick the next terminal to minimize the need for reassigning cuts; second, instead of reassigning cuts, we modify the existing fractional cuts for unassigned terminals so as to remain feasible while paying a small extra cost in edge load.

We now define the "cut-inclusion" ordering over terminals. For every terminal $i \in T$, let O_i denote the largest (outermost) cut in $\mathcal{C}_i$, that is, $\forall C \in \mathcal{C}_i$, $C \subseteq O_i$. We say that terminal i dominates (or precedes) terminal j in the cut-inclusion ordering, written $i >_{CI} j$, if $O_i \subset O_j$ (if $O_i = O_j$ we break ties arbitrarily but consistently). Cut-inclusion defines a partial order on terminals. Note that we can pre-process the cut family $\mathcal{C}$ by reassigning cuts among terminals, such that for all pairs of terminals $i, j \in T$ with $i >_{CI} j$, and for all cuts $C_i \in \mathcal{C}_i$ and $C_j \in \mathcal{C}_j$ with $r_i, r_j \in C_i \cap C_j$, we have $C_i \subseteq C_j$. We call this property the "inclusion invariant". Ensuring this invariant requires a straightforward pairwise reassignment of cuts among the terminals, and we omit the details. Note that following this reassignment, for every terminal i, the new outermost cut of i, O_i, is the same as or a subset of its original outermost cut.

As the algorithm proceeds we modify the collection $\mathcal{C}$ as well as build up the collection $\mathcal{A}$ of integral cuts A_i for $i \in T$. For example, we may split a cut C into two cuts containing the same nodes as C and with weights summing to that of C. As cuts in $\mathcal{C}$ are modified, their ownership by terminals remains unchanged, and we therefore continue using the same notation for them. Furthermore, if for two cuts C_1 and C_2, we have (for example) $C_1 \subseteq C_2$ at the beginning of the algorithm, this relationship continues to hold throughout the algorithm. This implies that the inclusion invariant continues to hold throughout the algorithm. We ensure that throughout the execution of the algorithm the cut family $\mathcal{C}$ continues to be a fractional laminar family for terminals T'. At any point of time, the depth of a vertex or a terminal, as well as the cut-inclusion ordering is defined with respect to the current

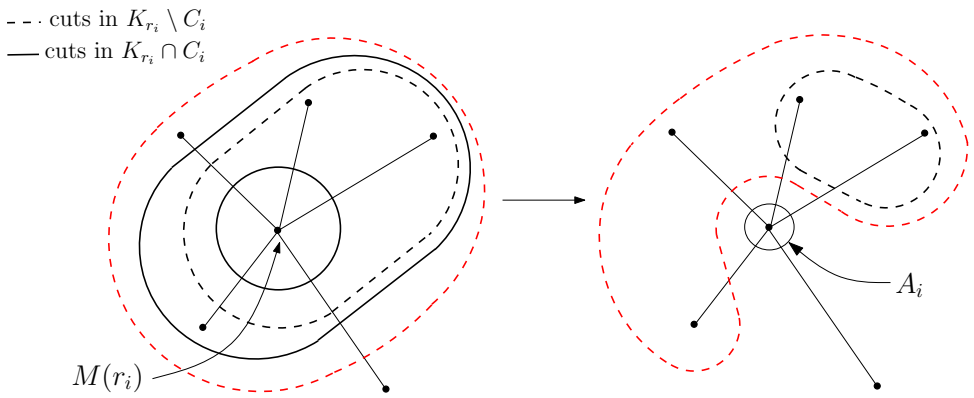

Figure 4.2: An iteration of algorithm *Round-2* (Steps 3b & 3d)

fractional family $\mathcal{C}$.

As before, let S_e denote the set of cuts in $\mathcal{C}$ that cross e — $S_e = \{C \in \mathcal{C} | e \in \delta(C)\}$. Recall that K_v denotes the set of cuts in $\mathcal{C}$ containing the vertex v, and of these K_v^1 denotes the inner-most cuts with total weight exactly 1.

The rounding algorithm is given in Figure 4.3. Roughly speaking, at every step, the algorithm picks a maximum depth terminal i and assigns the cut $M(r_i)$ to it (recall that $M(r_i)$ is the meta-node of the vertex r_i where terminal i resides). It "pays" for this cut using fractional cuts in $K_{r_i}^1$. Of course some of the cuts in $K_{r_i}^1$ belong to other commodities, and need to be replaced with new fractional cuts. The cut-inclusion invariant ensures that these other commodities reside at meta-nodes other than $M(r_i)$, so we modify each cut in $K_{r_i}^1 \setminus C_i$ by removing $M(r_i)$ from it (see Figure 4.2). This process potentially increases the total loads on edges incident on $M(r_i)$ by small amounts, but on no other edges. Step 3c of the algorithm deals with the case in which edges incident on $M(r_i)$ are already overloaded; In this case we avoid loading those edges further by assigning to i some subset of the meta-node $M(r_i)$. Lemmas 4.11 and 4.12 show that this case does not arise too often.

For a terminal i and edge e, if at the time that i is picked in Step 3a of the algorithm e is in $\delta(M(r_i))$, we say that i accesses e. If $e \in E_i$, we say that i defaults on e, and if e is in

$\delta(A_i)$ after this iteration, then we say that i loads e.

During the course of the algorithm integral loads on edges increase, but fractional loads may increase or decrease. To study how these edge loads change during the course of the algorithm, we divide edges into five sets. Let X_{-1} denote the set of edges with $\ell_e^{\mathcal{A}} \leq c_e - 1$ and $\ell_e^{\mathcal{C}} > 0$. For $a \in \{0, 1\}$, let X_a denote the set of edges with $\ell_e^{\mathcal{A}} = c_e + a$ and $\ell_e^{\mathcal{C}} > 0$. Y denotes the set of edges with $\ell_e^{\mathcal{A}} \geq c_e + 2$ and $\ell_e^{\mathcal{C}} > 0$, and Z denotes the set of edges with $\ell_e^{\mathcal{C}} = 0$. Every edge starts out with a zero integral load. As the algorithm proceeds, the edge goes through one or more of the X_as, may enter the set Y, and eventually ends up in the set Z. As for the CSCP, when an edge enters Z, we merge the end-points of the edge into a single meta-node. However, unlike for the CSCP, edges may get loaded even after entering Z. When an edge enters Y, we avoid loading it further (Step 3c), and instead load some edges in Z. Nevertheless, we ensure that edges in Z are loaded no more than once.

As before our analysis has two components. First we show (Lemma 4.7) that the cuts produced by the algorithm are feasible. The following lemmas give the desired guarantees on the edges' final loads: Lemmas 4.8 and 4.9 analyze the loads of edges in X_a for $a \in \{-1, 0, 1\}$; Lemma 4.10 analyzes edges in Y and Lemmas 4.11 and 4.12 analyze edges in Z. We put everything together in the proof of Lemma 4.4 at the end of this section.

Lemma 4.7. *For all i, $r_i \in A_i \subseteq O_i$.*

Proof. Each cut A_i is set equal to the meta-node of r_i at some stage of the algorithm. Therefore, $r_i \in A_i$ for all i. Furthermore, at the time that i is assigned an integral cut, $A_i \subseteq M(r_i) \subseteq O_i$. □

Next we prove some facts about the fractional and integral loads as an edge goes through the sets X_a. The proofs of the following two lemmas are similar to that of Lemma 4.6.

Lemma 4.8. *At any point of time, for every edge $e \in X_{-1}$, $\ell_e^{\mathcal{A}} + \ell_e^{\mathcal{C}} \leq c_e$.*

Input: Graph $G = (V, E)$ with capacities c_e on edges, a set of terminals T with a fractional laminar cut family $\mathcal{C}$.
Output: A collection of cuts $\mathcal{A}$, one for each terminal in T.

