

## Network Coding Fundamentals

Christina Fragouli<sup>1</sup> and Emina Soljanin<sup>2</sup>

<sup>1</sup> *École Polytechnique Fédérale de Lausanne (EPFL), Switzerland,  
christina.fragouli@epfl.ch*

<sup>2</sup> *Bell Laboratories, Alcatel-Lucent, USA, emina@research.bell-labs.com*

### Abstract

Network coding is an elegant and novel technique introduced at the turn of the millennium to improve network throughput and performance. It is expected to be a critical technology for networks of the future. This tutorial addresses the first most natural questions one would ask about this new technique: how network coding works and what are its benefits, how network codes are designed and how much it costs to deploy networks implementing such codes, and finally, whether there are methods to deal with cycles and delay that are present in all real networks. A companion issue deals primarily with applications of network coding.

# 1

---

## Introduction

---

Networked systems arise in various communication contexts such as phone networks, the public Internet, peer-to-peer networks, ad-hoc wireless networks, and sensor networks. Such systems are becoming central to our way of life. During the past half a century, there has been a significant body of research effort devoted to the operation and management of networks. A pivotal, inherent premise behind the operation of all communication networks today lies in the way information is treated. Whether it is packets in the Internet, or signals in a phone network, if they originate from different sources, they are transported much in the same manner as cars on a transportation network of highways, or fluids through a network of pipes. Namely, independent information streams are kept separate. Today, routing, data storage, error control, and generally all network functions operate on this principle.

Only recently, with the advent of network coding, the simple but important observation was made that in communication networks, we can allow nodes to not only forward but also process the incoming independent information flows. At the network layer, for example, intermediate nodes can perform binary addition of independent bitstreams,

whereas, at the physical layer of optical networks, intermediate nodes can superimpose incoming optical signals. In other words, data streams that are independently produced and consumed do not necessarily need to be kept separate when they are transported throughout the network: there are ways to combine and later extract independent information. Combining independent data streams allows to better tailor the information flow to the network environment and accommodate the demands of specific traffic patterns. This shift in paradigm is expected to revolutionize the way we manage, operate, and understand organization in networks, as well as to have a deep impact on a wide range of areas such as reliable delivery, resource sharing, efficient flow control, network monitoring, and security.

This new paradigm emerged at the turn of the millennium, and immediately attracted a very significant interest in both Electrical Engineering and Computer Science research communities. This is an idea whose time has come; the computational processing is becoming cheaper according to Moore's law, and therefore the bottleneck has shifted to network bandwidth for support of ever-growing demand in applications. Network coding utilizes cheap computational power to dramatically increase network throughput. The interest in this area continues to increase as we become aware of new applications of these ideas in both the theory and practice of networks, and discover new connections with many diverse areas (see Figure 1.1).

Throughout this tutorial we will discuss both theoretical results as well as practical aspects of network coding. We do not claim to exhaustively represent and reference all current work in network coding; the presented subjects are the problems and areas that are closer to our interests and offer our perspective on the subject. However, we did attempt the following goals:

- (1) to offer an introduction to basic concepts and results in network coding, and
- (2) to review the state of the art in a number of topics and point out open research directions.

We start from the main theorem in network coding, and proceed to discuss network code design techniques, benefits, complexity require-

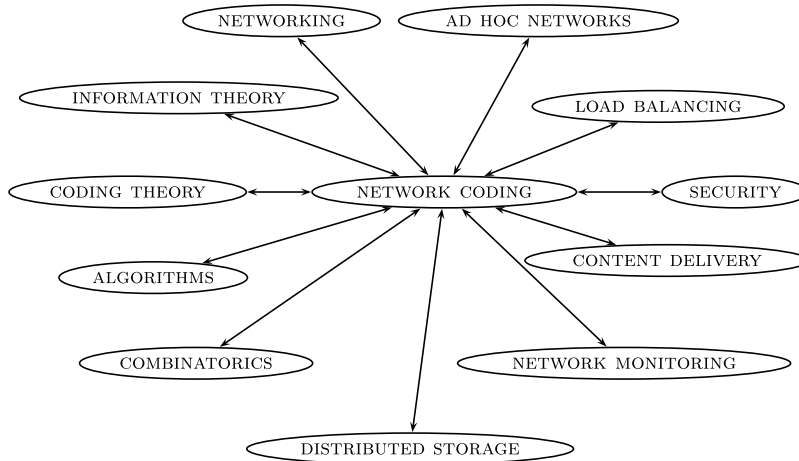


Fig. 1.1 Connections with other disciplines.

ments, and methods to deal with cycles and delay. A companion volume is concerned with application areas of network coding, which include wireless and peer-to-peer networks.

