On Network Coding Capacity - Matroidal Networks and Network Capacity Regions

by

Anthony Eli Kim

S.B., Electrical Engineering and Computer Science (2009), and
S.B., Mathematics (2009)
Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Electrical Engineering and Computer Science
August 5, 2010

Certified by .....
Muriel Médard
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Christopher J. Terman
Chairman, Department Committee on Graduate Theses

On Network Coding Capacity - Matroidal Networks and Network Capacity Regions

by

Anthony Eli Kim

Submitted to the Department of Electrical Engineering and Computer Science
on August 5, 2010, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

One fundamental problem in the field of network coding is to determine the network coding capacity of networks under various network coding schemes. In this thesis, we address the problem with two approaches: matroidal networks and capacity regions.

In our matroidal approach, we prove the converse of the theorem which states that, if a network is scalar-linearly solvable then it is a matroidal network associated with a representable matroid over a finite field. As a consequence, we obtain a correspondence between scalar-linearly solvable networks and representable matroids over finite fields in the framework of matroidal networks. We prove a theorem about the scalar-linear solvability of networks and field characteristics. We provide a method for generating scalar-linearly solvable networks that are potentially different from the networks that we already know are scalar-linearly solvable.

In our capacity region approach, we define a multi-dimensional object, called the network capacity region, associated with networks that is analogous to the rate regions in information theory. For the network routing capacity region, we show that the region is a computable rational polytope and provide exact algorithms and approximation heuristics for computing the region. For the network linear coding capacity region, we construct a computable rational polytope, with respect to a given finite field, that inner bounds the linear coding capacity region and provide exact algorithms and approximation heuristics for computing the polytope. The exact algorithms and approximation heuristics we present are not polynomial time schemes and may depend on the output size.

Thesis Supervisor: Muriel Médard
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my advisor Muriel Médard for her kind support, encouragement, and guidance throughout the course of this work. I am greatly indebted to her not only for introducing me to the field of network coding, but also for providing invaluable advices leading to my intellectual and personal developments. I would also like to thank Una-May O'Reilly for her support and guidance during the early developments leading up to this work and Michel Goemans for thoughtful discussions on parts of the work. I would like to thank members of the Network Coding and Reliable Communications Group of RLE for insightful discussions on network coding problems and for helpful personal advices on many subjects. Finally, I would like to thank my parents, Kihwan Kim and Guiyeum Kim, and sister Caroline for their overflowing love and support throughout my life; and, I would like to thank close friends and members of the MIT community for making my years at MIT unforgettable and, most importantly, enjoyable.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Network coding is a field at the intersection of network information theory and coding theory. The central idea of the field of network coding is that increased capabilities of intermediate nodes lead to improvements in information throughput of the network. In the traditional routing model of information networks, intermediate nodes simply copy and forward incoming packets, and the information flows were modeled as source-to-sink paths and Steiner trees. In the network coding model, intermediate nodes are now allowed more complicated operations to code on incoming packets and forward packets that might differ significantly from the incoming packets. It has been shown numerous times that the network coding model allows greater information throughput than in the traditional routing model and its applicability has been widely researched. One fundamental problem in the field of network coding is to determine the network coding capacity, the maximum amount of information throughput, of networks under various network coding schemes. In this work, we address the problem with two approaches: matroidal networks and capacity regions.

1.1 Network Coding Model

We give a network coding model that we will use in this work. Most of it is adapted from [8]. Further additional definitions are relegated to relevant chapters. Throughout the work, we assume that the networks are acyclic and the edges (or links) between nodes are delay-free and error-free.

Definition 1 (Network). *A* network $\mathcal{N}$ *is a finite, directed, acyclic multigraph given by a 6-tuple* $(\nu, \epsilon, \mu, \mathcal{A}, S, R)$ *where*

1. ν *is a node set,*

2. ϵ *is an edge set,*

3. μ *is a message set,*

4. $\mathcal{A}$ *is an alphabet,*

5. $S : \nu \rightarrow 2^{\mu}$ *is a source mapping, and*

6. $R : \nu \rightarrow 2^{\mu}$ *is a receiver mapping.*

We use a pair of nodes (x, y) to denote a directed edge from node x to node y; x is the *start node* and y is the *end node.* For each node x, if $S(x)$ is nonempty then x is a *source* and if $R(x)$ is nonempty then x is a *receiver.* The elements of $S(x)$ are called the *messages generated by* x and the elements of $R(x)$ are called the *messages demanded by* x. An *alphabet* $\mathcal{A}$ is a finite set with at least two elements. Each instance of a message is a vector of elements from the alphabet. For each node x, let $\text{In}(x)$ denote the set of messages generated by x and in-edges of x. Let $\text{Out}(x)$ denote the set of messages demanded by x and out-edges of x. For each node x, we fix an ordering of $\text{In}(x)$ and $\text{Out}(x)$ such that all messages occur before the edges in the resulting lists. In our definition of networks, there could be multiple source nodes and multiple receiver nodes with arbitrary demands.

There are several special classes of networks: *unicast networks* where there are exactly one message, one source and one receiver; *multicast networks* where there are exactly one message and one source, but an arbitrary number of receivers that demand the message; *two-level multicast networks* where there are multiple messages and there are exactly one source node that generates all the network messages and two receivers where one demands all the messages and the other demands a subset of the messages; *multiple unicast networks* where there are multiple messages and, for each message m, we have the unicast condition; *multiple multicast networks* where there are multiple messages and, for each message m, we have the multicast condition. A general network has multiple source nodes, multiple receiver nodes, and arbitrary demands of messages; they are sometimes referred to as *multi-source multi-sink networks* in literature. A multicast network in literature usually has multiple messages, but we restrict multicast networks to those with a single message in this work.

We define edge function, decoding function, message assignment and symbol function with respect to a finite field F of cardinality greater than or equal to $|\mathcal{A}|$. We choose such F so that each element from $\mathcal{A}$ can be uniquely represented with an element from F.

Definition 2 (Edge and Decoding Functions). *Let k and n be positive integers. For each edge $e = (x, y)$, an* edge function *is a map*

$$f_e : (F^k)^\alpha \times (F^n)^\beta \to F^n,$$

where α and β are number of messages generated by x and in-edges of x, respectively. For each node $x \in \nu$ and message $m \in R(x)$, a decoding function *is a map*

$$f_{x,m} : (F^k)^\alpha \times (F^n)^\beta \to F^k,$$

where α and β are number of messages generated by x and in-edges of x, respectively. We call k and n the source dimension *and* edge dimension, *respectively.*

Each source sends a message vector of length k and each edge carries a message vector of length n. We denote the collections of edge and decoding functions by $\mathcal{F}_e = \{f_e : e \in \epsilon\}$ and $\mathcal{F}_d = \{f_{x,m} : x \in \nu, m \in R(x)\}$.

Definition 3 (Message Assignment). *A* message assignment *is a map $a : \mu \to F^k$, i.e., each message is assigned with a vector from F^k.*

Definition 4 (Symbol Function). *A* symbol function *is a map $s : \epsilon \to F^n$ defined recursively, with respect to $\mathcal{N}$ and $\mathcal{F}_e$, such that for all $e = (x, y) \in \epsilon$,*

$$s(e) = f_e(a(m_1), \ldots, a(m_\alpha), s(e_{\alpha+1}), \ldots, s(e_{\alpha+\beta})),$$

where $m_1, \ldots, m_\alpha$ are the messages generated by x and $e_{\alpha+1}, \ldots, e_{\alpha+\beta}$ are the in-edges of x. Note that the symbol function is well-defined as network $\mathcal{N}$ is a directed acyclic multigraph.

Definition 5 (Network Code). *A* network code *on $\mathcal{N}$ is a 5-tuple $(F, k, n, \mathcal{F}_e, \mathcal{F}_d)$ where*

1. *F is a finite field, with $|F| \geq |\mathcal{A}|$,*

2. *k is a source dimension,*

3. *n is an edge dimension,*

4. *$\mathcal{F}_e$ is a set of edge functions on network $\mathcal{N}$,*

5. *$\mathcal{F}_d$ is a set of decoding functions on network $\mathcal{N}$.*

We shall use the prefix (k, n) before codes when we wish to be more specific on parameters k and n. When k and n are clear from the context, we will sometimes omit them. There are several special classes of network codes: *routing network codes*, where edge and decoding functions simply copy input vector components to output vector components, *linear network codes*, where edge and decoding functions are linear over F, and *nonlinear network codes*, where edge and decoding functions are nonlinear over F. *Vector-linear network codes* are linear network codes with $k = n$. *Scalar-linear network codes* are linear network codes with $k = n = 1$.

Definition 6 (Network Code Solution). *A network code $(F, k, n, \mathcal{F}_e, \mathcal{F}_d)$ is a* network code solution, *or* solution *for short, if for every message assignment $a : \mu \to F^k$,*

$$f_{x,m}(a(m_1), \ldots, a(m_\alpha), s(e_{\alpha+1}), \ldots, s(e_{\alpha+\beta})) = a(m),$$

for all $x \in \nu$ and $m \in R(x)$. Note that $m_1, \ldots, m_\alpha$ are messages generated by x, and $e_{\alpha+1}, \ldots, e_{\alpha+\beta}$ are in-edges of x. If the above equation holds for a particular node $x \in \nu$ and message $m \in R(x)$, then we say node x's demand m is satisfied.

A network $\mathcal{N}$ is *routing-solvable* if it has a routing network code solution. Similarly, we say that network $\mathcal{N}$ is *linearly solvable* (*scalar-linearly solvable, vector-linearly solvable, nonlinearly solvable*) if it has a *linear* (*scalar-linear, vector-linear, nonlinear*) network code solution.

1.2 Previous Works

In a seminal work in 2000, Ahlswede et al. [1] introduced the network coding model to the problem of communicating information in networks. They showed that the extended capabilities of intermediate nodes to code on incoming packets give greater information throughput than in the

traditional routing model. They also showed that the capacity of any multiple multicast network of a certain class is equal to the minimum of min-cuts between the source node and receiver nodes.

Single source networks and linear network coding are comparatively well-understood. Li et al. [22] showed that linear network coding is sufficient for certain multiple multicast networks. Koetter and Médard [20] reduced the problem of determining scalar-linear solvability to solving a set of polynomial equations over some finite field and suggested connections between scalar-linearly solvable networks and nonempty varieties in algebraic geometry. They showed that scalar-linear solvability of many special case networks, such as two-level multicasts, can be determined by their method. Dougherty et al. [9] strengthened the connection by demonstrating solvably equivalent pairs of networks and polynomial collections; for any polynomial collection, there exists a network that is scalar-linearly solvable over field F if and only if the polynomial collection is solvable over F. It is known that scalar-linear network codes are not sufficient in general. The M-network due to Koetter in [23] is a network with no scalar-linear solution but has a vector-linear solution. Lehman and Lehman [21] using 3-CNF formulas also provided an example where a vector solution is necessary.

More recently, matroidal approaches to analyze networks have been quite successful. Dougherty et al. [7, 8] defined and studied matroidal networks and suggested connections between networks and matroids. They used matroidal networks constructed from well-known matroids to show in [6] that not all solvable networks have a linear solution over some finite-field alphabet and vector dimension. They also constructed a matroidal network to show that Shannon-type information inequalities are not sufficient for computing network coding capacities in general. Recently, El Rouayheb et al. [10] strengthened the connection between networks and matroids by constructing "solvably equivalent" pairs of networks and matroids via index codes with their own construction method; the network has a vector-linear solution over a field if and only if the matroid has a multilinear representation over the same field. In another recent work [25], Sun et al. studied the matroid structure of single-source networks which they define as network matroid and showed connections between the network matroids and a special class of linear network codes.

The capacity regions of networks are less well-understood, but a few explicit outer bounds of capacity regions of networks exist. One easy set of outer bounds is the max-flow/min-cut bounds, which were sufficient in the case of certain multiple multicast networks. Harvey et al.[16] combined

information theoretic and graph theoretic techniques to provide a computable outer bound on the network coding capacity regions of networks. Yan et al.[27] gave an explicit outer bound for networks that improved upon the max-flow/min-cut outer bound and showed its connection to a kind of minimum cost network coding problem. They used their results to compute the capacity region of a special class of 3-layer networks. Thakor et al.[26] gave a new computable outer bound, based on characterizations of all functional dependencies in networks, that is provably tighter than those given in [16] and [27].

Recently, explicit characterizations of capacity regions, albeit hard to compute, of networks were given using information theoretic approaches. Yan et al.[28] provided an exact characterization of the capacity regions for general multi-source multi-sink networks by bounding the constrained regions in the entropy space. However, they noted that explicitly evaluating the obtained capacity regions remains difficult in general. In a related work, Chan and Grant[3] showed that even the explicit characterization of capacity regions for single-source networks can be difficult since the computation of a capacity region reduces to the determination of the nonpolyhedral set of all entropy functions and that linear programming bounds do not suffice.

The routing capacity regions of networks are better understood via linear programming approaches. Cannons et al.[2] defined the notion of network routing capacity that is computable with a linear program and showed that every rational number in $(0, 1]$ is the routing capacity of some solvable network. Yazdi et al.[29, 30] extended a special case of Farkas Lemma called the "Japanese Theorem" to reduce an infinite set of linear constraints to a set of finitely many linear constraints in terms of minimal Steiner trees and applied the results to obtain the routing capacity region of undirected ring networks. In a subsequent work, Kakhbod and Yazdi[19] provided the complexity results on the description size of the finitely many inequalities obtained in [29, 30] and apply them to the undirected ring networks.

1.3 Our Results

We organize our contributions into two parts: matroidal networks and network capacity regions. In both approaches, we provide examples to demonstrate our main ideas.

1.3.1 Matroidal Networks

In our matroidal approach, we further study the matroidal networks introduced by Dougherty et al. [8]. Our contributions can be summarized as follows and we refer to Chapter 2 for details:

1. We prove the converse of a theorem in [8] which states that, if a network is scalar-linearly solvable then it is a matroidal network associated with a representable matroid over a finite field.

2. We prove a theorem about the scalar-linear solvability of networks and field characteristics.

3. We provide a method for generating scalar-linearly solvable networks that are potentially different from the networks that we already know are scalar-linearly solvable.

As a consequence, we obtain a correspondence between scalar-linearly solvable networks and representable matroids over finite fields in the framework of matroidal networks. It also follows that determining scalar-linear solvability of a network $\mathcal{N}$ is equivalent to determining the existence of a representable matroid $\mathcal{M}$ over a finite field and a valid network-matroid mapping between $\mathcal{M}$ and $\mathcal{N}$. We obtain a set of scalar-linearly solvable networks that are potentially different from the networks that are already known to be scalar-linearly solvable.

1.3.2 Network Capacity Regions

In our work concerning the network capacity regions, we continue the research along the lines of work by Cannons et al. [2]. Our contributions can be summarized as follows and we refer to Chapter 3 for details:

1. We define the network capacity region of networks and prove its notable properties: closedness, boundedness and convexity.

2. We show that the network routing capacity region is a computable rational polytope and provide exact algorithms and approximation heuristics for computing the region.

3. We define the semi-network linear coding capacity region that inner bounds the corresponding network linear coding capacity region, show that it is a computable rational polytope and provide exact algorithms and approximation heuristics for computing it.

While we present our results for the general directed acyclic networks, they generalize to directed networks with cycles and undirected networks. We note that the algorithms and heuristics we provide do have not polynomial running time in the input size. As our notion of the multi-dimensional network capacity region captures the notion of the single-dimensional network capacity in [2], our present work, in effect, addresses a few open problems proposed by Cannons et al. [2]: whether there exists an efficient algorithm for computing the network routing capacity and whether there exists an algorithm for computing the network linear coding capacity. It follows from our work that there exist combinatorial approximation algorithms for computing the network routing capacity and for computing a lower bound of the network linear coding capacity.

Chapter 2

Matroidal Networks Associated with Representable Matroids

In this chapter, we further study the matroidal networks introduced by Dougherty et al. [8]. We prove the converse of a theorem in [8] which states that, if a network is scalar-linearly solvable then it is a matroidal network associated with a representable matroid over a finite field. From [8] and our present work, it follows that a network is scalar-linearly solvable if and only if it is a matroidal network associated with a representable matroid over a finite field. The main idea of our work is to construct a scalar-linear network code from the network-matroid mapping between the matroid and network. Thereby, we show a correspondence between scalar-linearly solvable networks and representable matroids over finite fields in the framework of matroidal networks. It follows that determining scalar-linear solvability of a network $\mathcal{N}$ is equivalent to determining the existence of a representable matroid $\mathcal{M}$ over a finite field and a valid network-matroid mapping between $\mathcal{M}$ and $\mathcal{N}$. We also prove a theorem about the scalar-linear solvability of networks and field characteristics. Using our result and the matroidal network construction method due to Dougherty et al., we note that networks constructed from representable matroids over finite fields are scalar-linearly solvable. The constructed networks are potentially different from the classes of networks that are already known to be scalar-linearly solvable. It is possible that our approach provides a superset, but this is unknown at this time.

2.1 Definitions

Definition 7 (Global Linear Network Code). *A* global linear network code *is a 5-tuple* (F, k, n, ϕ_{msg}, ϕ_{edge}) *where*

1. F *is a finite field, with* $|F| \geq |\mathcal{A}|$,
2. k *is a source dimension,*
3. n *is an edge dimension,*
4. ϕ_{msg} *is the global coding vector function on messages,* $\phi_{msg} : \mu \rightarrow (F^{k\times k})^{|\mu|}$, *such that for message* m, $\phi_{msg}(m) = (M_1, \ldots, M_{|\mu|})^{\mathrm{T}}$ *where* M_i *is a* $k \times k$ *matrix over* F, *and*
5. ϕ_{edge} *is the global coding vector function on edges,* $\phi_{edge} : \epsilon \rightarrow (F^{n\times k})^{|\mu|}$, *such that for each edge* e, $\phi_{edge}(e) = (M_1, \ldots, M_{|\mu|})^{\mathrm{T}}$ *where* M_i *is a* $n \times k$ *matrix over* F.

Definition 8 (Global Linear Network Code Solution). *A global linear network code* (F, k, n, ϕ_{msg}, ϕ_{edge}) *is a* global linear network code solution, *if* $|F| \geq |\mathcal{A}|$ *and the following conditions are satisfied:*

1. *For each message* $m \in \mu$, $\phi_{msg}(m) = (0, \ldots, 0, I^{k\times k}, 0, \ldots, 0)^{\mathrm{T}}$ *where* $I^{k\times k}$ *is the* $k\times k$ *identity matrix over* F *and is in the coordinate corresponding to message* m.
2. *For each node* $x \in \nu$ *and edge* $e \in \mathrm{Out}(x)$, *if* $\phi_{edge}(e) = (M_1, \ldots, M_{|\mu|})^{\mathrm{T}}$, *then there exist matrices* $C_1, \ldots, C_{\alpha+\beta}$ *over* F *such that* $M_i = \sum_{j=1}^{\alpha+\beta} C_j M_i^j$, *for* $i = 1, \ldots, |\mu|$.
3. *For each node* $x \in \nu$ *and message* $m \in \mathrm{Out}(x)$, *if* $\phi_{msg}(m) = (M_1, \ldots, M_{|\mu|})^{\mathrm{T}}$, *then there exist matrices* $C'_1, \ldots, C'_{\alpha+\beta}$ *over* F *such that* $M_i = \sum_{j=1}^{\alpha+\beta} C'_j M_i^j$, *for* $i = 1, \ldots, |\mu|$.