1. Preprocess the family $\mathcal{C}$ so that it satisfies the inclusion invariant.
2. Initialize $T' = T$, $\mathcal{A} = \emptyset$, $Y, Z = \emptyset$, and $M(v) = \{v\}$ for all $v \in V$.
3. While there are terminals in T' do:
   a) Consider the set of unassigned terminals with the maximum depth, and of these let $i \in T'$ be a terminal that is undominated in the cut inclusion ordering. Let $E_i = Y \cap \delta(M(r_i))$.
   b) If $E_i = \emptyset$, let $A_i = M(r_i)$.
   c) If $E_i \neq \emptyset$ (we say that the terminal has "defaulted" on edges in E_i), let U_i denote the set of end-points of edges in E_i that lie in $M(r_i)$. If $r_i \in U_i$, abort and return error. Otherwise, consider the vertex in U_i that entered $M(r_i)$ first during the algorithm's execution, call this vertex u_i. Set A_i to be the meta-node of r_i just prior to the iteration where $M(u_i)$ becomes equal to $M(r_i)$.
   d) Add A_i to $\mathcal{A}$. Remove $\mathcal{C}_i$ from $\mathcal{C}$ and i from T'. For every $j \in T'$ and $C \in K^1_{r_i} \cap \mathcal{C}_j$, let $C = C \setminus \{M(r_i)\}$.
   e) If for some edge e, $\ell_e^{\mathcal{A}} = c_e + 2$ and $\ell_e^{\mathcal{C}} > 0$, add e to Y. If there exists an edge $e = (u, v)$ with $\ell_e^{\mathcal{C}} = 0$, merge the meta-nodes $M(u)$ and $M(v)$ (we say that the edge e has been "contracted".) Add all edges e with $\ell_e^{\mathcal{C}} = 0$ to Z and remove them from Y.
   f) Recompute the depths of vertices and terminals.

Figure 4.3: Algorithm *Round-2*—Rounding algorithm for multiway cut packing

Proof. We prove the claim by induction over time. Note that in the beginning of the algorithm, we have for all edges $\ell_e^{\mathcal{C}} \leq c_e$ and $\ell_e^{\mathcal{A}} = 0$, so the inequality $\ell_e^{\mathcal{A}} + \ell_e^{\mathcal{C}} \leq c_e$ holds.

Let us now consider a single iteration of the algorithm and suppose that the edge e remains in the set X_{-1} after this step. There are three events that influence the load of the edge $e = (u, v)$: (1) a terminal at some vertex in $M(u)$ accesses e; (2) a terminal at $M(v)$ accesses e; and, (3) a terminal at some other meta-node $M \neq M(u), M(v)$ is assigned an integral cut. Let us consider the third case first, and suppose that a terminal i is assigned.

Since $A_i \subseteq M$ and therefore $e \notin \delta(A_i)$ its integral load does not increase. However, in the event that $S_e \cap \mathcal{C}_i$ is non-empty, the fractional load on e may decrease (because cuts in $\mathcal{C}_i$ are removed from $\mathcal{C}$). Therefore, the inequality continues to hold.

Next we consider the case where a terminal, say i, with $r_i \in M(u)$ accesses e (the second case is similar). Note that $M(r_i) = M(u)$. In this case the integral load of the edge e potentially increases by 1 (if the terminal loads the edge). By the definition of X_{-1}, the new integral load on this edge is no more than $c_e - 1$. The fractional load on e changes in three ways:

- Cuts in $\mathcal{C}_i \cap S_e$ are removed from $\mathcal{C}$, decreasing $\ell_e^{\mathcal{C}}$.
- Some of the cuts in $(K_{r_i}^1 \setminus \mathcal{C}_i) \setminus S_e$ get "shifted" on to e increasing $\ell_e^{\mathcal{C}}$ (we remove the meta-node $M(r_i)$ from these cuts, and they may continue to contain $M(v)$).
- Cuts in $(K_{r_i}^1 \setminus \mathcal{C}_i) \cap S_e$ get shifted off from e decreasing $\ell_e^{\mathcal{C}}$ (these cuts initially contain $M(r_i)$ but not $M(v)$, and during this step we remove $M(r_i)$ from these cuts).

So the decrease in $\ell_e^{\mathcal{C}}$ is at least the total weight of $K_{r_i}^1 \cap S_e = K_u^1 \cap S_e$, whereas the increase is at most the total weight of $K_{r_i}^1 \setminus S_e = K_u^1 \setminus S_e$.

In order to account for the two terms, let α denote the total weight of cuts in $K_u \cap S_e$, and β denote the total weight of cuts in $K_v \cap S_e$. Then, $\alpha + \beta = \ell_e^{\mathcal{C}}$. As in the proof of Lemma 4.6, we have $d_v = d_u - \alpha + \beta$, and therefore $d_u \geq d_v$ implies $\alpha \geq \beta$. Now, suppose that $\alpha \geq 1$. Then $K_u^1 \subseteq S_e$. Therefore, the decrease in $\ell_e^{\mathcal{C}}$ due to the sets $K_u^1 \cap S_e = K_u^1$ is at least 1, and there is no corresponding increase, so the sum $\ell_e^{\mathcal{A}} + \ell_e^{\mathcal{C}}$ remains at most c_e.

Finally, suppose that $\alpha < 1$. Then K_u^1 contains all the cuts in $K_u \cap S_e$, the weight of $K_u^1 \cap S_e$ is exactly α, and so the decrease in $\ell_e^{\mathcal{C}}$ is at least α. Moreover, the total weight of $K_u^1 \setminus S_e$ is $1 - \alpha$, therefore, the increase in $\ell_e^{\mathcal{C}}$ due to the sets in $K_u^1 \setminus S_e$ is at most $1 - \alpha$.

Since $\ell_e^{\mathcal{C}}$ starts out as being equal to $\alpha + \beta$, its final value after this step is $1 - \alpha + \beta \leq 1$ as $\beta \leq \alpha$. Noting that $\ell_e^{\mathcal{A}}$ is at most $c_e - 1$ after the step, we get the desired inequality. □

Lemma 4.9. *For any edge $e = (u, v)$, from the time that e enters X_0 to the time that it exits X_1, $\ell_e^{\mathcal{C}} \leq 1$. Furthermore suppose (without loss of generality) that during this time in some iteration e is accessed by a terminal i with $r_i \in M(u)$, then following this iteration until the next time that e is accessed, we have $S_e \cap K_u = \emptyset$, and the next access to e (if any) is from a terminal in $M(v)$.*

Proof. First we note that if the lemma holds the first time an edge $e = (u, v)$ enters a set X_a, $a \in \{0, 1\}$, then it continues to hold while the edge remains in X_a. This is because during this time the integral load on the edge does not increase, and therefore throughout this time we assign integral cuts to terminals at meta-nodes different from $M(u)$ and $M(v)$ — this only reduces the fractional load on the edge e and shrinks the set S_e.