In order to provide a meaningful selection of literature for the novice reader, we reference a limited number of papers representing the topics we cover. We refer a more interested reader to the webpage [www.networkcoding.info](http://www.networkcoding.info) for a detailed literature listing. An excellent tutorial focused on the information theoretic aspects of network coding is provided in [49].

## 1.1 Introductory Examples

The following simple examples illustrate the basic concepts in network coding and give a preliminary idea of expected benefits and challenges.

### 1.1.1 Benefits

Network coding promises to offer benefits along very diverse dimensions of communication networks, such as throughput, wireless resources, security, complexity, and resilience to link failures.

## Throughput

The first demonstrated benefits of network coding were in terms of throughput when multicasting. We discuss throughput benefits in Chapter 4.

---

**Example 1.1.** Figure 1.2 depicts a communication network represented as a directed graph where vertices correspond to terminals and edges correspond to channels. This example is commonly known in the network coding literature as the butterfly network. Assume that we have slotted time, and that through each channel we can send one bit per time slot. We have two sources  $S_1$  and  $S_2$ , and two receivers  $R_1$  and  $R_2$ . Each source produces one bit per time slot which we denote by  $x_1$  and  $x_2$ , respectively (unit rate sources).

If receiver  $R_1$  uses all the network resources by itself, it could receive both sources. Indeed, we could route the bit  $x_1$  from source  $S_1$  along the path  $\{AD\}$  and the bit  $x_2$  from source  $S_2$  along the path  $\{BC, CE, ED\}$ , as depicted in Figure 1.2(a). Similarly, if the second receiver  $R_2$  uses all the network resources by itself, it could also receive both sources. We can route the bit  $x_1$  from source  $S_1$  along the path  $\{AC, CE, EF\}$ , and the bit  $x_2$  from source  $S_2$  along the path  $\{BF\}$  as depicted in Figure 1.2(b).

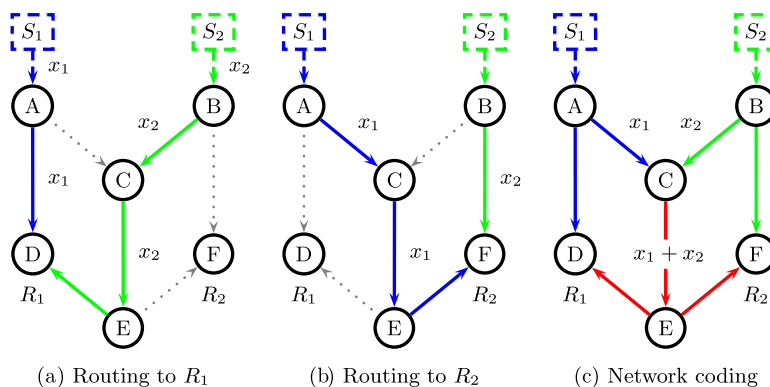


Fig. 1.2 The Butterfly Network. Sources  $S_1$  and  $S_2$  multicast their information to receivers  $R_1$  and  $R_2$ .

Now assume that both receivers want to simultaneously receive the information from both sources. That is, we are interested in multicasting. We then have a “contention” for the use of edge  $CE$ , arising from the fact that through this edge we can only send one bit per time slot. However, we would like to simultaneously send bit  $x_1$  to reach receiver  $R_2$  and bit  $x_2$  to reach receiver  $R_1$ .

Traditionally, information flow was treated like fluid through pipes, and independent information flows were kept separate. Applying this approach we would have to make a decision at edge  $CE$ : either use it to send bit  $x_1$ , or use it to send bit  $x_2$ . If for example we decide to send bit  $x_1$ , then receiver  $R_1$  will only receive  $x_1$ , while receiver  $R_2$  will receive both  $x_1$  and  $x_2$ .