Where, if $m_1, \ldots, m_\alpha$ *are messages generated by* x *and* $e_{\alpha+1}, \ldots, e_{\alpha+\beta}$ *are in-edges of* x, $\phi_{msg}(m_j) = (M_1^j, \ldots, M_{|\mu|}^j)^{\mathrm{T}}$ *for* $j = 1, \ldots, \alpha$ *and* $\phi_{edge}(e_j) = (M_1^j, \ldots, M_{|\mu|}^j)^{\mathrm{T}}$ *for* $j = \alpha+1, \ldots, \alpha+\beta$; $C_1, \ldots, C_\alpha$ *are* $n \times k$ *matrices and* $C_{\alpha+1}, \ldots, C_{\alpha+\beta}$ *are* $n \times n$ *matrices that would appear as coefficients in a linear edge function; and* $C'_1, \ldots, C'_\alpha$ *are* $k \times k$ *matrices and* $C'_{\alpha+1}, \ldots, C'_{\alpha+\beta}$ *are* $k \times n$ *matrices that would appear as coefficients in a linear decoding function.*

As with the network codes, we shall sometimes use the prefix (k, n) to emphasize the source and edge dimensions or omit k and n if they are clear from the context. It is straightforward

to check that the notions of linear network code solution and global linear network code solution are equivalent, as noted in previous works in algebraic network coding (for instance, [20] for the $k = n = 1$ case).

Proposition 9. *Let $\mathcal{N} = (\nu, \epsilon, \mu, \mathcal{A}, S, R)$ be a network. Then, $\mathcal{N}$ has a (k, n) linear network code solution if and only if it has a (k, n) global linear network code solution.*

Proof. Let $(F, k, n, \mathcal{F}_e, \mathcal{F}_d)$ be a (k, n) linear network code solution for $\mathcal{N}$. Since $\mathcal{N}$ is a directed acyclic graph, we can order nodes in ν with a topological sort so that each edge go from a lower-ranked node to a higher-ranked node. The ordering of nodes induces an ordering of edges $\hat{e}_1, \ldots, \hat{e}_{|\epsilon|}$ such that no path exists from $\hat{e}_i$ to $\hat{e}_j$ for $i > j$. We define ϕ_{msg} for all m and ϕ_{edge} for $\hat{e}_1, \ldots, \hat{e}_{|\epsilon|}$ in that order:

1. For each message m, we define $\phi_{msg}(m) = (0, \ldots, 0, I^{k\times k}, 0, \ldots, 0)^{\mathrm{T}}$ where $I^{k\times k}$ is the $k \times k$ identity matrix and is in the coordinate corresponding to m.

2. For each $\hat{e}_j = (x, y)$, the edge function $f_{\hat{e}_j}$ can be written as

$$f_{\hat{e}_j}(a(m_1), \ldots, a(m_\alpha), s(e_{\alpha+1}), \ldots, s(e_{\alpha+\beta})) = \sum_{l=1}^{\alpha} C_l \cdot a(m_l) + \sum_{l=\alpha+1}^{\alpha+\beta} C_l \cdot s(e_l),$$

   where $m_1, \ldots, m_\alpha$ are messages generated by x and $e_{\alpha+1}, \ldots, e_{\alpha+\beta}$ are in-edges of x; and $C_1, \ldots, C_\alpha$ are $n \times k$ matrices and $C_{\alpha+1}, \ldots, C_{\alpha+\beta}$ are $n \times n$ matrices over F. Let $\phi_{msg}(m_j) = (M_1^j, \ldots, M_{|\mu|}^j)^{\mathrm{T}}$ for $j = 1, \ldots, \alpha$ and $\phi_{edge}(e_j) = (M_1^j, \ldots, M_{|\mu|}^j)^{\mathrm{T}}$ for $j = \alpha + 1, \ldots, \alpha + \beta$. We define $\phi_{edge}(\hat{e}_j) = (M_1, \ldots, M_{|\mu|})^{\mathrm{T}}$ where $M_i = \sum_{l=1}^{\alpha+\beta} C_l \cdot M_i^l$.

Note that $s(\hat{e}_j) = \sum_{i=1}^{|\mu|} M_i \cdot a(m_i)$. By construction, $(F, k, n, \phi_{msg}, \phi_{edge})$ is a valid (k, n) global linear code that satisfies the first two properties of global linear network code solutions. We check the third property. For each $x \in \nu$ and $m \in R(x)$, the decoding function $f_{x,m}$ can be written as

$$f_{x,m}(a(m_1), \ldots, a(m_\alpha), s(e_{\alpha+1}), \ldots, s(e_{\alpha+\beta})) = \sum_{l=1}^{\alpha} C_l \cdot a(m_l) + \sum_{l=\alpha+1}^{\alpha+\beta} C_l \cdot s(e_l),$$

and $f_{x,m}(a(m_1), \ldots, a(m_\alpha), s(e_{\alpha+1}), \ldots, s(e_{\alpha+\beta})) = a(m)$. Note that $m_1, \ldots, m_\alpha$ are messages generated at x and $e_{\alpha+1}, \ldots, e_{\alpha+\beta}$ are in-edges of x; $C_1, \ldots, C_\alpha$ are $k \times k$ matrices and $C_{\alpha+1}, \ldots,$

$C_{\alpha+\beta}$ are $k \times n$ matrices. Let $\phi_{msg}(m_j) = (M_1^j, \ldots, M_{|\mu|}^j)^{\mathrm{T}}$ for $j = 1, \ldots, \alpha$ and $\phi_{edge}(e_j) = (M_1^j, \ldots, M_{|\mu|}^j)^{\mathrm{T}}$ for $j = \alpha+1, \ldots, \alpha+\beta$. It follows that $[\phi_{msg}(m)]_i = \sum_{l=1}^{\alpha+\beta} C_l \cdot M_i^l$ for all i, where $[\phi_{msg}(m)]_i$ denotes the i-th coordinate of $\phi_{msg}(m)$; $(F, k, n, \phi_{msg}, \phi_{edge})$ is a (k, n) global linear network code solution.

The converse direction is similar and so we only sketch the proof. Let $(F, k, n, \phi_{msg}, \phi_{edge})$ be a (k, n) global linear network code solution for $\mathcal{N}$. For each edge e, we define edge function f_e by

$$f_e(a(m_1), \ldots, a(m_\alpha), s(e_{\alpha+1}), \ldots, s(e_{\alpha+\beta})) = \sum_{l=1}^{\alpha} C_l \cdot a(m_l) + \sum_{l=\alpha+1}^{\alpha+\beta} C_l \cdot s(e_l),$$

where $C_1, \ldots, C_{\alpha+\beta}$ are some matrices satisfying Definition 8. For each $x \in \nu$ and $m \in R(x)$, we define decoding function $f_{x,m}$ similarly using matrices $C_1, \ldots, C_{\alpha+\beta}$ from Definition 8. □

We have the following corollaries from the definitions:

Corollary 10. *Let $\mathcal{N} = (\nu, \epsilon, \mu, \mathcal{A}, S, R)$ be a network. Then, $\mathcal{N}$ has a (k, k) vector-linear network code solution if and only if it has a (k, k) global vector-linear network code solution.*

Corollary 11. *Let $\mathcal{N} = (\nu, \epsilon, \mu, \mathcal{A}, S, R)$ be a network. Then, $\mathcal{N}$ has a scalar-linear network code solution if and only if it has a global scalar-linear network code solution.*

In this chapter, we will focus on scalar-linear network codes, that is linear network codes with $k = n = 1$.

2.2 Matroids

We define matroids and three classes of matroids. See [24] for more background on matroids.

Definition 12. *A matroid $\mathcal{M}$ is an ordered pair $(\mathcal{S}, \mathcal{I})$ consisting of a set $\mathcal{S}$ and a collection $\mathcal{I}$ of subsets of $\mathcal{S}$ satisfying the following conditions:*

1. *$\emptyset \in \mathcal{I}$;*
2. *If $I \in \mathcal{I}$ and $I' \subseteq I$, then $I' \in \mathcal{I}$;*
3. *If I_1 and I_2 are in $\mathcal{I}$ and $|I_1| < |I_2|$, then there is an element e of $I_2 \setminus I_1$ such that $I_1 \cup \{e\} \in \mathcal{I}$.*

The set $\mathcal{S}$ is called the *ground set* of the matroid $\mathcal{M}$. A subset X of $\mathcal{S}$ is an *independent set* if it is in $\mathcal{I}$; X is a *dependent set* if not. A *base* B of $\mathcal{M}$ is a maximal independent set; for all elements $e \in \mathcal{S} \setminus B$, $B \cup \{e\} \notin \mathcal{I}$. It can be shown that all bases have the same cardinality. A *circuit* of $\mathcal{M}$ is a minimal dependent set; for all elements e in C, $C \setminus \{e\} \in \mathcal{I}$. For each matroid, there is an associated function r called *rank* that maps the power set $2^{\mathcal{S}}$ into the set of nonnegative integers. The rank of a set $X \subseteq \mathcal{S}$ is the maximum cardinality of an independent set contained in X.

Definition 13 (Matroid Isomorphism). *Two matroids $\mathcal{M}_1 = (\mathcal{S}_1, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{S}_2, \mathcal{I}_2)$ are isomorphic if there is a bijection map ψ from $\mathcal{S}_1$ to $\mathcal{S}_2$ such that for all $X \subseteq \mathcal{S}_1$, X is independent in $\mathcal{M}_1$ if and only if $\psi(X)$ is independent in $\mathcal{M}_2$.*

Definition 14 (Uniform Matroids). *Let c, d be nonnegative integers such that $c \leq d$. Let $\mathcal{S}$ be a d-element set and $\mathcal{I}$ be the collection $\{X \subseteq \mathcal{S} : |X| \leq c\}$. We define the uniform matroid of rank c on the d-element set to be $U_{c,d} = (\mathcal{S}, \mathcal{I})$.*

Definition 15 (Graphic Matroids). *Let G be an undirected graph with the set of edges, $\mathcal{S}$. Let $\mathcal{I} = \{X \subseteq \mathcal{S} : X$ does not contain a cycle$\}$. We define the graphic matroid associated with G as $\mathcal{M}(G) = (\mathcal{S}, \mathcal{I})$.*

Definition 16 (Representable/Vector Matroid). *Let A be a $d_1 \times d_2$ matrix over some field F. Let $\mathcal{S} = \{1, \ldots, d_2\}$ where element i in $\mathcal{S}$ corresponds to the ith column vector of A and $\mathcal{I} = \{X \subseteq \mathcal{S} :$ corresponding column vectors form an independent set$\}$. We define the vector matroid associated with A as $\mathcal{M}(A) = (\mathcal{S}, \mathcal{I})$. A matroid $\mathcal{M}$ is F-representable if it is isomorphic to a vector matroid of some matrix over field F. A matroid is representable if it is representable over some field. Note that F is not necessarily finite.*

The bases of $U_{c,d} = (\mathcal{S}, \mathcal{I})$ are exactly subsets of $\mathcal{S}$ of cardinality c and the circuits are subsets of $\mathcal{S}$ of cardinality $c+1$. Each base of $\mathcal{M}(G)$ is a spanning forest of G, hence an union of spanning trees in connected components of G, and each circuit is a single cycle within a connected component. It is known that the graphic matroids are representable over any field F. On the other hand, the uniform matroid $U_{2,4}$ is not representable over $GF(2)$.

2.3 Matroidal Networks

We define matroidal networks and present a method for constructing matroidal networks from matroids; for more details and relevant results, we refer to [8].

Definition 17. *Let $\mathcal{N}$ be a network with message set μ, node set ν, and edge set ϵ. Let $\mathcal{M} = (\mathcal{S}, \mathcal{I})$ be a matroid with rank function r. The network $\mathcal{N}$ is a* matroidal network *associated with $\mathcal{M}$ if there exists a function $f : \mu \cup \epsilon \to \mathcal{S}$, called the* network-matroid mapping, *such that the following conditions are satisfied:*

1. *f is one-to-one on μ;*
2. *$f(\mu) \in \mathcal{I}$;*
3. *$r(f(\mathrm{In}(x))) = r(f(\mathrm{In}(x) \cup \mathrm{Out}(x)))$, for every $x \in \nu$.*

We define $f(A)$ to be $\{f(x) \mid x \in A\}$ for a subset A of $\mu \cup \epsilon$.

Theorem 18 (Construction Method)**.** *Let $\mathcal{M} = (\mathcal{S}, \mathcal{I})$ be a matroid with rank function r. Let $\mathcal{N}$ denote the network to be constructed, μ its message set, ν its node set, and ϵ its edge set. Then, the following construction method will construct a matroidal network $\mathcal{N}$ associated with $\mathcal{M}$. We do not address issues of complexity of the method.*

We choose the alphabet $\mathcal{A}$ to be any set with at least two elements. The construction will simultaneously construct the network $\mathcal{N}$, the network-matroid mapping $f : \mu \cup \epsilon \to \mathcal{S}$, and an auxiliary function $g : \mathcal{S} \to \nu$, where for each $x \in \mathcal{S}$, $g(x)$ is either

1. *a source node with message m and $f(m) = x$; or*
2. *a node with in-degree 1 and whose in-edge e satisfies $f(e) = x$.*

The construction is completed in 4 steps and each step can be completed in potentially many different ways:

Step 1: Choose any base $B = \{b_1, \ldots, b_{r(\mathcal{S})}\}$ of $\mathcal{M}$. Create network source nodes $n_1, \ldots, n_{r(\mathcal{S})}$ and corresponding messages $m_1, \ldots, m_{r(\mathcal{S})}$, one at each node. Let $f(m_i) = b_i$ and $g(b_i) = n_i$.

Step 2: (to be repeated until no longer possible).

Find a circuit $\{x_0, \ldots, x_j\}$ in $\mathcal{M}$ such that $g(x_1), \ldots, g(x_j)$ have been already defined but not $g(x_0)$. Then we add:

1. *a new node y and edges $e_1, \ldots, e_j$ such that e_i connects $g(x_i)$ to y. Let $f(e_i) = x_i$.*

2. *a new node n_0 with a single in-edge e_0 that connects y to n_0. Let $f(e_0) = x_0$ and $g(x_0) = n_0$.*

Step 3: (can be repeated arbitrarily many times).
If $\{x_0, \ldots, x_j\}$ is a circuit of $\mathcal{M}$ and $g(x_0)$ is a source node with message m_0, then add to the network a new receiver node y which demands the message m_0 and has in-edges $e_1, \ldots, e_j$ where e_i connects $g(x_i)$ to y. Let $f(e_i) = x_i$.
Step 4: (can be repeated arbitrarily many times).
Choose a base $B = \{x_1, \ldots, x_{r(S)}\}$ of $\mathcal{M}$ and create a receiver node y that demands all the network messages and has in-edges $e_1, \ldots, e_{r(S)}$ where e_i connects $g(x_i)$ to y. Let $f(e_i) = x_i$.

The following theorem is from [8]. The original theorem states with a representable matroid, but the same proof still works with a representable matroid over a finite field.

Theorem 19. *If a network is scalar-linearly solvable over some finite field, then the network is matroidal. Furthermore, the network is associated with a representable matroid over a finite field.*

2.4 Scalar-linear Solvability

We prove the converse of Theorem 19 and that a network is scalar-linearly solvable over a finite field of characteristic p if and only if the network is a matroidal network associated with a representable matroid over a finite field of characteristic p. In what follows, we assume that $d_2 \geq d_1$.

Lemma 20. *Let A be a $d_1 \times d_2$ matrix over a finite field F and $\mathcal{M}(A)$ be the corresponding representable matroid. Then, there exists an arbitrarily large finite field F' and a $d_1 \times d_2$ matrix A' over F' such that the corresponding matroid $\mathcal{M}(A')$ is isomorphic to $\mathcal{M}(A)$.*

Proof. We show that any finite field F' that contains F as a subfield works; for instance, extension fields of F. We consider the same matrix A over F', so choose $A' = A$, and show that a set of column vectors of A is independent over F if and only if it is independent over F'. Assume columns $v_1, \ldots, v_k$ are dependent by some scalars a_i's in F, $a_1 v_1 + \cdots + a_k v_k = 0$. Since F' contains F, all operations with elements of the subfield F stay in the subfield, and the same scalars still work in F', i.e., $a_1 v_1 + \cdots + a_k v_k = 0$ in F'. Hence, the vectors are dependent over F'. Assume column vectors $v_1, \ldots, v_k$ are independent over F. We extend the set of vectors to a basis of F^{d_1}. Then,

the matrix formed by the basis has a nonzero determinant over F. By similar reasons as before, the same matrix has a nonzero determinant when considered as a matrix over F'. Hence, the column vectors of the basis matrix are independent over F' and, in particular, the column vectors $v_1, \ldots, v_k$ are independent over F'. □

Theorem 21. *If a network $\mathcal{N}$ is matroidal and is associated with a representable matroid over a finite field F, then $\mathcal{N}$ is scalar-linearly solvable.*

Proof. Let $\mathcal{N} = (\nu, \epsilon, \mu, \mathcal{A}, S, R)$ be a matroidal network. Let A be the $d_1 \times d_2$ matrix over the finite field F such that $\mathcal{N}$ is a matroidal network associated with the corresponding matroid $\mathcal{M}(A) = (\mathcal{S}, \mathcal{I})$. By Lemma 20, we assume that the finite field F is large enough to represent all elements in $\mathcal{A}$, i.e., $|F| \geq |\mathcal{A}|$. By Definition 17, there exists a network-matroid mapping $f : \mu \cup \epsilon \to \mathcal{S}$. Assume $r(\mathcal{S}) = d_1$; otherwise, we remove redundant rows without changing the structure of the matroid. Let $f(\mu) = \{i_1, \ldots, i_{|\mu|}\}$. As $f(\mu) \in \mathcal{I}$, the columns indexed by $f(\mu)$ form an independent set. We extend $f(\mu)$ to a basis B of F^{d_1}, if necessary, by adding column vectors of A. Without loss of generality, assume the first d_1 columns of A form the basis B after reordering. By performing elementary row operations, we uniquely express A in the form

$$A = [I_{d_1} \mid A']$$

where A' is a $d_1 \times (d_2 - d_1)$ matrix and such that $\{i_1, \ldots, i_{|\mu|}\}$ now corresponds to the first $|\mu|$ columns of A. Note that the structure of the corresponding matroid stays the same. We introduce dummy messages $m_{|\mu|+1}, \ldots, m_{d_1}$, if necessary, by adding a disconnected node that generates these messages. We assign global coding vectors on the resulting $\mathcal{N}$ as follows:

1. for each edge e, let $\phi_{edge}(e) = A_{f(e)}$; and
2. for each message m, let $\phi_{msg}(m) = A_{f(m)}$,

where A_i denotes the i-th column of A. We show that the global linear network code defined above is valid and satisfies all the demands. For each node $x \in \nu$, we have $r(f(\mathrm{In}(x))) = r(f(\mathrm{In}(x) \cup \mathrm{Out}(x)))$. It follows that for each edge $e \in \mathrm{Out}(x)$, $A_{f(e)}$ is a linear combination of $\{A_{f(e')} : e' \in \mathrm{In}(x)\}$. Equivalently, $\phi_{edge}(e)$ is a linear combination of coding vectors in $\{\phi_{msg}(m) : m \in \mathrm{In}(x)\} \cup \{\phi_{edge}(e) : e \in \mathrm{In}(x)\}$. For each message $m \in \mathrm{Out}(x)$, $A_{f(m)}$ is a linear combination of

$\{A_{f(e')} : e' \in \mathrm{In}(x)\}$. Similarly, $\phi_{msg}(m)$ is a linear combination of coding vectors in $\{\phi_{msg}(m) : m \in \mathrm{In}(x)\} \cup \{\phi_{edge}(e) : e \in \mathrm{In}(x)\}$. Note, furthermore, that $\phi_{msg}(m)$ is the standard basis vector corresponding to m. It follows that the global linear network code $(F, \mathcal{F}_e, \mathcal{F}_d)$ thus defined is a global linear network code solution. Removing the dummy messages, it follows that $\mathcal{N}$ is scalar-linearly solvable. □

Given an arbitrary matrix A, assigning its column vectors as global coding vectors will not give a global linear network code solution necessarily. In essence, the theorem shows that, while we cannot use column vectors of A directly, we can do the described operations to produce an equivalent representation of A from which we can derive a global linear network code solution. From Theorems 18 and 21, we obtain a method for constructing scalar-linearly solvable networks: pick any representable matroid over a finite field F and construct a matroidal network $\mathcal{N}$ using Theorem 18. Combining Theorems 19 and 21, we obtain the following theorem.