Consider the first time that an edge $e = (u, v)$ moves from the set X_{-1} to X_0. Suppose that at this step we assign an integral cut to a terminal i residing at node $r_i \in M(u)$. Prior to this step, $\ell_e^{\mathcal{A}} = c_e - 1$, and so by Lemma 4.8, $\ell_e^{\mathcal{C}} \leq 1$. As before define α to be the total weight of cuts $K_u \cap S_e$, and β to be the total weight of cuts $K_v \cap S_e$. Then following the same argument as in the proof of Lemma 4.8, we conclude that the final fractional weight on e is at most $\beta + 1 - \alpha \leq 1$. Furthermore, since $K_u \cap S_e \subseteq K_u^1$, we either remove all these cuts from $\mathcal{C}$ or shift them off of edge e. Moreover, any new cuts that we shift on to e do not contain the meta-node $M(r_i) = M(u)$, and in particular do not contain the vertex u. Therefore at the end of this step, $S_e \cap K_u = \emptyset$. This also implies that following this iteration terminals in $M(v)$ have depth larger than terminals in $M(u)$, and so the next access to e must be from a terminal in $M(v)$.

The same argument works when an edge moves from X_0 to X_1. We again make use of the fact that prior to the step the fractional load on the edge is at most 1. □

Lemma 4.10. *During any iteration of the algorithm, for any edge $e \in Y$, the following are satisfied:*

- $\ell_e^{\mathcal{C}} \leq 1$
- *If the edge $e = (u, v)$ is accessed by a terminal i with $r_i \in M(u)$, then following this iteration until the next time that e is accessed, we have $S_e \cap K_u = \emptyset$, and the next access to e (if any) is from a terminal in $M(v)$.*
- *If a terminal i with $r_i \in M(u)$ accesses $e = (u, v)$, then $r_i \neq u$, $A_i \cap \{u, v\} = \emptyset$, and so i does not load e. Also, consider any previous access to the edge by a terminal in $M(u)$; then prior to this access, $r_i \notin M(u)$.*

Proof. The first two parts of this lemma extend Lemma 4.9 to the case of $e \in Y$, and are otherwise identical to that lemma. The proof for these claims is analogous to the proof of Lemma 4.9. The only difference is that terminals accessing an edge $e \in Y$ default on this edge. However, this does not affect the argument: when a terminal defaults on the edge, the edge's fractional load changes in the same way as if the terminal did not default; the only change is in the way an integral cut is assigned to the terminal. Since these claims depend only on how the fractional load on the edge changes, they continue to hold while the edge is in Y.

For the third part of the lemma, since $A_i \subseteq M(r_i) = M(u)$ and $v \notin M(u)$, $v \notin A_i$. Next we show that $u \notin A_i$. Consider the iterations of the algorithm during which $\ell_e^{\mathcal{C}} \leq 1$. During this time the edge was accessed at least twice prior to being accessed by i (once when e moved from X_0 to X_1, once when e moved from X_1 to Y, and possibly multiple times while $e \in Y$). Let the last two accesses be by the terminals j_1 and j_2, at iterations t_1 and t_2, $t_1 \leq t_2$. For $a \in \{0, 1\}$, let $M^a(u)$ and $M^a(v)$ denote the meta-nodes of u and v respectively just prior to iteration t_a, and $M(u)$ and $M(v)$ denote the respective meta-nodes

just prior to the current iteration. Then by Lemma 4.9 and the second part of this lemma, we have $r_{j_1} \in M^1(u)$ and $r_{j_2} \in M^2(v)$. We claim that $i >_{CI} j_2 >_{CI} j_1$. Given this claim, if $r_i \in M^1(u) = M^1(r_{j_1})$, then since i and j_1 have the same depth at iteration t_1, we get a contradiction to the fact that the algorithm picks j_1 before i in Step 3a. Therefore, $r_i \notin M(u)$ at any iteration prior to t_1, and in particular, $r_i \neq u$. Finally, since $u \in U_i$ and $U_i \cap A_i = \emptyset$, this also implies that $u \notin A_i$.

It remains to prove the claim. We will prove that $j_2 >_{CI} j_1$. The proof for $i >_{CI} j_2$ is analogous. In fact we will prove a stronger statement: between iterations t_1 and t_2, all terminals with cuts in S_e dominate j_1 in the cut-inclusion ordering. We prove this by induction. By Lemma 4.9, prior to iteration t_1, S_e does not contain any cuts belonging to terminals at $M(v)$. Following the iteration, S_e only contains fractional cuts in K^1_u that got shifted on to the edge e. Prior to shifting, these cuts contain $M^1(u)$, and therefore r_{j_1}, but do not belong to j_1. Then, these cuts are subsets of O_{j_1}, and so by the inclusion invariant, they belong to terminals dominating j_1 in the cut-inclusion ordering. Therefore, the claim holds right after the iteration t_1. Finally, following the iteration until the next time that e is accessed (by j_2), the set S_e only shrinks, and so the claim continues to hold. □

In order to analyze the loading of edges in Z, we need some more notation. Let $\mathcal{M}$ denote the collection of sets of vertices that were meta-nodes at some point during the algorithm. For any edge $e \in Z$, let M_e denote the meta-node formed when e enters Z; then M_e is the smallest set in $\mathcal{M}$ containing both the end points of e. Note that the collection $\mathcal{A} \cup \mathcal{M}$ is laminar.

Lemma 4.11. *An edge $e \in Z$ is loaded only if after the formation of M_e a terminal residing at a vertex in M_e defaults on an edge in $\delta(M_e)$. (Note that this may happen after M_e has merged with some other meta-nodes.)*

Proof. Let i be a defaulting terminal that loads the edge $e \in Z$. Then $e \in \delta(A_i)$, and therefore, $A_i \subsetneq M_e$ and $r_i \in M_e$. Furthermore, since A_i is a strict subset of M_e, $U_i \cap M_e \neq \emptyset$, and therefore, i defaults on an edge $e' \in Y$ with at least one end-point in M_e. But if both the end-points of e' are in M_e, then we must have $\ell^{\mathcal{C}}_{e'} = 0$ contradicting the fact that e' is in Y. Therefore, $e' \in \delta(M_e)$. □

Lemma 4.12. *For any meta-node $M \in \mathcal{M}$, after its formation, at most one terminal residing at a vertex in M can default on edges in $\delta(M)$ (even after M has merged with other meta-nodes).*

Proof. For the sake of contradiction, suppose that two terminals i and j, both residing at vertices in M default on edges in $\delta(M)$ after the formation of M, with i defaulting before j. Let M_1 (M_2) denote the meta-node containing M just before i (j) defaulted. Note that $M \subseteq M_1 \subseteq M_2$. Consider an edge $e \in E_j \cap \delta(M)$ (recall that E_j is the set of edges that j defaults on, so this set is non-empty by our assumption). Then $e \in \delta(M) \cap \delta(M_2) \subseteq \delta(M_1)$. Therefore, at the time that i defaulted, e was accessed by i, and by the third claim in Lemma 4.10, $r_j \notin M_1$. This contradicts the fact that $r_j \in M$. □

Finally we can put all these lemmas together to prove our main result on algorithm *Round-2*.

Lemma 4.13. *Given a fractional laminar cut family $\mathcal{C}$ feasible for the MCP on a graph G with* integral *edge capacities c_e, the algorithm* Round-2 *produces an integral family of cuts $\mathcal{A}$ that is feasible for the MCP on G with edge capacities $c_e + 3$.*

Proof. We first note that the third part of Lemma 4.10 implies that for all i, $r_i \notin U_i$, and therefore the algorithm never aborts. Then Lemma 4.7 implies that we get a feasible cut packing. Finally, note that every edge starts out in the set X_{-1}, goes through one or more of the X_a's, $a \in \{0, 1\}$, potentially goes through Y, and ends up in Z. An edge e enters

Input: Graph $G = (V, E)$ with edge capacities c_e, commodities $S_1, \cdots, S_k$, common sink t, a feasible solution d to the program **MCP-LP**.
Output: A fractional laminar family of cuts $\mathcal{C}$ that is feasible for G with edge capacities $c_e + o(1)$.