The simple but important observation made in the seminal work by Ahlswede *et al.* is that we can allow intermediate nodes in the network to process their incoming information streams, and not just forward them. In particular, node  $C$  can take bits  $x_1$  and  $x_2$  and **xor** them to create a third bit  $x_3 = x_1 + x_2$  which it can then send through edge  $CE$  (the **xor** operation corresponds to addition over the binary field).  $R_1$  receives  $\{x_1, x_1 + x_2\}$ , and can solve this system of equations to retrieve  $x_1$  and  $x_2$ . Similarly,  $R_2$  receives  $\{x_2, x_1 + x_2\}$ , and can solve this system of equations to retrieve  $x_1$  and  $x_2$ .

---

The previous example shows that if we allow intermediate node in the network to combine information streams and extract the information at the receivers, we can increase the throughput when multicasting. This observation is generalized to the main theorem for multicasting in Chapter 2.

### Wireless Resources

In a wireless environment, network coding can be used to offer benefits in terms of battery life, wireless bandwidth, and delay.

---

**Example 1.2.** Consider a wireless ad-hoc network, where devices  $A$  and  $C$  would like to exchange the binary files  $x_1$  and  $x_2$  using device  $B$  as a relay. We assume that time is slotted, and that a device can

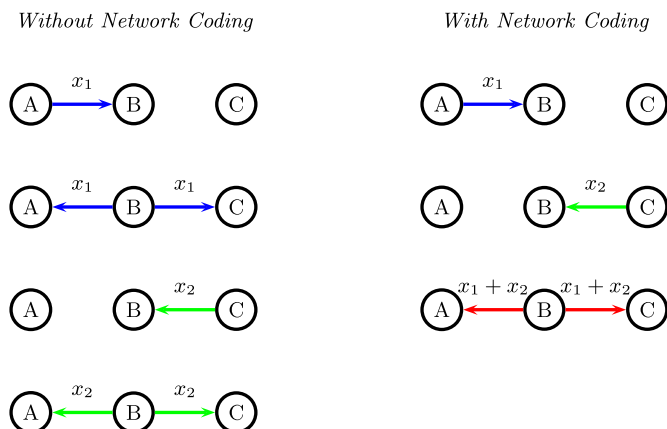


Fig. 1.3 Nodes  $A$  and  $C$  exchange information via relay  $B$ . The network coding approach uses one broadcast transmission less.

either transmit or receive a file during a timeslot (half-duplex communication). Figure 1.3 depicts on the left the standard approach: nodes  $A$  and  $C$  send their files to the relay  $B$ , who in turn forwards each file to the corresponding destination.

The network coding approach takes advantage of the natural capability of wireless channels for broadcasting to give benefits in terms of resource utilization, as illustrated in Figure 1.3. In particular, node  $C$  receives both files  $x_1$  and  $x_2$ , and bitwise xors them to create the file  $x_1 + x_2$ , which it then broadcasts to both receivers using a common transmission. Node  $A$  has  $x_1$  and can thus decode  $x_2$ . Node  $C$  has  $x_2$  and can thus decode  $x_1$ .

This approach offers benefits in terms of energy efficiency (node  $B$  transmits once instead of twice), delay (the transmission is concluded after three instead of four timeslots), wireless bandwidth (the wireless channel is occupied for a smaller amount of time), and interference (if there are other wireless nodes attempting to communicate in the neighborhood).

---

The benefits in the previous example arise from that broadcast transmissions are made maximally useful to all their receivers. Network coding for wireless is examined in the second part of this review. As we will

discuss there,  $x_1 + x_2$  is nothing but some type of binning or hashing for the pair  $(x_1, x_2)$  that the relay needs to transmit. Binning is not a new idea in wireless communications. The new element is that we can efficiently implement such ideas in practice, using simple algebraic operations.

### Security

Sending linear combinations of packets instead of uncoded data offers a natural way to take advantage of multipath diversity for security against wiretapping attacks. Thus systems that only require protection against such simple attacks, can get it “for free” without additional security mechanisms.

---

**Example 1.3.** Consider node  $A$  that sends information to node  $D$  through two paths  $ABD$  and  $ACD$  in Figure 1.4. Assume that an adversary (Calvin) can wiretap a single path, and does not have access to the complementary path. If the independent symbols  $x_1$  and  $x_2$  are sent uncoded, Calvin can intercept one of them. If instead linear combinations (over some finite field) of the symbols are sent through the different routes, Calvin cannot decode any part of the data. If for example he retrieves  $x_1 + x_2$ , the probability of his guessing correctly  $x_1$  equals 50%, the same as random guessing.

---

Similar ideas can also help to identify malicious traffic and to protect against Byzantine attacks, as we will discuss in the second part of this review.