Theorem 22. *A network is scalar-linearly solvable if and only if the network is a matroidal network associated with a representable matroid over a finite field.*

One implication of the theorem is that the class of scalar-linearly solvable networks in the algebraic network coding problem corresponds to the class of representable matroids over finite fields in the framework of matroidal networks. In effect, our results show a connection between scalar-linearly solvable networks, which are tractable networks for network coding, and representable matroids over finite fields, which are also particularly tractable in terms of description size.

In light of Dougherty et al.'s approach [7, 8], relationships between field characteristics and linear solvability of matroidal networks are important. In the case of scalar-linear network codes, we fully characterize a relationship with the following theorem. Note that a network might be a matroidal network with respect to more than one representable matroids of different field characteristics and, thus, is possibly scalar-linearly solvable with respect to fields of different characteristics.

Theorem 23. *A network is scalar-linearly solvable over a finite field of characteristic p if and only if the network is a matroidal network associated with a representable matroid over a finite field of characteristic p.*

Proof. We extend Theorems 19 and 21 and Lemma 20 to include field characteristic p, and the statement follows straightforwardly. □

Corollary 24. *Any matroidal network $\mathcal{N}$ associated with an uniform matroid is scalar-linearly solvable over a sufficiently large finite field of any characteristic. The same holds for the graphic matroids.*

Proof. It is straightforward to show that for any uniform matroid $\mathcal{M}$ and a prime p, there is a sufficiently large finite field F of characteristic p and a matrix A such that $\mathcal{M}$ is a representable matroid associated with A over F. The same is true for graphic matroids. □

As a consequence, any matroidal networks constructed from uniform or graphic matroids will not have interesting properties like those constructed from the Fano and non-Fano matroids in Dougherty et al. [7, 8].

2.5 Examples

In this section, we provide examples of scalar-linearly solvable networks that follow from Theorem 21. As mentioned before, we get a method for constructing scalar-linearly solvable networks from Theorems 18 and 21: pick any representable matroid over a finite field F and construct a matroidal network. We assume $\mathcal{A} = \{0, 1\}$ throughout this section.

The Butterfly network $\mathcal{N}_1$ in Fig. 2-1 is a matroidal network that can be constructed from the uniform matroid $U_{2,3}$. The ground set $\mathcal{S}$ of $U_{2,3}$ is $\{a, b, c\}$. Nodes 1-2 are the source nodes and nodes 5-6 are the receiver nodes. See Fig. 2-1 and Table 2.1 for details of the construction and a global scalar-linear network code solution. Note that the sets under 'Variables' column are order-sensitive.

Network $\mathcal{N}_2$ in Fig. 2-2 is a matroidal network constructed from the uniform matroid $U_{2,4}$. The ground set $\mathcal{S}$ of $U_{2,4}$ is $\{a, b, c, d\}$. Nodes 1 and 2 are the source nodes and nodes 7-9 are the receiver nodes. $U_{2,4}$ is a representable matroid associated with $A = \begin{bmatrix} 1 & 0 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{bmatrix}$ over $\mathbb{F}_3$ and, hence, it has a scalar-linear network code solution over $\mathbb{F}_3$. See Fig. 2-2 and Table 2.2 for details.

Consider the graph G and the matroidal network $\mathcal{N}_3$ constructed from $\mathcal{M}(G)$ in Fig. 2-3. The ground set $\mathcal{S}$ of $\mathcal{M}(G)$ is $\{1, \ldots, 7\}$, representing the edges of G. Nodes 1-4 are the source nodes and nodes 11-13 are the receiver nodes. $\mathcal{M}(G)$ is a representable matroid over field $\mathbb{F}_2$ and, by Theorem 21, the network has a scalar-linear network code solution over $\mathbb{F}_2$, as shown by the global

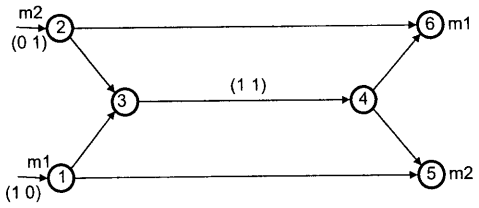

Figure 2-1: The Butterfly network $\mathcal{N}_1$ constructed from $U_{2,3}$

Step	Variables	Nodes	x	$g(x)$
1	$\{a, b\}$	n_1, n_2	a	n_1
			b	n_2
2	$\{c, a, b\}$	n_3, n_4	c	n_4
3	$\{b, a, c\}$	n_5		
3	$\{a, b, c\}$	n_6		
4	none	not used		

Table 2.1: Construction of the Butterfly network $\mathcal{N}_1$ from $U_{2,3}$.

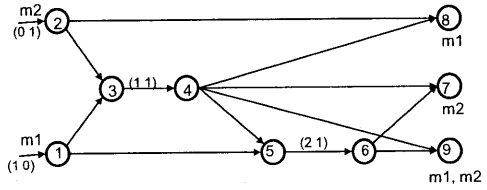

Figure 2-2: Network $\mathcal{N}_2$ constructed from $U_{2,4}$

Step	Variables	Nodes	x	$g(x)$
1	$\{a, b\}$	n_1, n_2	a	n_1
			b	n_2
2	$\{c, a, b\}$	n_3, n_4	c	n_4
2	$\{d, a, c\}$	n_5, n_6	d	n_6
3	$\{b, c, d\}$	n_7		
3	$\{a, b, c\}$	n_8		
4	$\{c, d\}$	n_9		

Table 2.2: Construction of $\mathcal{N}_2$ from $U_{2,4}$

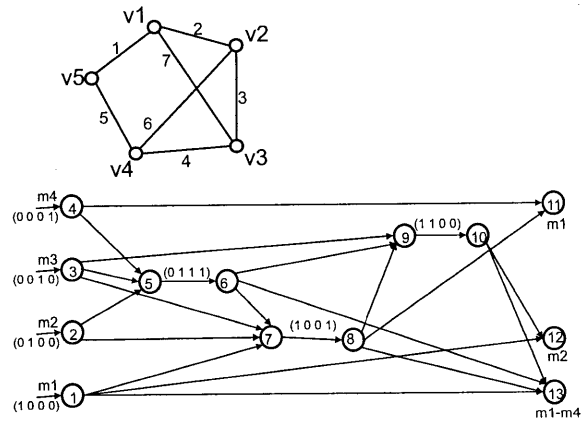

Figure 2-3: Graph G and network $\mathcal{N}_3$ constructed from $\mathcal{M}(G)$

Step	Variables	Nodes	x	$g(x)$
1	$\{3, 4, 5, 7\}$	$n_1 - n_4$	3	n_1
			4	n_2
			5	n_3
			7	n_4
2	$\{1, 4, 5, 7\}$	n_5, n_6	1	n_6
2	$\{2, 1, 3, 4, 5\}$	n_7, n_8	2	n_8
2	$\{6, 1, 2, 5\}$	n_9, n_{10}	6	n_{10}
3	$\{3, 2, 7\}$	n_{11}		
3	$\{4, 3, 6\}$	n_{12}		
4	$\{1, 2, 3, 6\}$	n_{13}		

Table 2.3: Construction of $\mathcal{N}_3$ from $\mathcal{M}(G)$ in Fig. 2-3

coding vectors on $\mathcal{N}_3$ in Fig. 2-3. This example shows that our results provide networks which are different from the networks previously known to be scalar-linearly solvable such as multicast, 2-level multicast and disjoint multicast networks. It is possible that network $\mathcal{N}_3$ can be constructed from a set of polynomials as in Dougherty et al. [9] or via index codes as in El Rouayheb et al. [10].

2.6 Discussion and Conclusion

In this chapter, we showed that any matroidal network associated with a representable matroid over a finite field is scalar-linearly solvable. Combined with an earlier result of Dougherty et al., it follows that a network is scalar-linearly solvable if and only if it is a matroidal network associated with a representable matroid over a finite field. It also follows that determining scalar-linear solvability of a network is equivalent to finding a representable matroid over a finite field and a valid network-matroid mapping. We also showed a relationship between scalar-linear solvability of networks and field characteristics. Moreover, we obtained a method for generating scalar-linearly solvable networks from representable matroids over finite fields and a set of scalar-linearly solvable networks that are possibly different from those networks that we already know are scalar-linearly solvable.

Unfortunately, the results presented in this chapter do not seem to generalize to vector-linear network coding or more general network coding schemes. The difficulty is that the matroid structure requires that a subset of the ground set of a matroid is either independent or dependent, but what this corresponds to in vector-linear codes, for instance, is not clear. Instead of vectors over fields, we now have vectors over rings (matrices over a field, to be more specific) in vector-linear network coding, and we are unaware of suitable matroids on vectors over rings for our purpose. In fact, El Rouayheb et al. [10] also made a similar observation and suggested that FD-relations are more related to networks than are matroids.

Chapter 3

Network Capacity Regions

In this chapter, we study the network capacity region of networks along the lines of work by Cannons et al. [2]. We define the network capacity region of networks analogously to the rate regions in information theory and show its notable properties: closedness, boundedness and convexity. In the case of routing, we prove that the network routing capacity region $\mathcal{C}_r$ is a computable rational polytope and provide exact algorithms and approximation heuristics for computing the region. In the case of linear network coding, we define an auxiliary region $\mathcal{C}_l'$, called the semi-network linear coding capacity region, which is a computable rational polytope that inner bounds the network linear coding capacity region, and provide exact algorithms and approximation heuristics for computing $\mathcal{C}_l'$. More specifically, we show the network routing capacity region $\mathcal{C}_r$ ($\mathcal{C}_l'$) is an image of a higher-dimensional rational polytope under an affine map and consider the computation of $\mathcal{C}_r$ ($\mathcal{C}_l'$) as the polytope reconstruction problem with a ray oracle. Our results generalize to directed networks with cycles and undirected networks. We note that some underlying problems associated with the polytope reconstruction problem are NP-hard, such as the minimum cost directed Steiner tree problem, and that algorithms and heuristics we present are not polynomial time schemes. Rather, the algorithms and heuristics may have exponential running time in the input size, depending on the intermediate computations and the resulting output's description size.

As our notion of the multi-dimensional network capacity region captures the notion of the single-dimensional network capacity in [2], our present work, in effect, addresses a few open problems proposed by Cannons et al. [2]: whether there exists an efficient algorithm for computing the network routing capacity and whether there exists an algorithm for computing the network linear

coding capacity. Computing the single-dimensional network capacity is equivalent to computing a point on the boundary of the multi-dimensional network capacity region and a ray starting at the origin and reduces to solving associated linear programs in the case of the network routing capacity and (a lower bound of) network linear coding capacity. It follows from our work that there exist combinatorial approximation algorithms for computing the network routing capacity and for computing a lower bound of the network linear coding capacity.

A polytope has two equivalent descriptions; the vertex description in terms of vertices, or the extreme points, of the polytope and the hyperplane description in terms of linear inequalities defining facets of the polytope. In this work, we do not distinguish the two descriptions and use them interchangeably, but we note that converting one description into another can be computationally expensive; we use vertex enumeration algorithms to convert a hyperplane description into a vertex one and facet enumeration algorithms (essentially, convex hull algorithms) for the conversion in the other direction. See [12, 14, 18] for more details on polytopes and relevant algorithms.

In this chapter, we consider nondegenerate networks where, for each demand of a message at a receiver node, there is a path from a source node generating the message to the receiver node and where no message is both generated and demanded by the same node.

3.1 Fractional Network Coding Model

We define a fractional network coding model. Most definitions are adapted from Cannons et al.[2]. We use the notations where we write vectors or points in a multi-dimensional space with a hat as in $\hat{k}$ and let k_i denote the i-th coordinate of the vector $\hat{k}$. We also use $[\hat{k}]_i$ and $\hat{k}(i)$ to denote the i-th coordinate to avoid confusions when necessary. When it is clear from the context, we omit the hat to avoid cluttering symbols.

Definition 25 (Capacitated Network). *A* capacitated network $\mathcal{N}$ *is a finite, directed, acyclic multigraph given by a 7-tuple* $(\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ *where*

1. ν *is a node set,*

2. ϵ *is an edge set,*

3. μ *is a message set,*

4. $c : \epsilon \to \mathbb{Z}^+$ *is an edge capacity function,*

5. $\mathcal{A}$ *is an alphabet,*

6. $S : \nu \to 2^\mu$ *is a source mapping, and*

7. $R : \nu \to 2^\mu$ *is a receiver mapping.*

As we shall use only capacitated networks in this chapter, we use $\mathcal{N}$ to denote a capacitated network. We refer to networks defined in Chapter 1 as *ordinary networks.* We assume that the messages in μ are indexed as $m_1, \ldots, m_{|\mu|}$. We define fractional edge function, fractional decoding function, and fractional message assignment with respect to a finite field F, where $|F| \geq |\mathcal{A}|$, a source dimension vector $\hat{k}$, and an edge dimension n:

Definition 26 (Fractional Edge and Fractional Decoding Functions). *Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a capacitated network and $m_1, \ldots, m_{|\mu|}$ be the messages. Let $\hat{k} = (k_1, \ldots, k_{|\mu|})$ be a vector of positive integers and n be a positive integer. For each edge $e = (x, y)$, a* fractional edge function *is a map*

$$f_e : (F^{k_{i_1}}) \times \cdots \times (F^{k_{i_\alpha}}) \times (F^{nc(e_{\alpha+1})}) \times \cdots \times (F^{nc(e_{\alpha+\beta})}) \to F^{nc(e)},$$

where $m_{i_1}, \ldots, m_{i_\alpha}$ are α messages generated by x and $e_{\alpha+1}, \ldots, e_{\alpha+\beta}$ are β in-edges of x. For each node $x \in \nu$ and message $m_j \in R(x)$, a fractional decoding function *is a map*

$$f_{x,m_j} : (F^{k_{i_1}}) \times \cdots \times (F^{k_{i_\alpha}}) \times (F^{nc(e_{\alpha+1})}) \times \cdots \times (F^{nc(e_{\alpha+\beta})}) \to F^{k_j},$$

where $m_{i_1}, \ldots, m_{i_\alpha}$ are α messages generated by x and $e_{\alpha+1}, \ldots, e_{\alpha+\beta}$ are β in-edges of x.

We call $\hat{k} = (k_1, \ldots, k_{|\mu|})$ the source dimension vector, *where k_i is the source dimension for message m_i, and n the* edge dimension. *We denote the collections of fractional edge and fractional decoding functions by $\mathcal{F}_e = \{f_e : e \in \epsilon\}$ and $\mathcal{F}_{x,m} = \{f_{x,m} : x \in \nu, m \in R(x)\}$, respectively.*

Definition 27 (Fractional Message Assignment). *Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a capacitated network and $m_1, \ldots, m_{|\mu|}$ be the messages. A* fractional message assignment *is a collection of maps $\overline{a} = (a_1, \ldots, a_{|\mu|})$ where a_i is a message assignment for m_i, $a_i : m_i \to F^{k_i}$.*

Definition 28 (Fractional Network Code). *Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a capacitated network and $m_1, \ldots, m_{|\mu|}$ be the messages in μ. A* fractional network code *on $\mathcal{N}$ is a 5-tuple $(F, \hat{k}, n, \mathcal{F}_e, \mathcal{F}_d)$*

where

1. *F is a finite field, with $|F| \geq |\mathcal{A}|$,*

2. *$\hat{k} = (k_1, \ldots, k_{|\mu|})$ is a source dimension vector,*

3. *n is an edge dimension,*

4. *$\mathcal{F}_e$ is a collection of fractional edge functions on $\mathcal{N}$,*

5. *$\mathcal{F}_d$ is a collection of fractional decoding functions on $\mathcal{N}$.*

As with the ordinary network codes in Chapter 1, we have different kinds of fractional network codes defined analogously: *fractional routing network codes*, *fractional linear network codes*, and *fractional nonlinear network codes.* We shall use the prefix $(\hat{k}, n)$ before codes to emphasize the parameters $\hat{k}$ and n.

Definition 29 (Fractional Network Code Solution). *Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a capacitated network and $m_1, \ldots, m_{|\mu|}$ be the messages. A fractional network code $(F, \hat{k}, n, \mathcal{F}_e, \mathcal{F}_d)$ is a* fractional network code solution, *or* fractional solution *for short, if for every fractional message assignment $\overline{a} = (a_1, \ldots, a_{|\mu|})$,*

$$f_{x,m_j}(a_{i_1}(m_{i_1}), \ldots, a_{i_\alpha}(m_{i_\alpha}), s(e_{\alpha+1}), \ldots, s(e_{\alpha+\beta})) = a_j(m_j),$$

for all $x \in \nu$ and $m_j \in R(x)$. Note that $m_{i_1}, \ldots, m_{i_\alpha}$ are α messages generated by x and $e_{\alpha+1}, \ldots, e_{\alpha+\beta}$ are β in-edges of x. If the above equation holds for a particular $x \in \nu$ and message $m \in R(x)$, then we say node x's demand m is satisfied.

As with network code solutions for ordinary networks, we have special classes of fractional network code solutions: *fractional routing network code solutions*, *fractional linear network code solutions*, and *fractional nonlinear network code solutions.* When it is clear from the context, we refer to them by appropriate abridged versions from time to time.

If $(F, \hat{k}, n, \mathcal{F}_e, \mathcal{F}_d)$ is a fractional network code solution for $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$, source node $x \in \nu$ sends a vector of k_i symbols from F for each message $m_i \in S(x)$; each receiver node $x \in \nu$ demands the original vector of k_i symbols corresponding to message m_i for each $m_i \in R(x)$; and each edge e carries a vector of $c(e)n$ symbols. We refer to coordinates of the symbol vector of

length k_i corresponding to message m_i as *message m_i's coordinates.* Note that each coordinate of a message is independent from others. We use coordinates and symbols for messages interchangeably; the i-th symbol of message m refers to the i-th coordinate of the message. We refer to coordinates of the symbol vector of length $c(e)n$ on edge e as *edge e's coordinates.* Note that a coordinate of edge e can be *active*, meaning it actively carries a symbol in the fractional network code solution, or *inactive*, meaning it is not used in the solution. For instance, if an edge with 5 available coordinates has to send 2 independent symbols through the edge, then it suffices to use only 2 coordinates; in this case, the coordinates that carry symbols are active and the other 3 coordinates are inactive.

We define a notion of *minimal network code solutions* as follows:

Definition 30 (Minimal Fractional Network Code Solution). *A fractional network code solution $(F, \hat{k}, n, \mathcal{F}_e, \mathcal{F}_d)$ for $\mathcal{N}$ is* minimal *if the set A of all active coordinates of edges in the solution is minimal, i.e., there exists no $(\hat{k}, n)$ fractional network code solution for $\mathcal{N}$ with the set of active coordinates that is a strict subset of A.*

3.2 Network Capacity Regions

3.2.1 Definitions

Definition 31 (Achievable Coding Rate Vector). *Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a capacitated network. A vector of positive numbers $\left(\frac{k_1}{n}, \ldots, \frac{k_{|\mu|}}{n}\right) \in \mathbb{Q}_+^{|\mu|}$ is an* achievable coding rate vector *if there exists a fractional network code solution $(F, \hat{k}, n, \mathcal{F}_e, \mathcal{F}_d)$ for $\mathcal{N}$ where $\hat{k} = (k_1, \ldots, k_{|\mu|})$.*

Definition 32 (Network Capacity Region). *Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a capacitated network and $m_1, \ldots, m_{|\mu|}$ be the messages. The* network capacity region *$\mathcal{C}$ of $\mathcal{N}$ is the closure of all achievable coding rate vectors in $\mathbb{R}^{|\mu|}$,*

$$\mathcal{C} = \text{closure}\left\{\frac{\hat{k}}{n} : \frac{\hat{k}}{n} = \left(\frac{k_1}{n}, \ldots, \frac{k_{|\mu|}}{n}\right) \text{ is an achievable coding rate vectors}\right\}.$$

By definition, a network capacity region is a set of points in the Euclidean space $\mathbb{R}_+^{|\mu|}$.