1. For every $a \in [k]$ and terminal $i \in S_a$ do the following: Order the vertices in G in increasing order of their distance under d_a from r_i. Let this ordering be $v_0 = r_i, v_1, \cdots, v_n$. Let $\mathcal{C}_i$ be the collection of cuts $\{v_0, v_1, \cdots, v_b\}$, one for each $b \in [n]$, $d_a(r_i, v_b) < 1$, with weights $w(\{v_0, \cdots, v_b\}) = d_a(r_i, v_{b+1}) - d_a(r_i, v_b)$. Let $\mathcal{C}$ denote the collection $\{\mathcal{C}_i\}_{i \in \cup_a S_a}$.
2. Let $N = nk$. Round up the weights of all the cuts in $\mathcal{C}$ to multiples of $1/N^2$, and truncate the collection so that the total weight of every sub-collection $\mathcal{C}_i$ is exactly 1. Also split every cut with weight more than $1/N^2$ into multiple cuts of weight exactly $1/N^2$ each, assigned to the same commodity.
3. While there are pairs of cuts in $\mathcal{C}$ that cross, consider any pair of cuts $C_i, C_j \in \mathcal{C}$ belonging to terminals $i \neq j$ that cross each other. Transform these cuts into new cuts for i and j according to Figure 4.5.

Figure 4.4: Algorithm *Lam-1*—Algorithm to convert an LP solution for the CSCP into a feasible fractional laminar family

Y when its integral load becomes $c_e + 2$. Lemma 4.10 implies that edges in Y never get loaded, and so at the time that an edge e enters Z, $\ell_e^{\mathcal{A}} \leq c_e + 2$. After this point the edge stays in Z, and Lemmas 4.11 and 4.12 imply that it gets loaded at most once. Therefore, the final load on the edge is at most $c_e + 3$. □

4.5 Constructing Fractional Laminar Cut Packings

We now show that fractional solutions to the program **MCP-LP** can be converted in polynomial time into fractional laminar cut families while losing only a small factor in edge load. We begin with the common sink case.

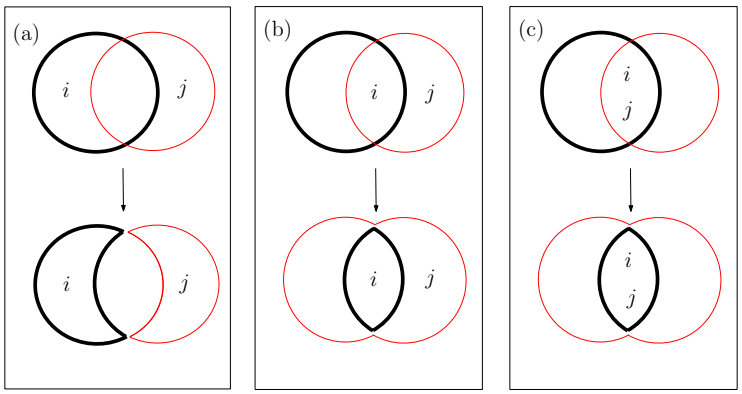

Figure 4.5: Rules for transforming an arbitrary cut family into a laminar one for the CSCP. The dark cuts in this figure correspond to the terminal i, and the light cuts to terminal j; t lies outside all the cuts.

4.5.1 Obtaining Laminarity in the Common Sink Case

We prove Lemma 4.1 in this section. Our algorithm involves starting with a solution to **MCP-LP**, converting it into a feasible fractional *non-laminar* family of cuts, and then resolving pairs of crossing cuts one at a time by applying the rules in Figure 4.5. The algorithm is given in Figure 4.5.

Lemma 4.14. *Consider an instance of the CSCP with graph $G = (V, E)$, common sink t, edge capacities c_e, and commodities $S_1, \cdots, S_k$. Given a feasible solution d to **MCP-LP**, algorithm* Lam-1 *produces in polynomial time a fractional laminar cut family $\mathcal{C}$ that is feasible for the CSCP on G with edge capacities $c_e + o(1)$.*

Proof. We first note that the family $\mathcal{C}$ is feasible for the given instance of CSCP at the end of Step 2, but is not necessarily laminar. Since the number of distinct cuts in $\mathcal{C}$ after Step 1 is at most $nk = N$, at the end of Step 2, edge loads are at most $c_e + 1/N$. As we tranform the cuts in Step 3, we maintain the property that no cut $C \in \mathcal{C}$ contains the sink t, but every cut $C \in \mathcal{C}_i$ contains the node r_i for terminal i. It is also easy to see from Figure 4.5 that the load on every edge stays the same. Finally, in every iteration of this step, the

number of pairs of crossing cuts strictly decreases. Therefore, the algorithm ends after a polynomial number of iterations. □

4.5.2 Obtaining Laminarity in the General Case

Obtaining laminarity in the general case involves a more careful selection and ordering of rules of the form given in Figure 4.5. The key complication in this case is that we must maintain separation of every terminal from every other terminal in its commodity set. We first show how to convert an integral collection of cuts feasible for the MCP into a feasible integral laminar collection of cuts. We lose a factor of 2 in edge loads in this process (see Lemma 4.15 below). Obtaining laminarity for an arbitrary fractional solution requires converting it first into an integral solution for a related cut-packing problem and then applying Lemma 4.15 (see algorithm *Lam-2* in Figure 4.9 and the proof of Lemma 4.2 following it).

Lemma 4.15. *Consider an instance of the MCP with graph $G = (V, E)$ and commodities $S_1, \cdots, S_k$, and let $\mathcal{C}^1 = \{C_i^1\}_{i \in S_a, a \in [k]}$ be a family of cuts such that for each $a \in [k]$ and $i \in S_a$, C_i^1 contains i but no other $j \in S_a$. Then algorithm* Integer-Lam-2 *produces a* laminar *cut collection $\mathcal{C}^2 = \{C_i^2\}_{i \in S_a, a \in [k]}$ such that for each $a \in [k]$ and $i \neq j \in S_a$, either C_i^2 or C_j^2 separates i from j, and $\ell_e^{\mathcal{C}^1} \leq 2\ell_e^{\mathcal{C}^2}$ for every edge $e \in E$.*

In the remainder of this section we interpret cuts as sets of vertices as well as sets of terminals residing at those vertices. The algorithm for laminarity in the integral case is given in Figure 4.6.

As in the common sink case, the algorithm starts by applying a series of simple rules to pairs of crossing cuts while maintaining the invariant that pairs of terminals belonging to the same commodity are always separated by at least one of the two cuts assigned to them. Certain kinds of crossings of cuts are easy to resolve while maintaining this invariant (Step 1

Input: Graph $G = (V, E)$ with edge capacities c_e, commodities $S_1, \cdots, S_k$, a family of cuts $\mathcal{C}$ with one cut for every terminal in $\cup_a S_a$, such that the cut for terminal $i \in S_a$ does not contain any terminal $j \neq i$ in S_a.
Output: A laminar collection of cuts, one for each terminal in $\cup_a S_a$, such that for all a and for all $i, j \in S_a$, $i \neq j$, either the cut for i or the cut for j separates i from j.