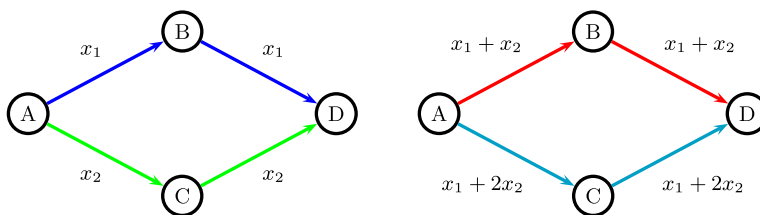


Fig. 1.4 Mixing information streams offers a natural protection against wiretapping.

### 1.1.2 Challenges

The deployment of network coding is challenged by a number of issues that will also be discussed in more detail throughout the review. Here we briefly outline some major concerns.

#### Complexity

Employing network coding requires nodes in the network to have additional functionalities.

---

**Example 1.4.** In Example 1.2, Figure 1.3, node  $B$  has additional memory requirements (needs to store file  $x_1$  instead of immediately broadcasting it), and has to perform operations over finite fields (bitwise `xor`  $x_1$  and  $x_2$ ). Moreover, nodes  $A$  and  $C$  need to also keep their own information stored, and then solve a system of linear equations.

---

An important question in network coding research today is assessing the complexity requirements of network coding, and investigating trade-offs between complexity and performance. We discuss such questions in Chapter 7.

#### Security

Networks where security is an important requirement, such as networks for banking transactions, need to guarantee protection against sophisticated attacks. The current mechanisms in place are designed around the assumption that the only eligible entities to tamper with the data are the source and the destination. Network coding on the other hand requires intermediate routers to perform operations on the data packets. Thus deployment of network coding in such networks would require to put in place mechanisms that allow network coding operations without affecting the authenticity of the data. Initial efforts toward this goal are discussed in the second part of the review.

### **Integration with Existing Infrastructure**

As communication networks evolve toward an ubiquitous infrastructure, a challenging task is to incorporate the emerging technologies such as network coding, into the existing network architecture. Ideally, we would like to be able to profit from the leveraged functionalities network coding can offer, without incurring dramatic changes in the existing equipment and software. A related open question is, how could network coding be integrated in current networking protocols. Making this possible is also an area of current research.

# 2

---

## The Main Theorem of Network Multicast

---

Network multicast refers to simultaneously transmitting the same information to multiple receivers in the network. We are concerned with sufficient and necessary conditions that the network has to satisfy to be able to support the multicast at a certain rate. For the case of unicast (when only one receiver at the time uses the network), such conditions have been known for the past 50 years, and, clearly, we must require that they hold for each receiver participating in the multicast. The fascinating fact that the main network coding theorem brings is that the conditions necessary and sufficient for unicast at a certain rate to each receiver are also necessary and sufficient for multicast at the same rate, provided the intermediate network nodes are allowed to combine and process different information streams. We start this chapter with a suitable single unicast scenario study, and then move to multicast. We use an algebraic approach to present and prove the network multicast theorem, and in the process, introduce basic notions in network coding.

## 2.1 The Min-Cut Max-Flow Theorem

Let  $G = (V, E)$  be a graph (network) with the set of vertices  $V$  and the set of edges  $E \subset V \times V$ . We assume that each edge has unit capacity, and allow parallel edges. Consider a node  $S \in V$  that wants to transmit information to a node  $R \in V$ .

---

**Definition 2.1.** A *cut* between  $S$  and  $R$  is a set of graph edges whose removal disconnects  $S$  from  $R$ . A *min-cut* is a cut with the smallest (minimal) value. The *value* of the cut is the sum of the capacities of the edges in the cut.

---

For unit capacity edges, the value of a cut equals the number of edges in the cut, and it is sometimes referred to as the size of the cut. We will use the term min-cut to refer to both the set of edges and to their total number. Note that there exists a unique min-cut value, and possibly several min-cuts, see for example Figure 2.1.

One can think about a min-cut as being a bottleneck for information transmission between source  $S$  and receiver  $R$ . Indeed, the celebrated max-flow, min-cut theorem, which we discuss below, claims that the maximum information rate we can send from  $S$  to  $R$  is equal to the min-cut value.