There are different classes of achievable coding rate vectors and, hence, corresponding classes of network capacity regions: the *network routing capacity region, $\mathcal{C}_r$,* which is the closure of all

achievable routing rate vectors; the *network linear coding capacity region, $\mathcal{C}_l$,* which is the closure of all achievable linear coding rate vectors; and the *network nonlinear coding capacity region, $\mathcal{C}$,* which is the closure of all achievable nonlinear coding rate vectors (or the network capacity region, equivalently).

3.2.2 Properties

We show that the network capacity regions are closed, bounded and convex sets and satisfy an additional property.

Theorem 33. *Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a capacitated network and $m_1, \ldots, m_{|\mu|}$ be the messages. The corresponding network capacity region $\mathcal{C}$ is a closed, bounded and convex set in $\mathbb{R}_+^{|\mu|}$.*

Proof. (Closedness) By definition, $\mathcal{C}$ is a closure of a set and, hence, closed.

(Boundedness) We show that $\frac{k_i}{n}$ is bounded for all i in the achievable coding rate vector $\left(\frac{k_1}{n}, \ldots, \frac{k_{|\mu|}}{n}\right)$. By symmetry, it suffices to show for $\frac{k_1}{n}$. Let n be the edge dimension, ν_1 be the set of nodes in ν that generate message m_1 and γ be the sum of capacities of out-edges of nodes in ν_1. Then, $k_1 \leq \gamma n$ as we cannot send more than γn independent coordinates of message m_1 and expect receivers to recover all the information. Hence, $\frac{k_1}{n} \leq \gamma$. It follows that $\mathcal{C}$ is bounded.

(Convexity) Let $x_0, x_1 \in \mathcal{C}$ and $\lambda \in [0, 1]$. We show that $x = (1 - \lambda)x_0 + \lambda x_1 \in \mathcal{C}$. We write $x = x_0 + \lambda(x_1 - x_0)$. There exists sequences of achievable coding rate vectors converging to x_0 and x_1, say $\{y_{0,j}\}$ and $\{y_{1,j}\}$ respectively. Let $\{\lambda_j\}$ be a sequence of rationals converging to λ. Then, $y_j = y_{0,j} + \lambda_j(y_{1,j} - y_{0,j})$ is an achievable coding rate vector for $j = 1, 2, \ldots$. Let $\lambda_j = \frac{p}{q}$, $y_{0,j} = \left(\frac{k_1}{n}, \ldots, \frac{k_{|\mu|}}{n}\right)$ and $y_{1,j} = \left(\frac{k'_1}{n'}, \ldots, \frac{k'_{|\mu|}}{n'}\right)$. Then,

$$y_j = \left(\frac{(q-p)k_1n' + pk'_1n}{qnn'}, \ldots, \frac{(q-p)k_{|\mu|}n' + pk'_{|\mu|}n}{qnn'}\right).$$

There exists a fractional network code solution $(F, \hat{k}, qnn', \mathcal{F}_e, \mathcal{F}_d)$ where $\hat{k} = ((q-p)k_1n' + pk'_1n, \ldots, (q-p)k_{|\mu|}n' + pk'_{|\mu|}n)$; if NC_1 and NC_2 are two fractional network code solutions with rate vectors $y_{0,j}$ and $y_{1,j}$, then for first $(q-p)nn'$ coordinates we employ $(q-p)n'$ copies of NC_1

and for the remaining pnn' coordinates we employ pn copies of NC_2. Then,

$$\begin{aligned}|x - y_j| &= |x_0 + \lambda(x_1 - x_0) - y_{0,j} - \lambda_j(y_{1,j} - y_{0,j})| \\ &= |(1-\lambda)x_0 - (1-\lambda_j)y_{0,j} + \lambda x_1 - \lambda_j y_{1,j}| \\ &\leq |(1-\lambda)x_0 - (1-\lambda_j)y_{0,j}| + |\lambda x_1 - \lambda_j y_{1,j}|.\end{aligned}$$

Since $\lambda_j \to \lambda$ and $y_{0,j} \to x_0$, $(1-\lambda_j)y_{0,j}$ converges to $(1-\lambda)x_0$ and $|(1-\lambda)x_0 - (1-\lambda_j)y_{0,j}|$ can be made arbitrarily small for sufficiently large j. Similarly, $|\lambda x_1 - \lambda_j y_{1,j}|$ can be made arbitrarily small for sufficiently large j. It follows that the sequence of achievable coding rate vectors $\{y_j\}$ converges to x and that $x \in \mathcal{C}$. □

Corollary 34. *Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a capacitated network and $m_1, \ldots, m_{|\mu|}$ be the messages. The corresponding network routing capacity region, $\mathcal{C}_r$, and network linear coding capacity region, $\mathcal{C}_l$, are closed, bounded and convex regions in $\mathbb{R}_+^{|\mu|}$.*

We note that the network capacity regions are of very special kind by definition:

Proposition 35. *The network capacity region $\mathcal{C}$ is a region such that if $\hat{r} = (r_1, \ldots, r_{|\mu|}) \in \mathcal{C}$, then the parallelepiped $[0, r_1] \times \ldots \times [0, r_{|\mu|}]$ is contained in $\mathcal{C}$. The same holds for the network routing capacity region $\mathcal{C}_r$ and the network linear coding capacity region $\mathcal{C}_l$.*

We use $\mathrm{bd}\,\mathcal{C}$ to denote the boundary of the network capacity regions. Similarly, we use $\mathrm{bd}\,\mathcal{C}_r$ and $\mathrm{bd}\,\mathcal{C}_l$ to denote the boundaries of corresponding regions.

3.3 Network Routing Capacity Regions

In this section, we prove that the network routing capacity region is a bounded rational polytope, and provide exact algorithms and approximation heuristics for computing it. Since multi-source multi-sink networks can be reduced to multiple multicast networks, it suffices to show the results with respect to the multiple multicast networks; for each message m, we add a "super source node" that generates the message m and connects to source nodes that generate m via edges of infinite, or sufficiently large, capacities. We assume that the given networks in this section are multiple multicast networks for simpler presentation of results.

3.3.1 Properties

Theorem 36. *The network routing capacity region $\mathcal{C}_r$ is a bounded rational polytope in $\mathbb{R}_+^{|\mu|}$ and is computable.*

Proof. (Polytope) It suffices to consider minimal fractional routing solutions since any fractional routing solution can be reduced to a minimal one by successively making unnecessary active edge coordinates inactive. For each coordinate of a message m, it suffices to route it along a Steiner tree rooted at the source node of m and spanning all the receiver nodes demanding m. Hence, any minimal fractional routing solution consists of routing messages along Steiner trees. Let $\mathcal{T}_i$ be the set of all Steiner trees rooted at the source node of message m_i and spanning all receiver nodes that demand m_i, and $\mathcal{T}$ be the union, $\mathcal{T} = \mathcal{T}_1 \cup \ldots \cup \mathcal{T}_{|\mu|}$. Note that $\mathcal{T}$ is a finite set. Then, any minimal fractional routing solution $(F, \hat{k}, n, \mathcal{F}_e, \mathcal{F}_d)$ satisfies the following constraints:

$$\begin{array}{lcll} \sum_{T\in\mathcal{T}} T(e)\cdot x(T) & \leq & c(e)\cdot n, & \forall e \in \epsilon \\ \sum_{T\in\mathcal{T}_i} x(T) & = & k_i, & \forall 1 \leq i \leq |\mu| \\ x & \geq & 0, & \end{array}$$

where $x(T)$ is the number of times Steiner tree T is used in the solution and $T(e)$ is an indicator that is 1 if the Steiner tree T uses the edge e, or 0 otherwise. Dividing all variables $x(T)$ by n, we obtain

$$\begin{array}{lcll} \sum_{T\in\mathcal{T}} T(e)\cdot x(T) & \leq & c(e), & \forall e \in \epsilon \\ \sum_{T\in\mathcal{T}_i} x(T) & = & \frac{k_i}{n}, & \forall 1 \leq i \leq |\mu| \\ x & \geq & 0. & \end{array}$$

It follows that all minimal fractional routing solutions, after scaling by n, satisfy

$$\begin{array}{lcll} \sum_{T\in\mathcal{T}} T(e)\cdot x(T) & \leq & c(e), & \forall e \in \epsilon \\ x & \geq & 0. & \end{array}$$

As the coefficients are in $\mathbb{Q}$, the above set of inequalities defines a bounded rational polytope $\mathcal{P}$, with rational extreme points, in $\mathbb{R}_+^{|\mathcal{T}|}$. The polytope is bounded, because edge capacities are finite and no Steiner tree can be used for routing for infinitely many times. Each minimal fractional routing solution reduces to a rational point inside the polytope $\mathcal{P}$, and each rational point x inside

$\mathcal{P}$ has a minimal fractional routing solution $(F, \hat{k}, n, \mathcal{F}_e, \mathcal{F}_d)$ that reduces to it, such that

$$\left(\frac{k_1}{n}, \ldots, \frac{k_{|\mu|}}{n}\right) = \left(\sum_{T \in \mathcal{T}_1} x(T), \ldots, \sum_{T \in \mathcal{T}_{|\mu|}} x(T)\right).$$

To see the latter statement, we take a rational point in $\mathcal{P}$, put rationals under a common denominator, and choose appropriate $\hat{k}$ and n. As rational points are dense, the closure of the rational points corresponding to minimal fractional routing solutions is exactly $\mathcal{P}$. It follows that the network routing capacity region $\mathcal{C}_r$ is the image of $\mathcal{P}$ under the affine map

$$\psi_r : (x(T))_{T \in \mathcal{T}} \mapsto \left(\sum_{T \in \mathcal{T}_1} x(T), \ldots, \sum_{T \in \mathcal{T}_{|\mu|}} x(T)\right).$$

As the affine map preserves rationality, it follows that the network routing capacity region is a bounded rational polytope in $\mathbb{R}_+^{|\mu|}$.

(Computability) We show that we can compute the vertex description (the extreme points) of the polytope $\mathcal{C}_r$. We compute the vertices $v_1, \ldots, v_h$ of polytope $\mathcal{P}$ by any vertex enumeration algorithm where the starting point can be any point that corresponds to using a single Steiner tree for the maximum number of times allowed by the network. We compute the images of the vertices of $\mathcal{P}$ under the affine map ψ_r. The network routing capacity region is given by the vertices of the convex hull of points $\psi_r(v_1), \ldots, \psi_r(v_h)$ in $\mathbb{R}_+^{|\mu|}$. □

The network routing capacity defined by Cannons et al.[2] corresponds to a point on the boundary of polytope $\mathcal{C}_r$; it is exactly the intersection point between the (outer) boundary $\mathrm{bd}\,\mathcal{C}_r$ and the ray $\hat{x} = (1, \ldots, 1)t, t \geq 0$. As the ray has a rational slope, the intersection point is rational and, hence, Corollary IV.6 in [2] follows straightforwardly. We use $\mathcal{P}_r$ to denote the "parent" polytope, in Theorem 36, of the network routing capacity region $\mathcal{C}_r$.

3.3.2 Algorithms

We provide exact algorithms and approximation heuristics for computing the network routing capacity region $\mathcal{C}_r$. This subsection goes together with next two subsections, so we advise the reader to refer to these subsections as necessary. We assume that a capacitated network $\mathcal{N}$ is given if not stated explicitly. We already provided an exact algorithm for computing $\mathcal{C}_r$ in the

proof of Theorem 36, which we refer to as Algorithm `VertexEnum_Route`. The algorithm takes the hyperplane description of $\mathcal{P}_r$ and outputs the hyperplane description of $\mathcal{C}_r$. Since the polytope $\mathcal{P}_r$ is defined in a high dimensional space $\mathbb{R}_+^{|\mathcal{T}|}$ where $|\mathcal{T}|$ could be exponential in the description size of networks, Algorithm `VertexEnum_Route` may not be efficient in practice as there could be exponentially many vertices.

Algorithm 1 Algorithm `VertexEnum_Route`($\mathcal{N}$)

1: Form the hyperplane description of polytope $\mathcal{P}_r$.
2: Compute the vertices of $\mathcal{P}_r$ with a vertex enumeration algorithm and obtain $v_1, \ldots, v_h$.
3: Compute the image of the vertices, $\psi_r(v_1), \ldots, \psi_r(v_h)$.
4: **return** convex hull of $\psi_r(v_1), \ldots, \psi_r(v_h)$.

To design more efficient algorithms and approximation heuristics, we recast the computation of the network routing capacity region as the polytope reconstruction problem with a ray oracle and use related results in literature [5, 15]. More specifically, we formulate the reconstruction problem as follows:

Definition 37 (Polytope Reconstruction Problem). *Let $\mathcal{Q}$ be a polytope in $\mathbb{R}^d$ that contains the origin in its interior and $\mathcal{O}_{Ray}$ be a ray oracle that given a ray of the form $\hat{x} = \hat{r}t, t \geq 0$, computes the intersection point between the ray and the boundary of $\mathcal{Q}$. Compute a polytope description of $\mathcal{Q}$ using a finite number of calls to the oracle $\mathcal{O}_{Ray}$.*

We reduce the computation of the network routing capacity region $\mathcal{C}_r$ to a polytope reconstruction problem by 1) reflecting $\mathcal{C}_r$ around the origin to get a symmetric polytope $\mathcal{Q}$ in $\mathbb{R}^{|\mu|}$ that contains the origin in its interior and 2) solving the linear programs similar to the one in Cannons et al. [2] to implement the ray oracle $\mathcal{O}_{Ray}$. To reflect $\mathcal{C}_r$, we map all calls to the ray oracle to equivalent calls with rays defined in $\mathbb{R}_+^{|\mu|}$. We use the algorithm outlined in Section 5 of Gritzmann et al. [15] to compute all the facets of the resulting polytope $\mathcal{Q}$ and recover the facets of the network routing capacity region $\mathcal{C}_r$. We refer to the overall algorithm as Algorithm `FacetEnum_Route`. The main idea of the algorithm is to first find a polytope $\mathcal{Q}'$ that contains $\mathcal{Q}$ and whose facet-defining hyperplanes are a subset of those for $\mathcal{Q}$ (Theorem 5.3 in [15]), and then successively add more facet-defining hyperplanes of $\mathcal{Q}$ to $\mathcal{Q}'$ by using $\mathcal{O}_{Ray}$. In other words, we start with a polytope that contains $\mathcal{Q}$ and successively shrink it until it becomes $\mathcal{Q}$. By Theorem 5.5 in Gritzmann et al. [15],

we need at most

$$f_0(\mathcal{Q}) + (|\mu| - 1)f^2_{|\mu|-1}(\mathcal{Q}) + (5|\mu| - 4)f_{|\mu|-1}(\mathcal{Q})$$

calls to the ray oracle $\mathcal{O}_{Ray}$ to compute the facets, where $f_i(\mathcal{Q})$ denotes the number of i-dimensional faces of $\mathcal{Q}$ (the 0-th dimensional faces being the points). Because of the symmetries around the origin, we need at most

$$f_0(\mathcal{C}_r) + (|\mu| - 1)f^2_{|\mu|-1}(\mathcal{C}_r) + (5|\mu| - 4)f_{|\mu|-1}(\mathcal{C}_r)$$

calls to the ray oracle where $f_i(\mathcal{C}_r)$ denotes the number of i-dimensional faces of $\mathcal{C}_r$ that do not contain the origin.

Algorithm 2 Algorithm `FacetEnum_Route`($\mathcal{N}$, $\mathcal{O}_{Ray}$)

1: Form the hyperplane description of $\mathcal{P}_r$ in $\mathbb{R}_+^{|\mathcal{T}|}$.
2: Internally, reflect $\mathcal{C}_r$ around the origin to get the polytope $\mathcal{Q}$ in $\mathbb{R}^{|\mu|}$.
3: Using $\mathcal{O}_{Ray}$, compute a polytope $\mathcal{Q}'$ containing $\mathcal{Q}$.
4: **while** $\mathcal{Q}'$ has undetermined facets **do**
5: Compute the vertices of $\mathcal{Q}'$.
6: Using $\mathcal{O}_{Ray}$, compute the intersection points on rays defined by the vertices of $\mathcal{Q}'$.
7: Add newly found facet-defining hyperplanes of $\mathcal{Q}$ to $\mathcal{Q}'$.
8: **end while**
9: Retrieve facets of $\mathcal{C}_r$.
10: **return** the facet description of $\mathcal{C}_r$.

Depending on the implementation of $\mathcal{O}_{Ray}$, we get exact algorithms and approximation heuristics for computing $\mathcal{C}_r$. If we use an exact algorithm for the ray oracle $\mathcal{O}_{Ray}$, we get an exact hyperplane description of the network routing capacity region via Algorithm `FacetEnum_Route`. If instead we use an approximation algorithm for the oracle that computes some point r such that the actual intersection point lies between r and Ar, then we obtain approximation heuristics that compute a set of points r such that the boundary $\mathrm{bd}\,\mathcal{C}_r$ lies between points r and Ar. We note that an approximation algorithm for $\mathcal{O}_{Ray}$ does not necessary work with Algorithm `FacetEnum_Route` to give an approximation algorithm for $\mathcal{C}_r$, where an A-approximation of $\mathcal{C}_r$ would be a polytope $\mathcal{P}$ such that $\mathcal{P} \subset \mathcal{C}_r \subset A\mathcal{P}$. While an approximation algorithm for the oracle $\mathcal{O}_{Ray}$ does not necessarily lead to a polytope description of $\mathcal{C}_r$, it might be faster and more efficient than exact algorithms and, hence, more applicable to compute a quick "sketch" of the capacity region $\mathcal{C}_r$. One

approximation heuristic for computing the region $\mathcal{C}_r$ would be to take a sufficiently large number of rays evenly spread apart throughout the space $\mathbb{R}_+^{|\mu|}$ and use an approximate oracle $\mathcal{O}_{Ray}$ to find the approximate intersection points. As there are many simple variations of this approach, we do not go into the details of the heuristics themselves in this work.

3.3.3 Implementations of Exact and Approximate Oracle $\mathcal{O}_{Ray}$

In this subsection, we provide both exact and approximation algorithms for the ray oracle $\mathcal{O}_{Ray}$ used in Algorithm `FacetEnum_Route`. The implementations of the oracle reduce to solving a linear program. We use any linear programming algorithms, such as the ellipsoid algorithm and simplex algorithm, to solve the linear program exactly and obtain an exact oracle $\mathcal{O}_{Ray}$. For the approximate ray oracles, we design a combinatorial approximation algorithm using techniques by Garg and Könemann [13]. Alternatively, we could use the ellipsoid algorithm with an approximate separation oracle, but as the ellipsoid algorithm is slow in practice, this might not be a viable approach. As the network routing capacity region $\mathcal{C}_r$ is a rational polytope, it suffices to consider rays with a rational slope in $\mathbb{Q}_+^{|\mu|}$.

Algorithms

Given the hyperplane description of the polytope $\mathcal{P}_r$,

$$\begin{array}{lll} \sum_{T\in\mathcal{T}} T(e)\cdot x(T) & \leq & c(e), \quad \forall e\in\epsilon \\ x & \geq & 0, \end{array}$$

and a ray with a rational slope, $\hat{x} = \hat{q}t, t \geq 0$, we want to compute the rational intersection point of the ray and the boundary of $\mathcal{C}_r$. It is straightforward to see that the intersection point is exactly $\lambda_{max}\hat{q}$ where λ_{max} is the optimal value to the linear program

$$\begin{array}{lll} \max & \lambda & \\ \text{s.t.} & \sum_{T\in\mathcal{T}} T(e)\cdot x(T) \leq c(e), & \forall e\in\epsilon \\ & \sum_{T\in\mathcal{T}_i} x(T) \geq \lambda q_i, & \forall 1\leq i\leq|\mu| \\ & x,\lambda \geq 0. & \end{array} \tag{3.3.1}$$

Since the coefficients of the linear program are rational, the optimal value λ_{max} and the corresponding solution x are also rational. While written in a different form, the linear program is equivalent to the one in Cannons et al. [2] when the ray is $\hat{x} = (1, \ldots, 1)t, t \geq 0$. We use any linear programming algorithm, such as the ellipsoid algorithm and simplex algorithm, to solve the above linear program exactly and obtain Algorithm `OracleRayExact_Route`. We note that the running time of Algorithm `OracleRayExact_Route` could be poor as the linear program has exponentially many variables (one for each Steiner tree) and the associated separation problem is NP-hard (the minimum cost directed Steiner tree problem).