---

1. While there are pairs of cuts in $\mathcal{C}$ that cross, do (see Figure 4.8):

   a) Consider any pair of cuts $C_i, C_j \in \mathcal{C}$ belonging to terminals $i \neq j$ that cross each other, such that $r_i \in C_i \setminus C_j$ and $r_j \in C_j \setminus C_i$. Reassign $C_i = C_i \setminus C_j$ and $C_j = C_j \setminus C_i$. Return to Step 1.

   b) Consider any three terminals i_1, i_2, i_3 with cuts C_1, C_2 and C_3 such that $r_{i_1} \in C_1 \cap C_2 \setminus C_3$, $r_{i_2} \in C_2 \cap C_3 \setminus C_1$, and $r_{i_3} \in C_3 \cap C_1 \setminus C_2$. Then, reassign these respective intersections to the three terminals. Return to Step 1.

   c) Consider any pair of cuts $C_i, C_j \in \mathcal{C}$ belonging to terminals $i, j \in S_a$ for some a that cross each other, such that $r_i \in C_i \cap C_j$ and $r_j \in C_j \setminus C_i$. Reassign $C_i = C_i \cap C_j$ and $C_j = C_i \cup C_j$. Return to Step 1.

   d) Consider any pair of cuts $C_i, C_j \in \mathcal{C}$ belonging to terminals $i \neq j$ that cross each other, such that $r_i, r_j \in C_i \cap C_j$, $i \in S_a$ and $j \in S_b$ with $a \neq b$.

      - Suppose that there is no $i' \in S_a \cap C_j$ with $C_i \subset C_{i'}$. Then, reassign $C_i = C_i \cup C_j$ and $C_j = C_i \cap C_j$; return to Step 1. Conversely, if there is no $j' \in S_b \cap C_i$ with $C_j \subset C_{j'}$. Then, reassign $C_j = C_i \cup C_j$ and $C_i = C_i \cap C_j$; return to Step 1. (This transformation is similar to Step 1c.)
      - If neither of those cases hold, let $i_0 = i$, and let $i_1, \cdots, i_x$ denote the terminals in $S_a \cap C_j$ with $C_i \subset C_{i_1} \subset C_{i_2} \subset \cdots \subset C_{i_x}$. For $x' \leq x - 2$, reassign $C_{i_{x'}} = (C_{i_{x'+1}} \setminus C_j) \cup C_{i_{x'}}$, $C_{i_{x-1}} = C_{i_x} \cup C_j$, and $C_{i_x} = C_{i_x} \cap C_j \setminus C_{i_{x-1}}$. Reassign cuts to j and terminals in $S_b \cap C_i$ likewise. Return to Step 1.

   e) If none of the above rules match, then go to Step 2.

Figure 4.6: Part 1 of Algorithm *Integer-Lam-2*—Algorithm to convert an integral family of multiway cuts into a laminar one

2. Let $\mathcal{G}$ be a directed graph on the vertex set $\cup_a S_a$, with edges colored red or blue, defined as follows: for terminals $i \neq j$, $\mathcal{G}$ contains a red edge from i to j if and only if $C_j \subset C_i$, and contains a blue edge from i to j if and only if $r_j \in C_i$, $r_i \notin C_j$, and $C_j \setminus C_i \neq \emptyset$. We note that since no pair of terminals i and j matches the rules in Step 1, whenever C_i and C_j intersect $\mathcal{G}$ contains an edge between i and j.

   While there is a directed blue cycle in $\mathcal{G}$, consider the shortest such cycle $i_1 \to i_2 \to \cdots \to i_x \to i_1$. For $x' \leq x$, $x' \neq 1$, assign to $i_{x'}$ the cut $C_{i_{x'}} \cap C_{i_{x'-1}}$, and assign to i_1 the cut $C_{i_1} \cap C_{i_x}$.

3. We show in Lemma 4.16 that at this step $\mathcal{G}$ is acyclic. For every connected component in $\mathcal{G}$ do:

   a) Let T be the set of terminals in the component and A be the set of corresponding cuts. Assign capacities $p_e = 2\ell_e^A$ to edges in G. Let G_p be the graph obtained by merging all pairs of vertices that have an edge e with $p_e = 0$ between them. We call the vertices of G_p "meta-nodes" (note that these are sets of vertices in the original graph). At any point of time, let R_i denote the meta-node at which a terminal i resides.

   b) While there are terminals in T, pick any "leaf" terminal i (that is, a terminal with no outgoing red or blue edges in $\mathcal{G}$). Reassign to i the cut R_i. Reduce the capacity of every edge $e \in \delta(R_i)$ by 1. Remove i from T; remove i and all edges incident on it from $\mathcal{G}$. Recompute the graph G_p based on the new capacities.

Figure 4.7: Part 2 of Algorithm *Integer-Lam-2*—Algorithm to convert an integral family of multiway cuts into a laminar one

of the algorithm resolves these crossings; see also Figure 4.8). In Steps 2 and 3, we ignore the commodities that each terminal belongs to, and assign new laminar cuts to terminals while ensuring that the new cut of each terminal lies within its previous cut (and therefore, separation continues to be maintained). These steps incur a penalty of 2 in edge loads.

The rough idea behind Steps 2 and 3 is to consider the set of all "conflicting" terminals, call it F. Then we can assign to each terminal $i \in F$ the cut $\cap_{j \in F} \hat{C}_j$ where $\hat{C}_j$ is either the cut of terminal j or its complement depending on which of the two contains r_i. These intersections are clearly laminar, and are subsets of the original cuts assigned to terminals. Furthermore, if each terminal gets a unique intersection, then edge loads increase by a factor

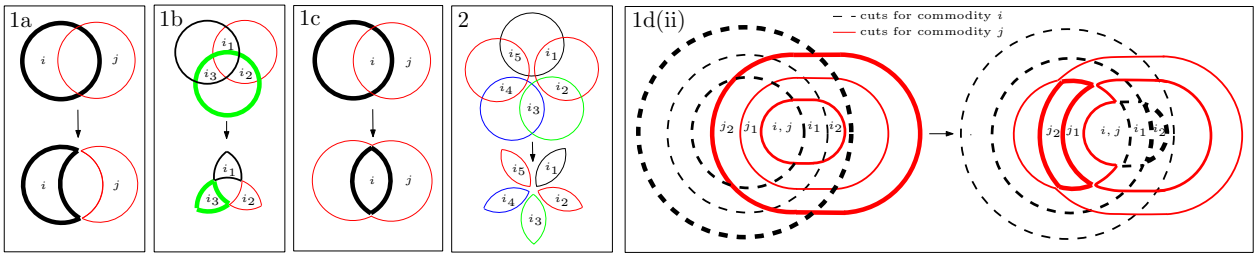

Figure 4.8: Some simple rules for resolving crossing cuts. See algorithm *Integer-Lam-2* in Figure 4.6 for formal descriptions.

of at most 2. Unfortunately, some groups of terminals may share the same intersections. In order to get around this, we assign cuts to terminals in a particular order suggested by the structure of the conflict graph on terminals (graph $\mathcal{G}$ in the algorithm) and assign appropriate intersections to them while explicitly ensuring that edge loads increase by a factor of no more than 2.

Throughout the algorithm, every terminal in $\cup_a S_a$ has an integral cut assigned to it. The proof of Lemma 4.15 is established in three parts: Lemma 4.16 establishes the laminarity of the output cut family, Lemma 4.18 argues separation, and Lemma 4.19 analyzes edge loads.

Lemma 4.16. *Algorithm* Integer-Lam-2 *runs in polynomial time and produces a laminar cut collection.*

Proof. As in the previous section define the crossing number of a family of cuts to be the number of pairs of cuts that cross each other. We first note that in every iteration of Steps 1 and 2 of the algorithm, the crossing number of the cut family $\mathcal{C}$ strictly decreases: no new crossings are created in these steps, while the crossings of the two or more cuts involved in each transformation are resolved (see Figure 4.8). Therefore, after a polynomial number of steps, we exit Steps 1 and 2 and go to Step 3.

Next, we claim that during Step 3 of the algorithm the graph $\mathcal{G}$ is acyclic. This implies

that while $\mathcal{G}$ is non-empty, we can always find a leaf terminal in Step 3; therefore every terminal in $\mathcal{G}$ gets assigned a new cut. It is immediate that the graph does not contain any directed blue cycles or any directed red cycles (the latter follows because red edges define a partial order over terminals). Suppose the graph contains three terminals i_1, i_2 and i_3 with a red edge from i_1 to i_2, and a red or blue edge from i_2 to i_3, then it is easy to see that there must be a red or blue edge from i_1 to i_3. Therefore, any multi-colored directed cycle must reduce to either a smaller blue cycle or a cycle of length 2. Neither of these cases is possible (the latter is ruled out by definition), and therefore the graph cannot contain any multi-colored cycles.

Now consider cuts assigned during Step 3. Let T be the set of terminals corresponding to some component in $\mathcal{G}$ and $j \notin T$. Then before T is processed, j's cut is laminar with respect to all the cuts in A_T, and is therefore a subset of some meta-node in G_{p^T}. So the new cuts assigned to terminals in T are also laminar with respect to j's cut.

Finally, consider any two cuts assigned during Step 3 of the algorithm and belonging to two terminals in the same component of $\mathcal{G}$. Consider the set of all meta-nodes created during this iteration of Step 3. This set is laminar, and the cuts assigned during this iteration are a subset of this laminar family. Therefore, they are laminar. □

Lemma 4.17. *For a commodity i assigned a cut in Step 3 of algorithm* Integer-Lam-2*, let C_i^1 be its cut before this step, and C_i^2 be the new cut assigned to it. Then $C_i^2 \subseteq C_i^1$.*

Proof. We assume without loss of generality that prior to Step 3 each edge load is at most one; this can be achieved by splitting a multiply-loaded edge into many edges. We focus on the behavior of the algorithm for a single component T of $\mathcal{G}$ and prove the lemma by induction over time.

Consider an iteration of Step 3b during which some terminal $i \in T$ is assigned and let C_i be its original cut. Consider any vertex $v \notin C_i$ and let P be a shortest simple path from

r_i to v in G_{p^T} (where the length of an edge e is given by p_e^T just prior to when i is assigned a new cut). It is easy to see that there is one such shortest path that crosses each new cut assigned prior to this iteration in Step 3b at most twice – suppose there are multiple entries and exits for some cut, we can "short-cut" the path by connecting the first point on the path inside the cut to the last point on the path inside the cut via a simple path of length 0 lying entirely inside the cut. We pick P to be such a path. We will prove that P's length is at least 2. So the meta-node containing i must lie inside the cut C_i, and the lemma holds.

Let T_1 (resp. T_2) be the set of terminals in $T \setminus C_i$ (resp. $T \cap C_i$) that are assigned new cuts before i in this iteration. We first note that for any j in T_1, prior to this step, there is no edge from j to i (as j is assigned before i), so $r_i \notin C_j$, and this along with $r_j \notin C_i$ implies that C_i and C_j are disjoint. This implies that the new cut of j (which is a subset of C_j by induction) is also disjoint from C_i, and therefore cannot load any edge with an end-point in C_i. So the only new cuts assigned this far in Step 3b that load edges in P belong to terminals in T_2.

Now we will analyze P's length by accounting for all the newly assigned cuts that load its edges. Let S_P be the set of terminals in T_2 that load an edge in P, and $j \in S_P$. Since the new cut of j intersects P, by the induction hypothesis, C_j should either intersect P or contain the entire path inside it. If C_j contains P entirely, then $C_j \setminus C_i \neq \emptyset$, and furthermore $r_i, r_j \in C_i \cap C_j$. This implies that either $C_i \subset C_j$ and there is a directed red edge from j to i, or $C_i \setminus C_j \neq \emptyset$, that is, C_i and C_j cross and should have matched the rule in Step 1d of the algorithm. Both possibilities lead to a contradiction. Therefore, C_j must intersect P.

Finally, the original total length of the path is at least $2|S_P| + 2$, because each terminal in S_P contributes two units towards its length, and another two units is contributed by C_i. Out of these up to $2|S_P|$ units of length is consumed by terminals in S_P. Therefore, at the time that i is assigned a cut, at least 2 units remain. □

Lemma 4.18. *When algorithm* Integer-Lam-2 *terminates, for every* $a \in [k]$ *and* $i \neq j \in S_a$, *either* C_i *or* C_j *separates* i *from* j.

Proof. We claim that for every $a \in [k]$ and $i \neq j \in S_a$, at every time step during the execution of the algorithm, $|C_i \cap C_j \cap \{r_i, r_j\}| \leq 1$. Then since by Lemma 4.16 the final solution is laminar, the lemma follows. We prove this claim by induction over time. First, if during any iteration of the algorithm, we "shrink" the cut of any terminal (that is, reassign to the terminal a cut that is a strict subset of its original cut), then the claim continues to hold for that terminal, because intersections of the terminal's cut only shrink in that step. Note that cuts of terminals expand only in Steps 1c and 1d of the algorithm (by construction and by Lemma 4.17).

Suppose that during some iteration we apply the transformation in Step 1c to terminals i and j, reassigning $C_j = C_i \cup C_j$, and the claim fails to hold for terminal j. Specifically, suppose that for some $j' \in S_a$, after the iteration we have $r_j, r_{j'} \in C_j \cap C_{j'}$. Then, $r_j \in C_{j'}$, and therefore $C_{j'}$ intersected C_j prior to the iteration, and by the induction hypothesis $r_{j'} \in C_i \setminus C_j$ prior to the iteration. If $r_i \in C_{j'}$, then prior to the iteration, i and j' contradicted the induction hypothesis. Otherwise, i, j and j' satisfy the conditions in Step 1b of the algorithm, and this contradicts the fact that we apply the transformation in Step 1c at this iteration.

Next suppose that during some iteration we apply the transformation in the first part of Step 1d to terminals i and j, reassigning $C_j = C_i \cup C_j$, and the claim fails to hold for terminal j; in particular, for some $j' \in S_a$, after the iteration we have $r_j, r_{j'} \in C_j \cap C_{j'}$. Then, since $r_j \in C_{j'}$ and the pair of terminals did not match the criteria in Step 1c, it must be the case that $C_j \subset C_{j'}$ prior to the iteration. Furthermore, $r_{j'} \in C_i$ prior to the iteration and this contradicts the fact that we applied the transformation in the first part of Step 1d.

Finally, suppose that during some iteration we apply the transformation in the second

part of Step 1d. Then the cut assigned to every $i_{x'}$ for $x' \leq x-2$ is a subset of the previous cut of $i_{x'+1}$, but does not contain the latter terminal, and so by the arguments presented for the previous cases, once again the induction hypothesis continues to hold for those terminals. Furthermore, the cut assigned to i_x is a subset of its original cut and i_{x-1} does not belong to any of the new cuts except its own. The same argument holds for the $j_{y'}$ terminals. □

Lemma 4.19. *For the cut collection produced by algorithm* Integer-Lam-2 *the load on every edge is no more than twice the load of the integral family of cuts input to the algorithm.*

Proof. We first claim that edge loads are preserved throughout Steps 1 and 2 of the algorithm. This can be established via a case-by-case analysis by noting that in every transformation of these steps, the number of new cuts that an edge crosses is no more than the number of old cuts that the edge crosses prior to the transformation. It remains to analyze Step 3 of the algorithm. We claim that we only lose a factor of 2 in edge loads during this step of the algorithm. This is easy to see. Note that for every edge e, $\sum_T p_e^T \leq 2\ell_e^{\mathcal{C}_{\cup T}}$, where $\mathcal{C}_{\cup T}$ is the family of cuts belonging to terminals in any non-singleton component of $\mathcal{G}$ prior to Step 3. Moreover, in each iteration of the step, we only load an edge e to the extent of p_e^T. Therefore the lemma follows. □

Proof of Lemma 4.15 The proof follows immediately from Lemmas 4.16, 4.18 and 4.19.