Algorithm 3 Algorithm `OracleRayExact_Route`($\mathcal{N}$, $\hat{q}$)

1: Form the linear program (3.3.1) corresponding to $\mathcal{N}$ and $\hat{q}$.
2: Solve the linear program with a linear programming algorithm and obtain λ_{max}.
3: **return** $\lambda_{max}\hat{q}$.

We now provide a combinatorial approximation algorithm for solving the linear program (3.3.1) and, hence, for oracle $\mathcal{O}_{Ray}$. It computes a point $\hat{r}$ such that $\lambda_{max}\hat{q}$ is on the line segment between $\hat{r}$ and $(1+\omega)A\hat{r}$ for some numbers $\omega > 0$ and $A \geq 1$. The main idea is to view solving (3.3.1) as concurrently packing Steiner trees according to the ratio defined by $\hat{q}$, and use the results for the multicommodity flow and related problems by Garg and Könemann [13]. Instead of a shortest path algorithm, we use a minimum cost directed Steiner tree algorithm for our purpose. We assume we have an oracle $\mathcal{O}_{DSteiner}$ that solves the minimum cost directed Steiner tree problem, which is well-known to be NP-hard, within an approximation guarantee A:

Definition 38 (Minimum Cost Directed Steiner Tree Problem). *Given an acyclic directed multigraph $\mathcal{G} = (\nu, \epsilon)$, a length function $l : \epsilon \to \mathbb{R}^+$, a source node s and receiver nodes $n_1, \ldots, n_k$, find a minimum cost subset of edges ϵ' such that there is a directed path from s to each n_i in ϵ'. The cost of a subset ϵ' is $\sum_{e \in \epsilon'} l(e)$.*

First, we consider networks with exactly one message to route. In this case, the linear program (3.3.1) reduces to the following simpler linear program:

$$\begin{array}{lllll} \max & \sum_{T \in \mathcal{T}} x(T) & & & \\ \text{s.t.} & \sum_{T \in \mathcal{T}} T(e) \cdot x(T) & \leq & c(e), & \forall e \in \epsilon \\ & x & \geq & 0, & \end{array} \tag{3.3.2}$$

where $\mathcal{T}$ is the set of all Steiner trees rooted at the source node of the message and spanning all the receiver nodes demanding the message. Note that the original problem now reduces to the fractional directed Steiner tree packing problem (compare to the fractional Steiner tree packing problem in [17]). To solve (3.3.2), we use Algorithm `DSteinerTreePacking` which is a straightforward modification of the maximum multicommodity flow algorithm given in Section 2 of Garg and Könemann [13]. In Algorithm `DSteinerTreePacking`, mincost(l) denotes the cost of the (approximate) minimum cost directed Steiner tree found by $\mathcal{O}_{DSteiner}$.

Algorithm 4 Algorithm `DSteinerTreePacking`($\mathcal{N}$, ω, $O_{DSteiner}$, A)

$\eta = \frac{3}{16}\omega$; $\delta = (1+\eta)((1+\eta)L)^{-1/\eta}$
$f = 0$; $l(e) = \delta, \forall e \in \epsilon$
while mincost(l) $< A$ **do**
  Use $\mathcal{O}_{DSteiner}$ to compute an approximate minimum cost Steiner tree $\widetilde{T}$, under the length l.
  $d = \min_{e:\widetilde{T}(e)=1} c(e)$
  $f = f + d$
  Update $l(e) = l(e)\left(1 + \eta\frac{d}{c(e)}\right), \forall e$ s.t. $\widetilde{T}(e) = 1$.
end while
return $f/\log_{1+\eta}\frac{1+\eta}{\delta}$

Essentially the same analysis in [13] works for Algorithm `DSteinerTreePacking` except that our approximation guarantee is worse by a factor of A since we use an approximate oracle $\mathcal{O}_{DSteiner}$. We omit the analysis and summarize the performance of the algorithm as follows. For computations involving η and ω, we refer to Appendix A.1.1.

Theorem 39. *For $0 < \omega < 1$, Algorithm* `DSteinerTreePacking` *computes a $(1+\omega)A$-approximate solution to the linear program* (3.3.2) *in time $O(\omega^{-2}|\epsilon| \log L \cdot T_{DSteiner})$, where L is the maximum number of edges used in any Steiner tree in $\mathcal{T}$ and $T_{DSteiner}$ is the time required by oracle $\mathcal{O}_{DSteiner}$ to solve the minimum cost directed Steiner tree problem within an approximation guarantee of $A \geq 1$. Note that $L \leq |\epsilon|$.*

We now give Algorithm `OracleRayApprox_Route`, for oracle $\mathcal{O}_{Ray}$, which computes a $(1+\omega)A$-approximate solution to the linear program (3.3.1). It uses Algorithm `DSteinerTreePacking` as a subroutine.

Algorithm 5 Algorithm `OracleRayApprox_Route`($\mathcal{N}$, ω, $\mathcal{O}_{DSteiner}$, A, $\hat{q}$)

1: Using Algorithm `DSteinerTreePacking`, compute the approximate value z_i to the linear program (3.3.2) for each message m_i separately.
2: $z = \min_i \frac{z_i}{q_i}$; $scaling\_factor = \frac{|\mu|}{z}$
3: $\hat{q} = scaling\_factor \cdot \hat{q}$
4: $\eta = \frac{1}{9A}\omega$; $\delta = (|\epsilon|/(1-\eta A))^{-1/\eta A}$
5: $N = 2\left\lceil \frac{1}{\eta A} \log_{1+\eta} \frac{|\epsilon|}{1-\eta A} \right\rceil$
6: **while true do**
7: $t = 0$; $l(e) = \frac{\delta}{c(e)}, \forall e \in \epsilon$
8: **for** phase $i = 1, \ldots, N$ **do**
9: **for** iteration $j = 1, \ldots, |\mu|$ **do**
10: $\gamma = q_j$
11: **while** $\gamma > 0$ **do**
12: Using $\mathcal{O}_{DSteiner}$, compute the minimum cost directed Steiner tree $\widetilde{T} \in T_j$, under l.
13: $d = \min\{\gamma, c(e) : \widetilde{T}(e) = 1\}$
14: $\gamma = \gamma - d$
15: Update $l(e) = l(e)\left(1 + \eta\frac{d}{c(e)}\right), \forall e$ such that $\widetilde{T}(e) = 1$.
16: **if** $\sum_e l(e)c(e) \geq 1$ **then**
17: Goto Line 25.
18: **end if**
19: **end while**
20: **end for**
21: $t = t + 1$
22: **end for**
23: $scaling\_factor = 2scaling\_factor$; $\hat{q} = 2\hat{q}$
24: **end while**
25: **return** $scaling\_factor \cdot t / \log_{1+\eta} \frac{1}{\delta}$

Analysis

While Algorithm `OracleRayApprox_Route` is closely related to the maximum concurrent flow algorithm given in Section 5 of Garg and Könemann [13], there are significant differences and we give an analysis of the algorithm below. The linear program (3.3.1) can be thought of as the "concurrent" fractional directed Steiner tree packing problem. We have directed Steiner trees partitioned into different groups according to the network messages. Then, computing the optimal value of the linear program (3.3.1) is equivalent to fractionally routing along the Steiner trees so that the ratios among the overall usages of groups of Steiner trees correspond to the ratios among the coordinates of the $\hat{q}$ vector.

The dual linear program corresponding to (3.3.1) is

$$\begin{array}{lll} \min & \sum_e l(e)c(e) & \\ \text{s.t.} & \sum_{e:T(e)=1} l(e) - z(i) \leq 0, & \forall 1 \leq i \leq |\mu|, \forall T \in \mathcal{T}_i \\ & \sum_{i=1}^{|\mu|} q_i z(i) \geq 1 & \\ & l, z \geq 0. & \end{array} \tag{3.3.3}$$

From the dual linear program, we note that the associated separation problem is exactly the minimum cost directed Steiner tree problem. Let $\text{mincost}_i(l)$ denote the cost of the minimum cost directed Steiner tree in $\mathcal{T}_i$ under the length function l. Define $D(l) = \sum_{e \in \epsilon} l(e)c(e)$ and $\alpha(l) = \sum_{i=1}^{|\mu|} q_i \, \text{mincost}_i(l)$. Then, solving the dual linear program is equivalent to finding an assignment of lengths to edges, $l : \epsilon \rightarrow \mathbb{R}^+$, so as to minimize $\frac{D(l)}{\alpha(l)}$. Let β be the optimal value of the dual linear program, i.e., $\beta = \min_l \frac{D(l)}{\alpha(l)}$.

We first consider a modified version of Algorithm `OracleRayApprox_Route`, with the infinite while-loop in line 6 removed, with variable *scaling_factor* and line 23 removed, and with the finite for-loop in line 8 replaced with an infinite while-loop. The following holds:

Theorem 40. *Assume $\beta \geq 1$. For $0 < \omega < 1$, Algorithm* `OracleRayApprox_Route`, *with the modifications, returns a $(1+\omega)A$-approximate solution to the linear program* (3.3.1) *in at most $\left\lceil \frac{\beta}{\eta A} \log_{1+\eta} \frac{|\epsilon|}{1-\eta A} \right\rceil$ number of phases.*

Proof. Algorithm `OracleRayApprox_Route` in lines 8-22 proceeds in phases which in turn consist of $|\mu|$ iterations. Each iteration consists of variably many number of steps of the while-loop in line 11,

depending on how quickly γ is decreasing. In each j-th iteration in line 9, q_j units of message m_j are routed using Steiner trees in $\mathcal{T}_j$. By lines 16-18, the algorithm terminates as soon as $D(l) \geq 1$. Let t be the last phase in which the algorithm terminates. Let $l_{i,j}^{s-1}$ denote the length function l and $\gamma_{i,j}^{s-1}$ the variable γ at the start of s-th step of the while-loop (line 11) of j-th iteration in phase i. Let $\widetilde{T}_{i,j}^{s}$ denote the Steiner tree $\widetilde{T}$ selected in the s-th step of j-th iteration in phase i. Let $l_{i,0}$ be the length function at the start of phase i and $l_{i,|\mu|}$ be the length function at the end of phase i (after the termination of the $|\mu|$-th iteration). Note that $l_{i+1,0} = l_{i,|\mu|}$. Let $l_{i,j-1}$ be the length function at the start of iteration j of phase i. For simplicity, we denote $D(l_{i,|\mu|})$ and $\alpha(l_{i,|\mu|})$ by $D(i)$ and $\alpha(i)$, respectively. Then,

$$\begin{aligned} D(l_{i,j}^{s}) &= \sum_{e} l_{i,j}^{s}(e) \cdot c(e) \\ &= D(l_{i,j}^{s-1}) + \eta(\gamma_{i,j}^{s-1} - \gamma_{i,j}^{s}) \sum_{e:\widetilde{T}_{i,j}^{s}(e)=1} l_{i,j}^{s-1}(e) \\ &\leq D(l_{i,j}^{s-1}) + \eta(\gamma_{i,j}^{s-1} - \gamma_{i,j}^{s}) A \operatorname{mincost}_j(l_{i,j}^{s-1}). \end{aligned}$$

Then,

$$\begin{aligned} D(l_{i,j+1}) &\leq D(l_{i,j}) + \eta A \sum_{s} (\gamma_{i,j}^{s-1} - \gamma_{i,j}^{s}) \operatorname{mincost}_{j+1}(l_{i,j+1}) \\ &= D(l_{i,j}) + \eta A q_{j+1} \operatorname{mincost}_{j+1}(l_{i,j+1}) \\ &\leq D(l_{i,j}) + \eta A q_{j+1} \operatorname{mincost}_{j+1}(l_{i,|\mu|}), \end{aligned}$$

where we used the fact that $\operatorname{mincost}_j(l_{i,j}^{s}) \leq \operatorname{mincost}_j(l_{i,j+1})$ as the length function and mincost are nondecreasing in any fixed argument throughout the algorithm. After summing up the above

inequalities over the iterations of phase i, we get

$$\begin{aligned} D(l_{i,|\mu|}) &\leq D(l_{i,|\mu|-1}) + \eta A q_{|\mu|} \operatorname{mincost}_{|\mu|}(l_{i,|\mu|}) \\ &\leq D(l_{i,|\mu|-2}) + \eta A(q_{|\mu|-1} \operatorname{mincost}_{|\mu|-1}(l_{i,|\mu|}) + q_{|\mu|} \operatorname{mincost}_{|\mu|}(l_{i,|\mu|})) \\ &\vdots \\ &\leq D(l_{i,0}) + \eta A \sum_{j=1}^{|\mu|} q_j \operatorname{mincost}_j(l_{i,|\mu|}), \end{aligned}$$

and it follows that

$$D(i) \leq D(i-1) + \eta A \alpha(i).$$

Since $\frac{D(i)}{\alpha(i)} \geq \beta$, it follows that $D(i) \leq \frac{D(i-1)}{1-\eta A/\beta}$. Since $D(0) = |\epsilon|\delta$, we have for $i \geq 1$,

$$\begin{aligned} D(i) &\leq \frac{|\epsilon|\delta}{(1-\eta A/\beta)^i} \\ &= \frac{|\epsilon|\delta}{1-\eta A/\beta}\left(1 + \frac{\eta A}{\beta - \eta A}\right)^{i-1} \\ &\leq \frac{|\epsilon|\delta}{1-\eta A/\beta} e^{\frac{\eta A(i-1)}{\beta-\eta A}} \\ &\leq \frac{|\epsilon|\delta}{1-\eta A} e^{\frac{\eta A(i-1)}{\beta(1-\eta A)}}, \end{aligned}$$

where the last inequality uses the assumption that $\beta \geq 1$. The algorithm terminates in phase t for which $D(t) \geq 1$. Hence,

$$1 \leq D(t) \leq \frac{|\epsilon|\delta}{1-\eta A} e^{\frac{\eta A(t-1)}{\beta(1-\eta A)}}.$$

And it follows that

$$\frac{\beta}{t-1} \leq \frac{\eta A}{(1-\eta A)\ln\frac{1-\eta A}{|\epsilon|\delta}}. \tag{3.3.4}$$

In the first $t-1$ phases, we have routed $(t-1)q_j$ units of message m_j, for $j = 1, \ldots, |\mu|$. This routing solution may violate the edge capacity constraints, but, by the following claim, we obtain a feasible solution with $\lambda > \frac{t-1}{\log_{1+\eta} 1/\delta}$. See Claim 5.1 in [13] for the proof of the claim as the same proof works here.

Claim 41. *There exists a feasible solution with* $\lambda > \frac{t-1}{\log_{1+\eta}\frac{1}{\delta}}$.

Thus, the ratio of the values of the optimal dual and feasible primal solutions, ζ, is strictly less than $\frac{\beta}{t-1}\log_{1+\eta}\frac{1}{\delta}$. By (3.3.4), we get

$$\zeta < \frac{\eta A}{1-\eta A}\frac{\log_{1+\eta}\frac{1}{\delta}}{\ln\frac{1-\eta A}{|\epsilon|\delta}} = \frac{\eta A}{(1-\eta A)\ln(1+\eta)}\frac{\ln 1/\delta}{\ln\frac{1-\eta A}{|\epsilon|\delta}}.$$

For $\delta = (|\epsilon|/(1-\eta A))^{-1/\eta A}$, the ratio $\frac{\ln 1/\delta}{\ln\frac{1-\eta A}{|\epsilon|\delta}}$ equals $(1-\eta A)^{-1}$ and, hence,

$$\zeta \leq \frac{\eta A}{(1-\eta A)^2\ln(1+\eta)} \leq \frac{\eta A}{(1-\eta A)^2(\eta-\eta^2/2)} \leq (1-\eta A)^{-3}A.$$

For $\eta = \frac{1}{9A}\omega$, $(1-\eta A)^{-3}A$ is at most our desired approximation ratio $(1+\omega)A$ (see Appendix A.1.2 for details). By weak-duality, we have

$$1 \leq \zeta < \frac{\beta}{t-1}\log_{1+\eta}\frac{1}{\delta}$$

and, therefore, the number of phases in line 8 is strictly less than $1+\beta\log_{1+\eta}1/\delta$, which implies that Algorithm `OracleRayApprox_Route` terminates in at most $\left\lceil \frac{\beta}{\eta A}\log_{1+\eta}\frac{|\epsilon|}{1-\eta A}\right\rceil$ number of phases. □

Note that by Theorem 40, the running time of Algorithm `OracleRayApprox_Route`, with the modifications, depends on β. Note that β can be reduced/increased by scaling the $\hat{q}$ vector/capacities appropriately. We now remove the assumption $\beta \geq 1$ and analyze the running time of Algorithm `OracleRayApprox_Route` as a whole.

Theorem 42. *For $0 < \omega < 1$, Algorithm* `OracleRayApprox_Route` *computes a $(1+\omega)A$-approximate solution to the linear program (3.3.1) in time $O(\omega^{-2}(|\mu|\log A|\mu| + |\epsilon|)A\log|\epsilon| \cdot T_{DSteiner})$, where $T_{DSteiner}$ is the time required to solve the minimum cost directed Steiner tree problem with oracle $\mathcal{O}_{DSteiner}$ within an approximation guarantee A.*

Proof. To remove the dependency of the running time on β, we update the $\hat{q}$ vector and variable *scaling_factor* appropriately. Let z_i^* be the exact maximum fractional Steiner tree packing value for message m_i in line 1 and let $z^* = \min_i \frac{z_i^*}{q_i}$. Then, z^* is an upper bound on the maximum rate at which the messages can be routed in a minimal fractional routing solution. Since $z_i \leq z_i^* \leq Az_i$, $\frac{z}{|\mu|} \leq \beta \leq Az$. We scale the $\hat{q}$ vector as in line 3 so that $1 \leq \beta \leq A|\mu|$. The assumption $\beta \geq 1$ is satisfied, but β could now be as large as $A|\mu|$. We employ the doubling trick as explained in Section

5.2 in [13] and line 23 accomplishes this, together with $N = 2\left\lceil \frac{1}{\eta A} \log_{1+\eta} \frac{|\epsilon|}{1-\eta A} \right\rceil$ in the for-loop in line 8. Since we halve the "current" value of β after every N phases, the total number of phases is at most $N \log A|\mu|$. Since there are $|\mu|$ iterations per phase, there are at most $N|\mu| \log A|\mu|$ iterations in total. Each iteration consists of variably many number of steps and, in all steps but the last, we increase the length of an edge by a factor of $1+\eta$. By similar reasons as in the proof of Claim 5.1 in [13], the length of each edge can increase by a factor of $1+\eta$ at most $\log_{1+\eta} \frac{1}{\delta}$ times throughout the algorithm. Hence, the total number of steps of the while-loop in line 11 exceeds the total number of iterations by at most $|\epsilon| \log_{1+\eta} \frac{1}{\delta}$. The total number of steps, hence calls to the oracle $\mathcal{O}_{DSteiner}$, is at most $N|\mu| \log A|\mu| + |\epsilon| \log_{1+\eta} \frac{1}{\delta}$.