Given this lemma, algorithm *Lam-2* in Figure 4.9 converts an arbitrary feasible solution for **MCP-LP** into a feasible fractional laminar family.

Lemma 4.20. *Consider an instance of the MCP with graph $G = (V, E)$, edge capacities c_e, and commodities $S_1, \cdots, S_k$. Given a feasible solution d to **MCP-LP**, algorithm* Lam-2 *produces a fractional laminar cut family $\mathcal{C}$ that is feasible for the MCP on G with edge capacities $8c_e + o(1)$.*

Input: Graph $G = (V, E)$ with edge capacities c_e, commodities $S_1, \cdots, S_k$, a feasible solution d to the program **MCP-LP**.
Output: A fractional laminar family of cuts $\mathcal{C}$ that is feasible for G with edge capacities $8c_e + o(1)$.

1. For every $a \in [k]$ and every terminal $i \in S_a$ do the following: Order the vertices in G in increasing order of their distance under d_a from r_i. Let this ordering be $v_0 = r_i, v_1, \cdots, v_n$. Let $\mathcal{C}_i^1$ be the collection of cuts $\{v_0, v_1, \cdots, v_b\}$, one for each $b \in [n]$ with $d_a(r_i, v_b) < 0.5$, with weights $w^1(\{v_0, \cdots, v_b\}) = 2(\min\{d_a(r_i, v_{b+1}), 0.5\} - d_a(r_i, v_b))$. Let $\mathcal{C}^1$ denote the collection $\{\mathcal{C}_i^1\}_{i \in \cup_a S_a}$.

2. Let $N = n \sum_a |S_a|$. Round up the weights of all the cuts in $\mathcal{C}^1$ to multiples of $1/N^2$, and truncate the collection so that the total weight of every sub-collection $\mathcal{C}_i^1$ is exactly 1. Furthermore, split every cut with weight more than $1/N^2$ into multiple cuts of weight exactly $1/N^2$ each, assigned to the same commodity. Call this new collection $\mathcal{C}^2$ with weight function w^2. Note that every cut in this collection has weight exactly $1/N^2$.

3. Construct a new instance of MCP in the same graph G as follows. For each $a \in [k]$, construct N^2 new commodities with terminal sets identical to that of S_a (that is the terminals reside at the same nodes). For every new terminal corresponding to an older terminal i, assign to the new terminal a unique cut from $\mathcal{C}_i^2$ with weight 1. Call this new collection $\mathcal{C}^3$, and the new instance I.

4. Apply algorithm *Integer-Lam-2* from Figure 4.6 to the family $\mathcal{C}^3$ to obtain family $\mathcal{C}^4$.

5. For every $a \in [k]$ and every $i \in S_a$, let $\mathcal{C}_i^5$ be the set of $N^2/2$ innermost cuts in $\mathcal{C}^4$ assigned to terminals in the new instance I that correspond to terminal i. (Note that these cuts are concentric as they belong to a laminar family and all contain r_i. Therefore "innermost" cuts are well defined.) Assign a weight of $2/N^2$ to every cut in this set. Output the collection $\mathcal{C}^5$.

Figure 4.9: Algorithm *Lam-2*—Algorithm to convert an LP solution into a feasible fractional laminar family

Proof. Note first that the cut collection $\mathcal{C}^1$ satisfies the following properties: (1) For every $a \in [k]$ and $i \in S_a$, every cut in $\mathcal{C}_i^1$ contains r_i, but not r_j for $j \in S_a$, $j \neq i$; (2) The total weight of cuts in C_i^1 is 1; (3) For every edge e, $\ell_e^{\mathcal{C}^1} \leq 2 \sum_a d_a(e) \leq 2c_e$. The family $\mathcal{C}^2$ also satisfies the first two properties, however loads the edges slightly more than $\mathcal{C}^1$. Any edge belongs to at most N cuts, and therefore the load on the edge goes up by an additive amount of at most $1/N$. Therefore, for every e, $\ell_e^{\mathcal{C}^2} \leq 2c_e + 1/N$. Next, the collection $\mathcal{C}^3$

is a feasible integral family of cuts for the new instance I with $\ell_e^{\mathcal{C}^3} = N^2 \ell_e^{\mathcal{C}^2}$. Therefore, applying Lemma 4.15, we get that $\mathcal{C}^4$ is a feasible laminar integral family of cuts for I with $\ell_e^{\mathcal{C}^4} \leq 2N^2(2c_e + 1/N)$. Finally, in family $\mathcal{C}^5$, every terminal $i \in S_a$ gets assigned $N^2/2$ fractional cuts, each with weight $2/N^2$. Therefore, the total weight of cuts in $\mathcal{C}_i^5$ is 1. Now consider any two terminals $i, j \in S_a$ with $i \neq j$. Then, in all the N^2 commodities corresponding to S_a in instance I, either the cut assigned to i's counterpart, or that assigned to j's counterpart separates i from j. Say that among at least $N^2/2$ of the commodities in I', the cut assigned to i's counterpart separates i from j. Then, the innermost $N^2/2$ cuts assigned to i in $\mathcal{C}^5$ separate i from j. Therefore, the family $\mathcal{C}^5$ satisfies the first two conditions of feasibility as given in Definition 4.2. Finally, it is easy to see that on every edge e, $\ell_e^{\mathcal{C}^5} \leq 2/N^2 \ell_e^{\mathcal{C}^4} \leq 4(2c_e + 1/N)$. □

4.6 Concluding Remarks

Given that our algorithms rely heavily on the existence of good laminar solutions, a natural question is whether every feasible solution to the MCP can be converted into a laminar one with the same load. Figure 4.10 shows that this is not true. The figure displays one integral solution to the MCP where the solid edges represent the cut for commodity a, and the dotted edges represent the cut for commodity b. It is easy to see that this instance admits no fractional laminar solution with load 1 on every edge.

Is the "laminarity gap" small for the more general set multiway cut packing and multicut packing problems as well? We believe that this is not the case and there exist instances for both of those problems with a non-constant laminarity gap.

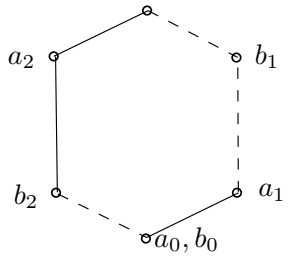

Figure 4.10: Each edge has capacity 1. There are two commodities with terminal sets $\{a_0, a_1, a_2\}$ and $\{b_0, b_1, b_2\}$.

Bibliography

[AA97] B. Awerbuch and Y. Azar. Buy-at-bulk network design. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, 1997.

[ACKN07] Y. P. Aneja, R. Chandrasekaran, S. N. Kabadi, and K. P. K. Nair. Flows over edge-disjoint mixed multipaths and applications. *Discrete Appl. Math.*, 155(15):1979–2000, 2007.

[AGA+08] Ashok Anand, Archit Gupta, Aditya Akella, Srinivasan Seshan, and Scott Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 219–230, 2008.

[AO02] C. Aggarwal and J. Orlin. On multi-route maximum flows in networks. *Networks*, 39:43–52, 2002.

[BC09] S. Barman and S. Chawla. Packing multiway cuts in capacitated graphs. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1048–1057. Society for Industrial and Applied Mathematics, 2009.

[BC10] S. Barman and S. Chawla. Region growing for multi-route cuts. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 404–418. Society for Industrial and Applied Mathematics, 2010.

[BC12] S. Barman and S. Chawla. Traffic-redundancy aware network design. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1487–1498. SIAM, 2012.

[BČHK07] H. Bruhn, J. Černỳ, A. Hall, and P. Kolman. Single source multiroute flows and cuts on uniform capacity networks. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 855–863. Society for Industrial and Applied Mathematics, 2007.

[BCK05] A. Bagchi, A. Chaudhary, and P. Kolman. Short length menger's theorem and reliable optical networking. *Theoretical Computer Science*, 339:315332, 2005.

[BCK10] Anand Bhalgat, Deeparnab Chakrabarty, and Sanjeev Khanna. Optimal lower bounds for universal and differentially private steiner tree and tsp. *CoRR*, abs/1011.3770, 2010.

[BCKS04] A. Bagchi, A. Chaudhary, P. Kolman, and J. Sgall. A simple combinatorial proof for the duality of multiroute flows and cuts. Technical Report 2004–662, Charles University, 2004.

[BCSK07] A. Bagchi, A. Chaudhary, C. Scheideler, and P. Kolman. Algorithms for fault-tolerant routing in circuit-switched networks. *SIAM J. Discret. Math.*, 21(1):141–157, 2007.

[CK08] C. Chekuri and S. Khanna. Algorithms for 2-route cut problems. In *ICALP*, 2008.

[CKNZ04] C. Chekuri, S. Khanna, J. Naor, and L. Zosin. A linear programming formulation and approximation algorithms for the metric labeling problem. *SIAM J. on Discrete Mathematics*, 18(3):608–625, 2004.

[CKR00] G. Calinescu, H. Karloff, and Y. Rabani. An improved approximation algorithm for multiway cut. *JCSS*, 60(3):564–574, 2000.

[CKR04] G. Calinescu, H. Karloff, and Y. Rabani. Approximation algorithms for the 0-extension problem. *SICOMP*, 34(2):358–372, 2004.

[CMVZ12] Julia Chuzhoy, Yury Makarychev, Aravindan Vijayaraghavan, and Yuan Zhou. Approximation algorithms and hardness of the k-route cut problem. In *SODA*, pages 780–799, 2012.

[CPR04] A. Caprara, A. Panconesi, and R. Rizzi. Packing cuts in undirected graphs. *Networks*, 44(1):1–11, 2004.

[FRT03] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455, 2003.

[GGP+94] M.X. Goemans, A.V. Goldberg, S. Plotkin, D.B. Shmoys, E. Tardos, and D.P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 223–232. Society for Industrial and Applied Mathematics, 1994.

[GK11] Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16:3 – 20, 2011.

[GKR03] Anupam Gupta, Amit Kumar, and Tim Roughgarden. Simpler and better approximation algorithms for network design. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 365–372, 2003.

[GKR+07] A. Gupta, A. Kumar, T. Roughgarden, et al. Approximation via cost sharing: Simpler and better approximation algorithms for network design. *Journal of the ACM (JACM)*, 54(3):11, 2007.

[GKR09] Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. Online and stochastic survivable network design. In *STOC*, pages 685–694, New York, NY, USA, 2009. ACM.

[GVY96] N. Garg, V. V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM J. Comput.*, 25(2):235–251, 1996.

[HST05] A. Hayrapetyan, C. Swamy, and É. Tardos. Network design for information networks. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 933–942. Society for Industrial and Applied Mathematics, 2005.

[Jai01] K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[JLN+05] Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for tsp, steiner tree, and set cover. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 386–395, 2005.

[Kar98] A. Karzanov. Minimum 0-extensions of graph metrics. *European J. of Combinatorics*, 19(1):71–101, 1998.

[Kis96] W. Kishimoto. A method for obtaining the maximum multi-route flow in a network. *Networks*, 27(4):279–291, 1996.

[KM00] D. R. Karger and M. Minkoff. Building steiner trees with incomplete global knowledge. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 2000.

[KS11] Petr Kolman and Christian Scheideler. Towards duality of multicommodity multiroute cuts and flows: Multilevel ball-growing. In *STACS*, pages 129–140, 2011.

[KT93] W Kishimoto and M. Takeuchi. On m-route flows in a network. *IEICE Trans J-76-A*, pages 1185–1200, 1993.

[KT02] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and Markov random fields. *JACM*, 49(5):616–639, 2002.

[LMW02] M. Li, B. Ma, and L. Wang. On the closest string and substring problems. *JACM*, 49(2):157–171, 2002.

[LV92] Jyh-Han Lin and Jeffrey Scott Vitter. Approximation algorithms for geometric median problems. *Inf. Process. Lett.*, 44:245–249, December 1992.

[LY78] C. L. Lucchesi and D. H. Younger. A minimax theorem for directed graphs. *J. London Math. Soc.*, 17:369–374, 1978.

[men] Menger's theorems and max-flow-min-cut. http://math.fau.edu/locke/Menger.htm.

[MMP08] Adam Meyerson, Kamesh Munagala, and Serge Plotkin. Cost-distance: Two metric network design. *SIAM J. Comput.*, 38:1648–1659, 2008.

[MSVV07] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22, 2007.

[nwc] The Network Coding Home Page. http://www.networkcoding.info/.

[RJ98] R. Ravi and J.Kececioglu. Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree. *Discrete Applied Mathematics*, 88:355–366, 1998.

[RSS08] Y. Rabani, L. Schulman, and C. Swamy. Approximation algorithms for labeling hierarchical taxonomies. In *SODA*, pages 671–680, 2008.

[RT02] T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):236–259, 2002.

[RZ00] Gabriel Robins and Alexander Zelikovsky. Improved steiner tree approximation in graphs. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 770–779, 2000.

[Sch03] A. Schrijver. *Combinatorial optimization*. Springer, 2003.

[Sch05] A. Schrijver. On the history of combinatorial optimization (till 1960). *Handbook of Discrete Optimization*, pages 1–68, 2005.

[Shm97] David B. Shmoys. Cut problems and their application to divide-and-conquer. In *Approximation algorithms for NP-hard problems*, pages 192–235. PWS Publishing Co., Boston, MA, USA, 1997.

[SSL04] David B. Shmoys, Chaitanya Swamy, and Retsef Levi. Facility location with service installation costs. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1088–1097, 2004.

[ST06] Zoya Svitkina and Éva Tardos. Facility location with hierarchical facility costs. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 153–161, 2006.

[STA97] David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274, 1997.

[Tal02] Kunal Talwar. The single-sink buy-at-bulk lp has constant integrality gap. In *Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 475–486, 2002.

[Tal06] K. Talwar. The single-sink buy-at-bulk lp has constant integrality gap. *Integer Programming and Combinatorial Optimization*, pages 475–486, 2006.

[WG97] L. Wang and D. Gusfield. Improved approximation algorithms for tree alignment. *Journal of Algorithms*, 25(2):255–273, 1997.

[WGMV95] D.P. Williamson, M.X. Goemans, M. Mihail, and V.V. Vazirani. A primal-dual approximation algorithm for generalized steiner network problems. *Combinatorica*, 15(3):435–454, 1995.

[WJG00] L. Wang, T. Jiang, and D. Gusfield. A more efficient approximation scheme for tree alignment. *SIAM Journal on Computing*, 30(1):283–299, 2000.

[WJL96] L. Wang, T. Jiang, and E. Lawler. Approximation algorithms for tree alignment with a given phylogeny. *Algorithmica*, 16(3):302–315, 1996.