Note that $\eta = \Theta(\frac{\omega}{A})$. Then,

$$
\begin{aligned}
N|\mu| \log A|\mu| + |\epsilon| \log_{1+\eta} \frac{1}{\delta} &= N|\mu| \log A|\mu| + \frac{|\epsilon|}{\eta A} \log_{1+\eta} \frac{|\epsilon|}{1-\eta A} \\
&= O\left(\frac{|\mu| \log A|\mu|}{\eta A} \log_{1+\eta} \frac{|\epsilon|}{1-\eta A} + \frac{|\epsilon|}{\eta A} \log_{1+\eta} \frac{|\epsilon|}{1-\eta A} \right) \\
&= O\left(\frac{|\mu| \log A|\mu| + |\epsilon|}{\eta A} \cdot \frac{\ln |\epsilon| + \ln(1/(1-\eta A))}{\ln(1+\eta)} \right) \\
&\leq O\left(\frac{|\mu| \log A|\mu| + |\epsilon|}{\eta A} \cdot \frac{\ln |\epsilon| + \ln(1/(1-\eta A))}{\eta - \eta^2/2} \right) \\
&= O\left(\frac{|\mu| \log A|\mu| + |\epsilon|}{\omega} \cdot \frac{\ln |\epsilon| + \ln 1/(1-\omega)}{\omega/A - (\omega/A)^2/2} \right) \\
&= O(\omega^{-2}(|\mu| \log A|\mu| + |\epsilon|) A \log |\epsilon|)
\end{aligned}
$$

□

3.3.4 Implementations of Oracle $\mathcal{O}_{DSteiner}$

The oracle $\mathcal{O}_{DSteiner}$ solves the minimum cost directed Steiner tree problem (Definition 38). A brute force approach is to loop through all possible subsets of ϵ and select one with the minimum cost that satisfies the directed Steiner tree conditions. Since checking whether a subset of edges supports a directed path from the source to each receiver node can be done in $O(|\nu| + |\epsilon|)$ time, the brute force approach has the total running time of $O((|\nu| + |\epsilon|)2^{|\epsilon|})$ and finds an exact optimal solution. We can also compute an $O(k)$-approximate solution by computing a shortest path from the source to each receiver and combining the paths to form a tree, where k is the number of receivers. A shortest

path can be computed efficiently and there are many shortest path algorithms. For instance, the Dijkstra algorithm suffices for our purpose and gives an $O(k)$-approximate solution in time $O(|\nu|(|\nu|+|\epsilon|)\log|\nu|)$ with a binary minheap. There exists an efficient approximation algorithm for the minimum cost directed Steiner tree problem with a significantly better approximation guarantee by Charikar et al. [4]. Charikar et al. designed a family of algorithms that achieves an approximation ratio of $i(i-1)k^{1/i}$ in time $O(n^i k^{2i})$ for any integer $i > 1$, where n is the number of nodes and k is the number of receivers. For our problem, $k \leq |\nu|$ and $n \leq |\nu|$ and we get algorithms that achieve an approximation ratio of $i(i-1)|\nu|^{1/i}$ in time $O(|\nu|^{3i})$ for any integer $i > 1$. For $i = \log|\nu|$, we obtain an approximation ratio of $O(\log^2|\nu|)$ in time of $O(|\nu|^{3\log|\nu|})$. We summarize with Table 3.1. Note the tradeoff between the approximation ratio A and the running time.

Table 3.1: Implementations of oracle $\mathcal{O}_{DSteiner}$

Algorithm for $\mathcal{O}_{DSteiner}$	Approximation Ratio A	Time
BruteForce	1	$O((\|\nu\|+\|\epsilon\|)2^{\|\epsilon\|})$
ShortestPathApproximation	$O(\|\nu\|)$	$O((\|\nu\|^2+\|\nu\|\|\epsilon\|)\log\|\nu\|)$
Charikar et al. [4]	$i(i-1)\|\nu\|^{1/i}$	$O(\|\nu\|^{3i})$
	$O(\log^2\|\nu\|)$	$O(\|\nu\|^{3\log\|\nu\|})$

3.4 Network Linear Coding Capacity Regions

In this section, we show a computable inner bound on the network linear coding capacity region $\mathcal{C}_l$ with respect to a given finite field. In particular, we show how to compute a polytope $\mathcal{C}'_l$, which we call the *semi-network linear coding capacity region*, that is contained in $\mathcal{C}_l$. If $\mathcal{C}'_l$ is strictly bigger than $\mathcal{C}_r$, then linear coding helps improve the information throughput through the network. It is unknown at this time how good of an approximation the polytope $\mathcal{C}'_l$ is to the actual network linear coding capacity region $\mathcal{C}_l$. Unlike in the computation of the network routing capacity region, the finite field is important in the computation of $\mathcal{C}'_l$. We assume that a network $\mathcal{N}$, not necessarily a multiple multicast network as in Section 3.3, and a finite field F are given in what follows, if not stated explicitly.

3.4.1 Definitions

Definition 43 (Weight Vectors and Partial Scalar-Linear Network Code Solutions). *Let $\mathcal{N}$ be a network with unit edge capacities and $m_1, \ldots, m_{|\mu|}$ be the messages. The* weight vectors associated with $\mathcal{N}$, *or simply* weight vectors, *are vectors w in $\{0,1\}^{|\mu|}$ such that there exists a scalar-linear network code solution for $\mathcal{N}$ when only messages m_i with $w_i = 1$ are considered, i.e., for $\mathcal{N}$ with the new message set $\mu' = \{m_i : w_i = 1\}$. We refer to the scalar-linear network code solutions corresponding to these weight vectors as* partial scalar-linear network code solutions, *or* partial scalar-linear solutions *for short.*

Note that by definition, Steiner trees are also partial scalar-linear network code solutions.

Definition 44 (Simple Fractional Linear Network Code Solution). *Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a capacitated network and $m_1, \ldots, m_{|\mu|}$ be the messages. A fractional network code $(F, \hat{k}, n, \mathcal{F}_e, \mathcal{F}_d)$ is a* simple fractional linear network code solution, *or* simple fractional linear solution *for short, if the fractional network code is linear over the finite field F and can be decomposed into a set of partial scalar-linear solutions of $\mathcal{N}$ (when considered with unit edge capacities).*

Definition 45 (Semi-Network Linear Coding Capacity Region). *Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a capacitated network and $m_1, \ldots, m_{|\mu|}$ be the messages. The* semi-network linear coding capacity region *$\mathcal{C}_l'$ of $\mathcal{N}$ is the closure of all coding rate vectors achievable by simple fractional linear network code solutions. Note $\mathcal{C}_l' \subset \mathbb{R}_+^{|\mu|}$.*

Clearly, the network linear coding capacity region $\mathcal{C}_l$ contains the semi-network linear coding capacity region $\mathcal{C}_l'$ as the set of fractional linear code solutions is a superset of the set of simple fractional linear code solutions.

3.4.2 Properties

Theorem 46. *Assume a finite field F is given. The semi-network linear coding capacity region $\mathcal{C}_l'$, with respect to F, is a bounded rational polytope in $\mathbb{R}_+^{|\mu|}$ and is computable.*

Proof. (Polytope) The proof is similar to the proof of Theorem 36. Let $\mathcal{N} = (\nu, \epsilon, \mu, c, \mathcal{A}, S, R)$ be a network. Let $w_1, \ldots, w_{k'}$ be all the weight vectors associated with $\mathcal{N}$. Let $\mathcal{W}_i$ be the set of all partial scalar-linear network code solutions that satisfy the demands corresponding to the

weight vector w_i and $\mathcal{W}$ be the union, $\mathcal{W} = \mathcal{W}_1 \cup \ldots \cup \mathcal{W}_{k'}$. Note that $\mathcal{W}_i$'s are finite nonempty disjoint sets. Then, any simple fractional linear code solution $(F, \hat{k}, n, \mathcal{F}_e, \mathcal{F}_d)$ can be decomposed into partial scalar-linear solutions in $\mathcal{W}$ and satisfies the following constraints:

$$\begin{array}{lcll} \sum_{W \in \mathcal{W}} W(e) \cdot x(W) & \leq & c(e) \cdot n, & \forall e \in \epsilon \\ \sum_{i=1}^{k'} \sum_{W \in \mathcal{W}_i} [w_i]_j x(W) & = & k_j, & \forall 1 \leq j \leq |\mu| \\ x & \geq & 0, & \end{array}$$

where $x(W)$ is the number of times the partial scalar-linear solution W is used in the fractional linear solution and $W(e)$ is an indicator that is 1 if the solution W uses edge e, or 0 otherwise. After dividing all the variables $x(W)$ by n, it follows that all simple fractional linear code solutions satisfy

$$\begin{array}{lcll} \sum_{W \in \mathcal{W}} W(e) \cdot x(W) & \leq & c(e), & \forall e \in \epsilon \\ x & \geq & 0. & \end{array} \tag{3.4.1}$$

Using the affine map

$$\psi_l : (x(W))_{W \in \mathcal{W}} \mapsto \left(\sum_{i=1}^{k'} \sum_{W \in \mathcal{W}_i} [w_i]_1 x(W), \ldots, \sum_{i=1}^{k'} \sum_{W \in \mathcal{W}_i} [w_i]_{|\mu|} x(W) \right),$$

we follow the similar lines of reasoning as in Theorem 36 to show that the semi-network linear coding capacity region $\mathcal{C}_l'$ is a bounded rational polytope in $\mathbb{R}_+^{|\mu|}$.

(Computability) We apply the same proof for $\mathcal{C}_r$ to $\mathcal{C}_l'$ using the inequalities in (3.4.1). □

We use $\mathcal{P}_l'$ to denote the "parent" polytope of $\mathcal{C}_l'$ defined by (3.4.1).

3.4.3 Algorithms

We obtain algorithms and heuristics for computing the semi-network linear coding capacity region $\mathcal{C}_l'$ from algorithms and heuristics in Section 3.3.3 with little modifications; we use the polytope description of $\mathcal{P}_l$ instead of $\mathcal{P}_r$ and use the ray oracle $\mathcal{O}_{Ray}$ for $\mathcal{C}_l'$. We denote the resulting algorithms by Algorithms `VertexEnum_LCode` and `FacetEnum_LCode`. We omit the details of the algorithms. This subsection goes together with next two subsections.

3.4.4 Implementations of Exact and Approximate Oracle $\mathcal{O}_{Ray}$

Algorithms

We provide the implementations of oracle $\mathcal{O}_{Ray}$ for the semi-network linear coding capacity region $\mathcal{C}'_l$. As the region $\mathcal{C}'_l$ is a rational polytope, it suffices to consider rays with a rational slope. Given the hyperplane description of the polytope $\mathcal{P}'_l$,

$$\begin{array}{lll} \sum_{W\in\mathcal{W}} W(e)\cdot x(W) & \leq & c(e), \quad \forall e\in\epsilon \\ x & \geq & 0, \end{array}$$

and a ray with a rational slope of the form $\hat{x} = \hat{q}t, t \geq 0$, we would like to compute the rational intersection point of the ray and the boundary of polytope $\mathcal{C}'_l$. The intersection point is $\lambda_{max}\hat{q}$ where λ_{max} is the optimal value to the linear program:

$$\begin{array}{lll} \max & \lambda & \\ \text{s.t.} & \sum_{W\in\mathcal{W}} W(e)\cdot x(W) & \leq c(e), \quad \forall e\in\epsilon \\ & \sum_{i=1}^{k'}\sum_{W\in\mathcal{W}_i}[w_i]_j x(W) & \geq \lambda q_j, \quad \forall 1\leq j\leq|\mu| \\ & x,\lambda & \geq 0, \end{array} \tag{3.4.2}$$

where $w_1,\ldots,w_{k'}$ are the weight vectors associated with network $\mathcal{N}$. Since the coefficients of the linear program are rational, the optimal value λ_{max} and the corresponding solution x are rational. We can use any linear programming algorithm to solve (3.4.2) exactly, as in Algorithm `OracleRayExact_Route`, and obtain Algorithm `OracleRayExact_LCode`. We omit the pseudocode.

Using techniques by Garg and Könemann [13], we provide a combinatorial approximation algorithm, Algorithm `OracleRayApprox_LCode`, for solving the linear program (3.4.2) approximately. While the linear program (3.4.2) looks similar to the linear program (3.3.1), it is much harder to solve. The algorithm computes a point $\hat{r}$ such that $\lambda_{max}\hat{q}$ is between $\hat{r}$ and $(1+\omega)B\hat{r}$, for some numbers $\omega > 0$ and $B \geq 1$. We assume we have oracles $\mathcal{O}_{SLinear}$ and $\mathcal{O}_{FCover}$ for the following two subproblems related to (3.4.2):

Definition 47 (Minimum Cost Scalar-Linear Network Code Problem). *Given a network $\mathcal{N} = (\nu,\epsilon,\mu,c,\mathcal{A},S,R)$ with unit edge capacities, a finite field F and a length function $l:\epsilon\to\mathbb{R}_+$, compute the minimum cost scalar-linear network code solution for $\mathcal{N}$ with respect to F, if it exists.*

The cost of a solution is the sum of lengths of the edges used in the solution. If there is no scalar-linear solution, then report "unsolvable."

Without the minimum cost condition, the above problem reduces to the decidability problem of determining whether or not a network has a scalar-linear solution, which is NP-hard by Theorem 3.2 in Lehman and Lehman [21]. Hence, the above problem is at least as hard as any NP-hard problem. We assume that $\mathcal{O}_{SLinear}$ solves the minimum cost scalar-linear network code problem exactly.

Definition 48 (Fractional Covering with Box Constraints Problem). *Given an $n \times m$ nonnegative integer matrix A, a nonnegative vector b, a positive vector c and a nonnegative integer vector u, compute*

$$\begin{array}{lllll} \min & \sum_{j=1}^{m} c(j)x(j) & & & \\ \text{s.t.} & \sum_j A(i,j)x(j) & \geq & b(i), & \forall 1 \leq i \leq n \\ & x(j) & \leq & u(j), & \forall 1 \leq j \leq m \\ & x & \geq & 0. & \end{array}$$

In Algorithm `OracleRayApprox_LCode`, $\mathcal{O}_{FCover}$ solves the fractional covering problem of the following form with an approximation guarantee of B:

$$\begin{array}{lllll} \min & \sum_{i=1}^{k'} y(i)U_i(l) & & & \\ \text{s.t.} & \sum_{i=1}^{k'} [w_i]_j y(i) & \geq & q_j, & \forall 1 \leq j \leq |\mu| \\ & y(j) & \leq & \lceil q_j \rceil, & \forall 1 \leq j \leq k' \\ & y & \geq & 0, & \end{array} \tag{3.4.3}$$

where $w_1, \ldots, w_{k'}$ are the weight vectors associated with the network and $U_i(l)$ is the cost of the minimum cost partial scalar-linear solution in $\mathcal{W}_i$ with respect to the length function l.

Algorithm 6 Algorithm `OracleRayApprox_LCode`($\mathcal{N}$, ω, $\mathcal{O}_{SLinear}$, $\mathcal{O}_{FCover}$, B, $\hat{q}$)

1: Using Algorithm `DSteinerTreePacking`, compute the (approximate) value z_i to (3.3.2) for each message m_i separately.
2: $z = \min_i \frac{z_i}{q_i}$; $scaling\_factor = \frac{|\mu|}{z}$
3: $\hat{q} = scaling\_factor \cdot \hat{q}$
4: $\eta = \frac{1}{9B}\omega$; $\delta = (|\epsilon|/(1-\eta B))^{-1/\eta B}$
5: $N = 2\left\lceil \frac{1}{\eta B} \log_{1+\eta} \frac{|\epsilon|}{1-\eta B} \right\rceil$
6: **while true do**
7: $t = 0$; $l(e) = \frac{\delta}{c(e)}, \forall e \in \epsilon$
8: **for** phase $i = 1, \ldots, N$ **do**
9: $\gamma = 1$
10: **while** $\gamma > 0$ **do**
11: Using $\mathcal{O}_{SLinear}$, compute $U_i(l)$ for each weight vector w_i and corresponding partial scalar-linear solution $\widetilde{W}_i$ with the minimum cost.
12: Using $\mathcal{O}_{FCover}$, solve (3.4.3) to get the $y(1), \ldots, y(k')$ values.
13: $\widetilde{W} = y(1)\widetilde{W}_1 + \cdots + y(k')\widetilde{W}_{k'}$
14: $s = \max\left\{ \frac{\widetilde{W}(e)}{c(e)} : e \text{ such that } \widetilde{W}(e) > c(e) \right\}$
15: $\widetilde{W} = \frac{1}{s}\widetilde{W}$
16: $\gamma = \gamma - \frac{1}{s}\gamma$
17: Update $l(e) = l(e)\left(1 + \eta\frac{\widetilde{W}(e)}{c(e)}\right)$.
18: **if** $\sum_e l(e)c(e) > 1$ **then**
19: Goto Line 26.
20: **end if**
21: **end while**
22: $t = t + 1$
23: **end for**
24: $scaling\_factor = 2scaling\_factor$; $\hat{q} = 2\hat{q}$
25: **end while**
26: **return** $scaling\_factor \cdot t / \log_{1+\eta} \frac{1}{\delta}$

Analysis

The corresponding dual linear program of (3.4.2) is

$$\begin{array}{lll} \min & \sum_e l(e)c(e) & \\ \text{s.t.} & \sum_{e\in\epsilon} W(e)l(e) - \sum_{j=1}^{k}[w_i]_j z(j) \geq 0, & \forall 1 \leq i \leq k', \forall W \in \mathcal{W}_i \\ & \sum_{j=1}^{k} q_j z(j) \geq 1 & \\ & l, z \geq 0. & \end{array} \tag{3.4.4}$$

We rewrite the dual linear program (3.4.4) as two recursively nested linear programs. Consider the following linear program derived from the dual program:

$$\begin{array}{lll} \max & \sum_{j=1}^{k} q_j z(j) & \\ \text{s.t.} & \sum_{j=1}^{k}[w_i]_j z(j) \leq U_i(l), & \forall 1 \leq i \leq k' \\ & z \geq 0, & \end{array} \tag{3.4.5}$$

where $U_i(l) = \min_{W\in\mathcal{W}_i} \sum_e W(e)l(e)$. Let $D(l) = \sum_e l(e)c(e)$ and $\alpha(l)$ be the optimal value of the linear program (3.4.5). Then, solving (3.4.4) is equivalent to finding an assignment of lengths to the edges, $l : \epsilon \to \mathbb{R}^+$, so as to minimize $\frac{D(l)}{\alpha(l)}$. Let β denote the optimal value of (3.4.4), i.e., $\beta = \min_l \frac{D(l)}{\alpha(l)}$. The dual linear program for (3.4.5) is

$$\begin{array}{lll} \min & \sum_{i=1}^{k'} y(i)U_i(l) & \\ \text{s.t.} & \sum_{i=1}^{k'}[w_i]_j y(i) \geq q_j, & \forall 1 \leq j \leq |\mu| \\ & y \geq 0. & \end{array} \tag{3.4.6}$$

Let $\alpha'(y) = \min_{j:q_j\neq 0} \frac{\sum_{i=1}^{k'}[w_i]_j y(i)}{q_j}$ and $D'_l(y) = \sum_{i=1}^{k'} y(i)U_i(l)$. Without loss of generality, we assume that an optimal solution to (3.4.6) satisfies $y(j) \leq \lceil q_j \rceil$ for all j; then (3.4.6) is equivalent to (3.4.3). Solving (3.4.6) is equivalent to finding an assignment of values to variables y, $y : \{1, \ldots, k'\} \to \mathbb{R}^+$, so as to minimize $\frac{D'_l(y)}{\alpha'(y)}$. Let β' be the optimal value of (3.4.6), i.e., $\beta' = \min_y \frac{D'_l(y)}{\alpha'(y)}$. By linear programming duality, $\alpha(l) = \beta'$. Then,

$$\beta = \min_l \frac{D(l)}{\alpha(l)} = \min_l \frac{D(l)}{\min_y \frac{D'_l(y)}{\alpha'(y)}}.$$

First, we consider Algorithm `OracleRayApprox_LCode` with the infinite while-loop in line 6 removed, with variable *scaling_factor* and line 24 removed, and with the finite for-loop in line 8 replaced with an infinite while-loop.

Theorem 49. *Assume $\beta \geq 1$. For $0 < \omega < 1$, Algorithm* `OracleRayApprox_LCode`*, with modifications as explained above, returns a $(1+\omega)B$-approximate solution to the linear program (3.4.2) in at most $\left\lceil \frac{\beta}{\eta B} \log_{1+\eta} \frac{|\epsilon|}{1-\eta B} \right\rceil$ number of phases.*

Proof. Let $l_{i,j-1}$ denote the length function l and $\gamma_{i,j-1}$ the variable γ at the start of the j-th iteration (of the while-loop in line 10) in phase i. Let $\widetilde{W}_{i,j}$ denote the fractional solution $\widetilde{W}$ computed in the j-th iteration in phase i. Let $l_{i,0}$ be the length function at the start of phase i, or equivalently, at the end of phase $i-1$. For simplicity, we denote $D(l_{i+1,0})$ and $\alpha(l_{i+1,0})$ by $D(i)$ and $\alpha(i)$. Note that for each phase i and iteration j,

$$\begin{aligned}
D(l_{i,j}) &= \sum_e l(e)c(e) \\
&= D(l_{i,j-1}) + \eta \sum_e l_{i,j-1}(e)\widetilde{W}_{i,j}(e) \\
&= D(l_{i,j-1}) + \eta \sum_e l_{i,j-1}(e)(\gamma_{i,j-1} - \gamma_{i,j})[y(1)\widetilde{W}_1 + \ldots + y(k')\widetilde{W}_{k'}]_e \\
&\leq D(l_{i,j-1}) + \eta(\gamma_{i,j-1} - \gamma_{i,j})B\alpha(l_{i,j-1}) \\
&\leq D(l_{i,j-1}) + \eta(\gamma_{i,j-1} - \gamma_{i,j})B\alpha(l_{i+1,0}),
\end{aligned}$$

where $\widetilde{W}_1, \ldots, \widetilde{W}_{k'}$ and $y(1), \ldots, y(k')$ are the partial scalar-linear solutions and variable y from line 11. Note that we used the fact that α is a nondecreasing function to obtain the last inequality. Summing up the above inequality over the iterations, we obtain

$$\begin{aligned}
D(l_{i+1,0}) &\leq D(l_{i+1,-1}) + \eta(\gamma_{i+1,-1} - \gamma_{i+1,0})B\alpha(l_{i+1,0}) \\
&\leq D(l_{i+1,-2}) + \eta(\gamma_{i+1,-2} - \gamma_{i+1,0})B\alpha(l_{i+1,0}) \\
&\vdots \\
&\leq D(l_{i,0}) + \eta B\alpha(l_{i+1,0}),
\end{aligned}$$

where $l_{i+1,-j}$ (similarly, $\gamma_{i+1,-j}$) denote the length function l (variable γ) at the start of the j-th

from the last iteration in phase i. It follows that $D(i) \leq D(i-1) + \eta B\alpha(i)$. From here, the analysis is the same as Theorem 40, with B replacing A. □

While the running time of Algorithm `OracleRayApprox_LCode` depends on β by Theorem 49, we can reduce/increase it by scaling the $\hat{q}$ vector/capacities appropriately. We now remove the assumption that $\beta \geq 1$ and analyze the running time of Algorithm `OracleRayApprox_LCode` as a whole.

Theorem 50. *For $0 < \omega < 1$, Algorithm* `OracleRayApprox_LCode` *computes a $(1+\omega)B$-approximate solution to the linear program (3.4.2) in time $O(\omega^{-2}(\log A|\mu| + |\epsilon|)B \log |\epsilon| \cdot (T_{FCover} + k'T_{SLinear}))$, where T_{FCover} is the time required to solve the fractional covering problem by $\mathcal{O}_{FCover}$ within an approximation guarantee B and $T_{SLinear}$ is the time required to solve the minimum cost scalar-linear network code problem exactly by $\mathcal{O}_{SLinear}$.*

Proof. Let z_i^* be the exact maximum fractional Steiner tree packing value for message m_i in line 1 and let $z^* = \min_i \frac{z_i^*}{q_i}$. Note that z^* is an upper bound on the maximum rate at which the demands can be satisfied by a simple fractional linear code solution. Since $z_i \leq z_i^* \leq Az_i$, $\frac{z}{|\mu|} \leq \beta \leq Az$. We scale the $\hat{q}$ vector and get $1 \leq \beta \leq A|\mu|$. The assumption $\beta \geq 1$ is satisfied, but now β could be as large as $A|\mu|$. We employ the doubling trick with $N = \lceil \frac{1}{\eta B} \log_{1+\eta} \frac{|\epsilon|}{1+\eta B} \rceil$. Then, the total number of phases is at most $N \log A|\mu|$. Each phase consists of variably many number of iterations of the while-loop in line 10 and, in all iterations except the last, we increase the length of an edge by a factor of $1+\eta$. Hence, the total number of iterations exceeds the total number of phases by at most $|\epsilon| \log_{1+\eta} \frac{1}{\delta}$. The total number of calls to the oracle $\mathcal{O}_{FCover}$ is at most $N \log A|\mu| + |\epsilon| \log_{1+\eta} \frac{1}{\delta}$. For each iteration, we call the oracle $\mathcal{O}_{SLinear}$ exactly k' times to compute the $U_i(l)$ values and, hence, the total number of calls to the oracle $\mathcal{O}_{SLinear}$ is at most $(N \log A|\mu| + |\epsilon| \log_{1+\eta} \frac{1}{\delta})k'$. From here, the proof follows Theorem 42 closely. □

3.4.5 Implementations of Oracles $\mathcal{O}_{SLinear}$ and $\mathcal{O}_{FCover}$

In this subsection, we discuss implementations of oracles $\mathcal{O}_{SLinear}$ and $\mathcal{O}_{FCover}$.

Oracle $\mathcal{O}_{SLinear}$

We only consider implementations of exact oracle $\mathcal{O}_{SLinear}$ for the minimum cost scalar-linear network code problem (Definition 47). The assumption that the finite field F is fixed for the computation of semi-network linear coding capacity region is important; it ensures termination of algorithms for $\mathcal{O}_{SLinear}$, given that the decidability of the linear coding problem without a fixed finite field is unknown at this time.

A brute force approach is to loop through all possible subset of active edges and try all possible combinations of global linear coding vectors (see Section 2.1) on these edges in time $O(2^{|\epsilon|}|F|^{|\mu||\epsilon|})$. For each edge $e = (x, y)$, it takes $O(|\mu|I)$ time to check if the global coding vector on e is the span of global coding vectors on in-edges of x, where I is the maximum in-degree of any node. The total running time is $O(2^{|\epsilon|}|F|^{|\mu||\epsilon|}|\mu||\epsilon|I)$ and is exponential in not only in $|\epsilon|$, but also in the number of messages, $|\mu|$. The brute force approach requires $O(|F|^{|\mu||\epsilon|})$ space.

We propose a faster algorithm of our own that solves the minimum cost scalar-linear network code problem using dynamic programming. First, we relabel nodes so that edges go from a lower-numbered node to a higher-numbered one and arrange the nodes in a line in order. This can be done as the network is acyclic and by a topological sort algorithm. Let $n_1, \ldots, n_{|\nu|}$ be the nodes in order. Second, we create states indexed by a triple (i, ϵ_i, ϕ_i), where

1. i is an integer, $1 \leq i \leq |\nu| - 1$,
2. ϵ_i is the set of edges that have the start node in $\{n_1, \ldots, n_i\}$ and the end node in $\{n_{i+1}, \ldots, n_{|\nu|}\}$, and
3. ϕ_i is the global coding vectors on edges in ϵ_i.

In each state (i, ϵ_i, ϕ_i), we store the minimum cost scalar-linear solution, if it exists, with the set of global coding vectors ϕ_i on ϵ_i when the network is restricted to nodes $n_1, \ldots, n_i$. In essence, we consider the restricted network of nodes $n_1, \ldots, n_i$ and compute its minimum cost scalar-linear solution based on the minimum cost scalar-linear solutions found on the restricted network of nodes $n_1, \ldots, n_{i-1}$. See Algorithm `OracleSLinear_DP` for more details. Note that *valid_next_state* and *valid_prev_state* are linked lists.

Theorem 51. *Algorithm* `OracleSLinear_DP` *solves the minimum cost scalar-linear network code problem in time* $O(|F|^{2|\mu|\Phi}|\nu||\mu|\Phi^2)$ *and space* $O(|F|^{|\mu|\Phi})$, *where* $\Phi = \max_i |\epsilon_i|$.

Proof. Let $\Phi = \max_i |\epsilon_i|$. Then, the number of states with a specific pair of i and ϵ_i is at most $|F|^{|\mu|\Phi}$ and the time to compute the states of the form (i, ϵ_i, ϕ_i) from the states of the form $(i-1, \epsilon_{i-1}, \phi_{i-1})$ takes $O(|F|^{2|\mu|\Phi}|\mu|\Phi^2)$, by simply looping pairs of the states (i, ϵ_i, ϕ_i) and $(i-1, \epsilon_{i-1}, \phi_{i-1})$ and checking if the global coding vectors in ϕ_i follow from those in ϕ_{i-1}. As i ranges from 1 to $|\nu| - 1$, the total running time of the algorithm is $O(|F|^{2|\mu|\Phi}|\nu||\mu|\Phi^2)$. The total space required is $O(|F|^{|\mu|\Phi})$ if we use the "sliding window" trick where we only keep states in two consecutive levels (corresponding to two consecutive values of i) at any time. □

Algorithm 7 Algorithm `OracleSLinear_DP`($\mathcal{N}$, l)

1: Sort nodes with a topological sort algorithm: $n_1, \ldots, n_{|\nu|}$.
2: *valid_next_state* $\leftarrow \emptyset$; *valid_prev_state* $\leftarrow (0, \emptyset, \emptyset)$, the trivial empty solution.
3: **for** $i = 1, \ldots, |\nu| - 1$ **do**
4: Update *valid_prev_state* so that only states that satisfy demands on n_i exist in the list.
5: *valid_next_state* $\leftarrow$ NULL
6: **for** each possible coding vector combination ϕ_{i+1} on ϵ_{i+1} **do**
7: Find, if possible, a state in *valid_prev_state* that leads to a minimum cost solution for $(i+1, \epsilon_{i+1}, \phi_{i+1})$.
8: If successful, add the state $(i+1, \epsilon_{i+1}, \phi_{i+1})$ to *valid_next_state*.
9: **end for**
10: *valid_prev_start* $\leftarrow$ *valid_next_state*
11: **end for**
12: From *valid_prev_state*, pick the minimum cost solution that satisfies all the demands at $n_{|\nu|}$.
13: **return** the minimum cost solution found or "unsolvable" if no such solution exists.

Clearly, the asymptotic behavior of Algorithm `OracleSLinear_DP` is much better than that of the brute force approach when $\Phi \ll |\epsilon|$ in the network. We summarize algorithms for oracle $\mathcal{O}_{SLinear}$ with Table 3.2:

Table 3.2: Implementations of oracle $\mathcal{O}_{SLinear}$

Algorithm for $\mathcal{O}_{SLinear}$	Approximation Ratio	Time	Space
BruteForce	1	$O(2^{\|\epsilon\|}\|F\|^{\|\mu\|\|\epsilon\|}\|\mu\|\|\epsilon\|I)$	$O(\|F\|^{\|\mu\|\|\epsilon\|})$
Algorithm `OracleSLinear_DP`	1	$O(\|F\|^{2\|\mu\|\Phi}\|\nu\|\|\mu\|\Phi^2)$	$O(\|F\|^{\|\mu\|\Phi})$

Oracle $\mathcal{O}_{FCover}$

The oracle $\mathcal{O}_{FCover}$ solves the fractional covering problem (Definition 48). We can solve the problem with a polynomial-time linear program solver such as the ellipsoid algorithm. In our problem of

networks, the number of variables of the linear program (3.4.3) is at most $2^{|\mu|}$ and the number of constraints is $|\mu|$. Hence, a polynomial-time linear program solver will give an exact solution in time polynomial in $2^{|\mu|}$. As the ellipsoid algorithm could be slow in practice and could depend on $2^{|\mu|}$ poorly, the actual running time might not be practical. Fleischer [11] proposed a combinatorial approximation algorithm for the covering problem that solves within an approximation ratio of $(1+\omega)$ by using $O(\omega^{-2}2^{|\mu|}\log(c^{\mathrm{T}}u))$ calls to an oracle that returns a most violated constraint (where $\omega > 0$). Note that Fleischer's algorithm is proposed for nonnegative integer matrix A and nonnegative integer vectors b, c and u, but the algorithm still works for our formulation of the fractional covering problem. As there are $|\mu|$ constraints and at most $2^{|\mu|}$ variables, the oracle for the most violated constraint can be implemented in time $O(2^{|\mu|}|\mu|)$, and this leads to an algorithm that computes a $(1+\omega)$-approximate solution in time $O(\omega^{-2}2^{2|\mu|}|\mu|\log(c^{\mathrm{T}}u))$. We summarize with Table 3.3:

Table 3.3: Implementations of oracle $\mathcal{O}_{FCover}$

Algorithm for $\mathcal{O}_{cover}$	Approximation Ratio B	Time
Ellipsoid Algorithm	1	$polynomial(2^{\|\mu\|}, \|\mu\|)$
Fleischer [11]	$1+\omega$	$O(\omega^{-2}2^{2\|\mu\|}\|\mu\|\log(c^{\mathrm{T}}u))$

3.5 Examples

In this section, we provide examples of network capacity regions in the case of two messages. Instead of Algorithm `FacetEnum_Route` (or `FacetEnum_LCode`), we use Algorithm `BoundaryTrace2D` as the polytope reconstruction algorithm. Algorithm `BoundaryTrace2D` is similar to the algorithm given by Cole and Yap [5] in that the common main idea is that three collinear boundary points define a face. Algorithm `BoundaryTrace2D` is different from Cole and Yap [5] in that it considers rays that start from the origin. See the pseudocode for the details of Algorithm `BoundaryTrace2D`. Note $\mathcal{L}$ is a linked list. We refer to Appendix A.2.1 for a proof of its correctness. The algorithm works both for $\mathcal{C}_r$ and $\mathcal{C}_l'$ with an appropriate oracle $\mathcal{O}_{Ray}$.

The example networks are given in Figure 3-1. Nodes 1 and 2 are the source nodes and nodes 5 and 6 are the receiver nodes. To compute the exact description of $\mathcal{C}_r$ or $\mathcal{C}_l'$, we hard-coded the corresponding linear programs and used a linear program solver, linprog, in MATLAB. As the networks are simple, it was easy to enumerate all Steiner trees and partial scalar-linear solutions,

Algorithm 8 Algorithm `BoundaryTrace2D`($\mathcal{N}$, $\mathcal{O}_{Ray}$)

$\mathcal{L} \leftarrow \emptyset$
Using $\mathcal{O}_{Ray}$, compute the intersection points on $x = e_i \cdot t, t \geq 0$ for $i = 1, 2$ and obtain (x_1, y_1) and (x_2, y_2).
Insert $(x_1, -1)$, (x_1, y_1),(x_2, y_2), and $(-1, y_2)$ onto the head of $\mathcal{L}$, in that order.
cur_pointer $\leftarrow$ the head of $\mathcal{L}$
while there exist three distinct points after *cur_pointer* **do**
  Let $pt1$, $pt2$, $pt3$, and $pt4$ be the four consecutive points starting at *cur_pointer*.
  Let line l_1 go through $pt1$ and $pt2$, and line l_2 go through $pt3$ and $pt4$.
  if intersection point p exists between l_1 and l_2 **then**
    Using $\mathcal{O}_{Ray}$, compute the boundary point r on $x = pt, t \geq 0$.
    if $r \neq pt2$ and $r \neq pt3$ **then**
      Insert point r into $\mathcal{L}$ between $pt2$ and $pt3$.
    else
      Advance *cur_pointer*.
    end if
  else
    Advance *cur_pointer*.
  end if
end while
return $\mathcal{L} \cup (0, 0)$, except $(x_1, -1)$ and $(-1, y_2)$.

and the corresponding linear programs were small. While this approach worked for the particular examples we present, it might not be suitable for bigger or more complicated networks. To get around the numerical issues, we used the tolerance of .05; for instance, two points whose corresponding coordinates differ by at most .05 are considered the same point. For the networks, we assume $\mathcal{A} = \{0, 1\}$. For the semi-network linear coding capacity regions, we assume the finite field F is $\mathbb{F}_2$. To obtain approximate intersections points, we simply used an approximate oracle $\mathcal{O}_{Ray}$ in place of the linear program solver. We note that the approximate oracle $\mathcal{O}_{Ray}$ worked well with Algorithm `BoundaryTrace2D` and led to its successful termination for these networks, but this may not hold for arbitrary networks in general.

For the network routing capacity region of network $\mathcal{N}_1$ in Figure 3-2, we used $\omega = .5$ in Algorithms `OracleRayApprox_Route` and `DSteinerTreePacking` for $\mathcal{O}_{Ray}$ and the algorithm due to Charikar et al. [4] for $\mathcal{O}_{DSteiner}$ with the approximation ratio $A = O(\log^2 |\nu|)$. For the semi-network linear coding capacity region of network $\mathcal{N}_1$ in Figure 3-3, we used $\omega = .5$ in Algorithms `OracleRayApprox_Route` and `DSteinerTreePacking` for $\mathcal{O}_{Ray}$, the algorithm due to Fleischer [11] with the approximation ratio $B = 1.1$ for $\mathcal{O}_{FCover}$, and Algorithm `OracleSLinear_DP` for $\mathcal{O}_{SLinear}$.

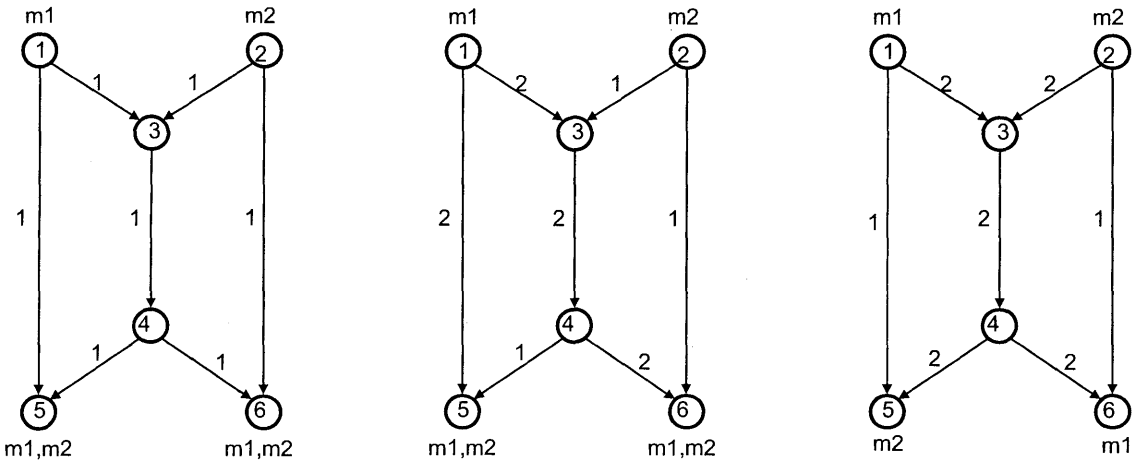

Figure 3-1: The Butterfly network with unit edge capacities and its variants: $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$.

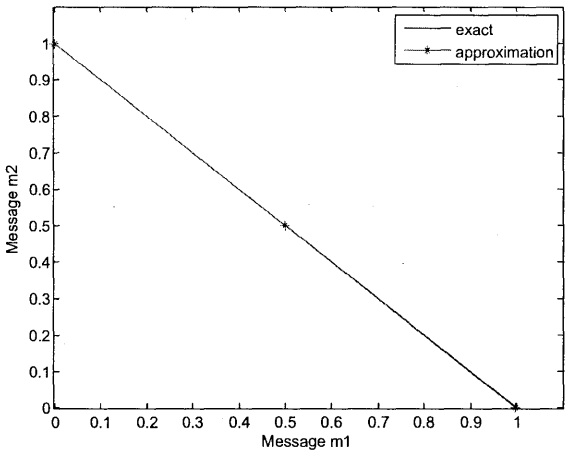

Figure 3-2: Network routing capacity region $\mathcal{C}_r$ of $\mathcal{N}_1$

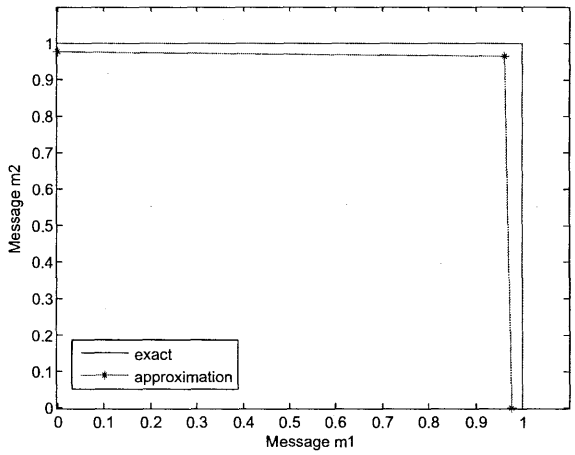

Figure 3-3: Semi-network linear coding capacity region $\mathcal{C}'_l$ of $\mathcal{N}_1$, with respect to $\mathbb{F}_2$

For the network routing capacity region of network $\mathcal{N}_2$ in Figure 3-4, we used $\omega = .5$ in Algorithms `OracleRayApprox_Route` and `DSteinerTreePacking` for $\mathcal{O}_{Ray}$ and the algorithm due to Charikar et al. [4] for $\mathcal{O}_{DSteiner}$ with the approximation ratio $A = O(\log^2 |\nu|)$. For the semi-network linear coding capacity region of network $\mathcal{N}_2$ in Figure 3-5, we used $\omega = .5$ in Algorithms `OracleRayApprox_Route` and `DSteinerTreePacking` for $\mathcal{O}_{Ray}$, the algorithm due to Fleischer [11] with the approximation ratio $B = 1.1$ for $\mathcal{O}_{FCover}$, and Algorithm `OracleSLinear_DP` for $\mathcal{O}_{SLinear}$.

For the network routing capacity region of network $\mathcal{N}_3$ in Figure 3-6, we used $\omega = .5$ in Algorithms `OracleRayApprox_Route` and `DSteinerTreePacking` for $\mathcal{O}_{Ray}$ and the brute force algorithm for $\mathcal{O}_{DSteiner}$ with the approximation ratio $A = 1$. For the semi-network linear coding capacity

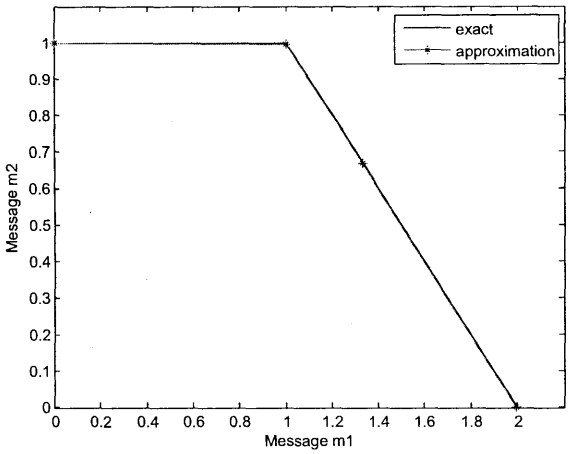

Figure 3-4: Network routing capacity region $\mathcal{C}_r$ of $\mathcal{N}_2$

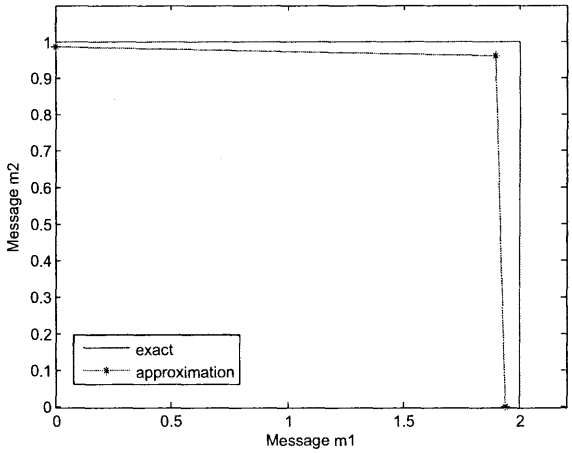

Figure 3-5: Semi-network linear coding capacity region $\mathcal{C}'_l$ of $\mathcal{N}_2$, with respect to $\mathbb{F}_2$

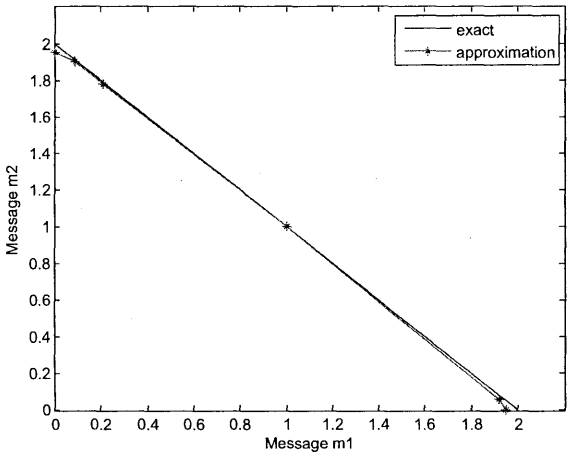

Figure 3-6: Network routing capacity region $\mathcal{C}_r$ of $\mathcal{N}_3$

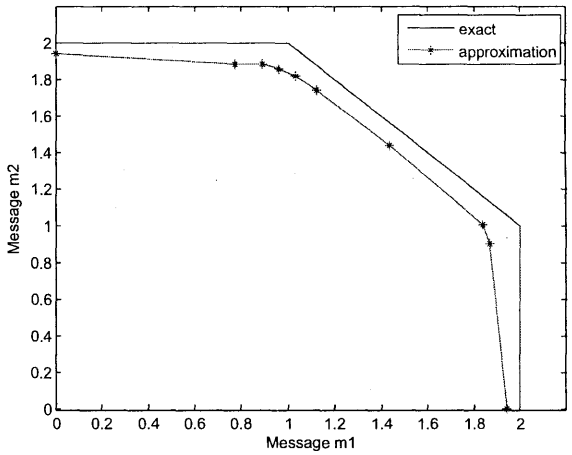

Figure 3-7: Semi-network linear coding capacity region $\mathcal{C}'_l$ of $\mathcal{N}_3$, with respect to $\mathbb{F}_2$

region of network $\mathcal{N}_3$ in Figure 3-7, we used $\omega = .9$ in Algorithm `OracleRayApprox_Route`, $\omega = .5$ in Algorithm `DSteinerTreePacking`, the algorithm due to Fleischer [11] with the approximation ratio $B = 1.1$ for $\mathcal{O}_{FCover}$, and Algorithm `OracleSLinear_DP` for $\mathcal{O}_{SLinear}$.

3.6 Discussion and Conclusion

In this chapter, we defined the network capacity region of networks analogously to the rate regions in information theory. In the case of the network routing capacity region, we showed that the region is a rational polytope and provided exact algorithms and approximation heuristics for computing the

polytope. In the case of the network linear coding capacity region, we defined an auxiliary polytope, the semi-network linear coding capacity region, that is a rational polytope and that inner bounds the network linear coding capacity region. We provided exact algorithms and approximation heuristics for computing the auxiliary polytope. We noted that the algorithms and heuristics presented in this chapter are not polynomial time schemes.

Our results have a few straightforward extensions. We can design membership algorithms that given a rate vector, determines whether or not there exists a fractional network code solution that achieves it from algorithms we provided, for the network routing capacity region and semi-network linear coding capacity region. Also, we can compute corresponding approximate solutions $(x(T))_{T\in\mathcal{T}}$ for linear program (3.3.1) by storing counters for Steiner trees $\widetilde{T}$ in Algorithm `OracleRayApprox_Route`; the same is true for $(x(W))_{W\in\mathcal{W}}$.

While our results apply to the networks defined on directed acyclic multigraphs, they generalize to directed networks with cycles and undirected networks straightforwardly. In the computation of the network routing capacity region, we consider minimal fractional routing solutions that can be decomposed into a set of Steiner trees defined appropriately for directed networks with cycles (or undirected networks) and modify the algorithms correspondingly. In the computation of the semi-network linear coding capacity region, we consider simple fractional linear coding solutions that can be decomposed into a set of partial scalar-linear solutions defined appropriately for the networks and modify the algorithms correspondingly.

In connection to Cannons et al. [2], our work essentially addresses a few problems proposed by Cannons et al.: whether there exists an efficient algorithm for computing the network routing capacity and whether there exists an algorithm for computing the network linear coding capacity. It follows from our work that there exist combinatorial approximation algorithms for computing the network routing capacity and for computing a lower bound of the network linear coding capacity.

We conclude with a few open problems related to our work: determine how good of an inner bound the semi-network linear coding capacity region is to the network linear coding capacity region; design an efficient algorithm, if possible, for computing the linear coding capacity region and network capacity region; design an efficient algorithm, if possible, for the minimum cost scalar-linear network code problem.

Appendix A

Computations and Proofs

A.1 Computations

A.1.1 Computation in Theorem 39

From Section 2 of Garg and Könemann [13] and the fact that oracle $\mathcal{O}_{DSteiner}$ is an approximate oracle with the approximation guarantee of A, it follows that the ratio of dual optimal and primal feasible solutions, ζ, satisfies

$$\zeta < (1-\eta)^{-2}A.$$

We choose η appropriately to make sure that the $\zeta < (1+\omega)A$. It suffices to choose η such that $(1-\eta)^{-2} \leq 1+\omega$, or equivalently, $(1-\eta)^{-1} \leq (1+\omega)^{1/2}$. By the Taylor Series Theorem, for $0 < \omega < 1$, we have

$$1 + \frac{1}{2}\omega - \frac{1}{8}\omega^2 \leq (1+\omega)^{1/2}.$$

Note that for $0 < \eta \leq \frac{1}{2}$,

$$(1-\eta)^{-1} = 1 + \eta + \eta^2 + \ldots = 1 + \eta\frac{1}{1-\eta} \leq 1 + 2\eta.$$

Then, for $\eta = \frac{3}{16}\omega$, we have that $0 < \omega < 1$ implies $0 < \eta \leq \frac{1}{2}$ and that

$$(1-\eta)^{-1} \leq 1 + 2\eta \leq 1 + \frac{3}{8}\omega \leq 1 + \frac{1}{2}\omega - \frac{1}{8}\omega^2 \leq (1+\omega)^{1/2}.$$

A.1.2 Computation in Theorem 42

We want to choose η appropriately so that $(1-\eta A)^{-3} \leq (1+\omega)$, or equivalently, that $(1-\eta A)^{-1} \leq (1+\omega)^{1/3}$. By the Taylor Series Theorem, for $0 < \omega < 1$, we have

$$1 + \frac{1}{3}\omega - \frac{1}{9}\omega^2 \leq (1+\omega)^{1/3}.$$

Note that for $0 < \eta A \leq \frac{1}{2}$,

$$(1-\eta A)^{-1} = 1 + \eta A + (\eta A)^2 + \ldots = 1 + \eta A \frac{1}{1-\eta A} \leq 1 + 2\eta A.$$

Then, for $\eta = \frac{1}{9A}\omega$, we have that $0 < \omega < 1$ implies $0 < \eta A \leq \frac{1}{2}$ and that

$$(1-\eta A)^{-1} \leq 1 + 2\eta A \leq 1 + \frac{2}{9}\omega \leq 1 + \frac{1}{3}\omega - \frac{1}{9}\omega^2 \leq (1+\omega)^{1/3}.$$

A.2 Proofs

A.2.1 Proof for Algorithm `BoundaryTrace2D`

Without loss of generality, we prove that Algorithm `BoundaryTrace2D` is correct for the computation of the network routing capacity region $\mathcal{C}_r$. Note that $\mathcal{L}$ is a linked list of computed boundary points (and the two auxiliary points $(x_1, -1)$ and $(-1, y_2)$) ordered clockwise. When two lines intersect, we mean that the lines intersect in exactly one point. When a line goes through a line segment, we mean that the line intersects the line segment in exactly one point.

Theorem 52. *Algorithm* `BoundaryTrace2D` *calls the oracle* $\mathcal{O}_{Ray}$ $O(n)$ *times where* n *is the number of edges in the polygon* $\mathcal{C}_r$.

Proof. We first show that, for each edge e of $\mathcal{C}_r$, the number of distinct boundary points computed in the interior of the edge (excluding the vertices) is at most 3 throughout the execution of Algorithm `BoundaryTrace2D`. Let edge e have 3 distinct boundary points computed in its interior: p_1, p_2, and p_3, ordered clockwise. Then, any 4 consecutive boundary points in $\mathcal{L}$ cannot produce a ray in line 9 that goes through the interior of edge e. If the set of 4 consecutive points contains at most 2 of p_1, p_2, and p_3, then, clearly, the ray does not go through the interior of e. If the set of 4 consecutive

points contains all 3 points p_1, p_2, and p_3 and a point before p_1 in $\mathcal{L}$, then the intersection point p in line 8 is exactly $pt2$ and no new boundary point is created. Similarly, it can be shown in other remaining cases that no new boundary point is introduced.

Note that, for each boundary point b in $\mathcal{L}$, there can be at most 3 calls to oracle $\mathcal{O}_{Ray}$ associated with it; a call to compute the boundary point for the first time, a call if b appears as $pt2$ in line 6 and as the boundary point r in line 9, and a call if b appears as $pt3$ in line 6 and as the boundary point r in line 9. As there are $n-2$ edges and $n-1$ vertices on the outer boundary of $\mathcal{C}_r$ and $O(1)$ calls to $\mathcal{O}_{Ray}$ for each boundary point computed, the statement follows. □

The correctness of `BoundaryTrace2D` follows from the fact that each vertex on the outer boundary of $\mathcal{C}_r$ is computed by oracle $\mathcal{O}_{Ray}$ and that after enough boundary points have been computed, the *cur_pointer* in the algorithm will advance to termination. It is easy to see that all vertices of the polygon $\mathcal{C}_r$ are included in the returned list $\mathcal{L}$. Assume a vertex v is missed in $\mathcal{L}$ and the algorithm terminated successfully. Assume that $pt1$, $pt2$, $pt3$ and $pt4$ are the four consecutive points in the resulting list such that the line segment between the origin and v and the line segment between $pt2$ and $pt3$ intersect. Note that $pt2$ and $pt3$ are on different edges of $\mathcal{C}_r$. Then, it is easy to see that we have the ray in line 9 going through the interior of the line segment between $pt2$ and $pt3$ and, hence, a new boundary point would have been added. Therefore, *cur_pointer* should not have advanced to termination, and this contradicts that the vertex v is missing from $\mathcal{L}$.

Bibliography

[1] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, Jul 2000.

[2] J. Cannons, R. Dougherty, C. Freiling, and K. Zeger. Network routing capacity. In *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, pages 11 –13, 4-9 2005.

[3] T. Chan and A. Grant. Mission impossible: Computing the network coding capacity region. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 320 –324, 6-11 2008.

[4] Moses Charikar, Chandra Chekuri, To-yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 192–200, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[5] Richard Cole and Chee K. Yap. Shape from probing. *J. Algorithms*, 8(1):19–38, 1987.

[6] R. Dougherty, C. Freiling, and K. Zeger. Insufficiency of linear coding in network information flow. *Information Theory, IEEE Transactions on*, 51(8):2745–2759, Aug. 2005.

[7] R. Dougherty, C. Freiling, and K. Zeger. Matroidal networks. In *Allerton Conference on Communication, Control, and Computing*, September 2007.

[8] R. Dougherty, C. Freiling, and K. Zeger. Networks, matroids, and non-shannon information inequalities. *Information Theory, IEEE Transactions on*, 53(6):1949–1969, June 2007.

[9] R. Dougherty, C. Freiling, and K. Zeger. Linear network codes and systems of polynomial equations. *Information Theory, IEEE Transactions on*, 54(5):2303–2316, May 2008.

[10] S. El Rouayheb, A. Sprintson, and C. Georghiades. A new construction method for networks from matroids. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 2872–2876, 28 2009-July 3 2009.

[11] Lisa Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1001–1010, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[12] Komei Fukuda. Frequently asked questions in polyhedral computation. `http://www.ifor.math.ethz.ch/~fukuda/polyfaq/polyfaq.html`, June 2004. Date retrieved: July 14, 2010.

[13] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

[14] Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of discrete and computational geometry*. CRC Press, Inc., 2 edition, 2004.

[15] Peter Gritzmann, Victor Klee, and John Westwater. Polytope Containment and Determination by Linear Probes. *Proc. London Math. Soc.*, s3-70(3):691–720, 1995.

[16] Nicholas J. A. Harvey, Robert Kleinberg, and April Rasala Lehman. On the capacity of information networks. *IEEE/ACM Trans. Netw.*, 14(SI):2345–2364, 2006.

[17] Kamal Jain, Mohammad Mahdian, and Mohammad R. Salavatipour. Packing steiner trees. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 266–274, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[18] Volker Kaibel and Marc E. Pfetsch. Some algorithmic problems in polytope theory. In *IN ALGEBRA, GEOMETRY, AND SOFTWARE SYSTEMS*, pages 23–47. Springer-Verlag, 2003.

[19] Ali Kakhbod and S. M. Sadegh Tabatabaei Yazdi. On describing the routing capacity regions of networks. *Mathematical Methods of Operations Research*, pages 1432–2994, May 2010.

[20] R. Koetter and M. Médard. An algebraic approach to network coding. *Networking, IEEE/ACM Transactions on*, 11(5):782–795, Oct. 2003.

[21] April Rasala Lehman and Eric Lehman. Complexity classification of network information flow problems. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 142–150, Jan. 2004.

[22] S.-Y.R. Li, R.W. Yeung, and Ning Cai. Linear network coding. *Information Theory, IEEE Transactions on*, 49(2):371–381, Feb. 2003.

[23] M. Médard, M. Effros, T. Ho, and D. Karger. On coding for non-multicast networks. In *Proc. 41st Annual Allerton Conference on Communication, Control and Computing*, Oct. 2003.

[24] J.G. Oxley. *Matroid Theory*. New York: Oxford Univ. Press, 1992.

[25] Qifu Sun, Siu Ting Ho, and S.-Y.R. Li. On network matroids and linear network codes. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 1833 –1837, 6-11 2008.

[26] S. Thakor, A. Grant, and T. Chan. Network coding capacity: A functional dependence bound. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 263 –267, june 2009.

[27] Xijin Yan, Jun Yang, and Zhen Zhang. An outer bound for multisource multisink network coding with minimum cost consideration. *IEEE/ACM Trans. Netw.*, 14(SI):2373–2385, 2006.

[28] Xijin Yan, Raymond W. Yeung, and Zhen Zhang. The capacity region for multi-source multi-sink network coding. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pages 116 –120, 24-29 2007.

[29] S. M. Sadegh Tabatabaei Yazdi, Serap A. Savari, Farzad Farnoud, and Gerhard Kramer. A multimessage capacity region for undirected ring networks. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pages 1091 –1095, 24-29 2007.

[30] S.M.S. Yazdi, S.A. Savari, K. Carlson, and G. Kramer. The capacity region of a collection of multicast sessions in an undirected ring network. In *Parallel Processing Workshops, 2007. ICPPW 2007. International Conference on*, pages 40 –40, 10-14 2007.