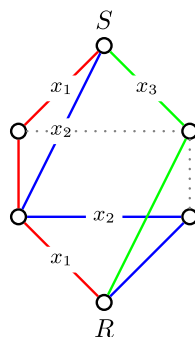


Fig. 2.1 A unicast connection over a network with unit capacity edges. The min-cut between  $S$  and  $R$  equals three. There exist three edge disjoint paths between  $S$  and  $R$  that bring symbols  $x_1$ ,  $x_2$ , and  $x_3$  to the receiver.

---

**Theorem 2.1.** Consider a graph  $G = (V, E)$  with unit capacity edges, a source vertex  $S$ , and a receiver vertex  $R$ . If the min-cut between  $S$  and  $R$  equals  $h$ , then the information can be sent from  $S$  to  $R$  at a maximum rate of  $h$ . Equivalently, there exist exactly  $h$  edge-disjoint paths between  $S$  and  $R$ .

---

We next give a constructive proof that, under the conditions of the theorem, there exist exactly  $h$  edge-disjoint paths between  $S$  and  $R$ . Since these paths are made up of unit capacity edges, information can be sent through each path at unit rate, giving the cumulative rate  $h$  over all paths. The procedure we present for constructing edge-disjoint paths between  $S$  and  $R$  is a part of the first step for all network code design algorithms that we discuss in Chapter 5.

*Proof.* Assume that the min-cut value between  $S$  and  $R$  equals  $h$ . Clearly, we cannot find more than  $h$  edge disjoint paths, as otherwise removing  $h$  edges would not disconnect the source from the receiver. We prove the opposite direction using a “path-augmenting” algorithm. The algorithm takes  $h$  steps; in each step we find an additional unit rate path from the source to the receiver.

Let  $p_{uv}^e$  be an indicator variable associated with an edge  $e$  that connects vertex  $u \in V$  to vertex  $v \in V$ . (There may exist multiple edges that connect  $u$  and  $v$ .)

Step 0: Initially, set  $p_{uv}^e = 0$  for all edges  $e \in E$ .

Step 1: Find a path  $P_1$  from  $S$  to  $R$ ,  $P_1 = \{v_0^1 = S, v_1^1, v_2^1, \dots, v_{\ell_1}^1 = R\}$ , where  $\ell_1$  is the length of the path, and set  $p_{v_i^1 v_{i+1}^1}^e = 1$ ,  $0 \leq i < \ell_1$ , using one edge between every consecutive pair of nodes. The notation  $p_{v_i^1 v_{i+1}^1}^e = 1$  indicates that the edge  $e$  has been used in the direction from  $v_i^1$  to  $v_{i+1}^1$ .

Step  $k$ : ( $2 \leq k \leq h$ ) Find a path  $P_k = \{v_0^k = S, v_1^k, v_2^k, \dots, v_{\ell_k}^k = R\}$  of length  $\ell_k$  so that the following condition holds:

*There exists edge  $e$  between  $v_i^k, v_{i+1}^k$ ,  $0 \leq i < \ell_k$  such that*

$$p_{v_i^k v_{i+1}^k}^e = 0 \quad \text{or} \quad p_{v_{i+1}^k v_i^k}^e = 1. \quad (2.1)$$

Accordingly, set  $p_{v_i^k v_{i+1}^k}^e = 1$  or  $p_{v_{i+1}^k v_i^k}^e = 0$ . This means that each newly found path uses only edges which either have not been used or have been used in the opposite direction by the previous paths.

Note that each step of the algorithm increases the number of edge disjoint paths that connect the source to the receiver by one, and thus at the end of step  $k$  we have identified  $k$  edge disjoint paths.

To prove that the algorithm works, we need to prove that, at each step  $k$ , with  $1 \leq k \leq h$ , there will exist a path whose edges satisfy the condition in (2.1). We will prove this claim by contradiction.

- (1) Assume that the min-cut to the receiver is  $h$ , but at step  $k \leq h$  we cannot find a path satisfying (2.1).
- (2) We will recursively create a subset  $\mathcal{V}$  of the vertices of  $V$ . Initially  $\mathcal{V} = \{S\}$ . If for a vertex  $v \in V$  there exists an edge connecting  $v$  and  $S$  that satisfies (2.1), include  $v$  to  $\mathcal{V}$ . Continue adding to  $\mathcal{V}$  vertices  $v \in V$  such that, for some  $u \in \mathcal{V}$ , there exists an edge between  $u$  and  $v$  that satisfies the condition in (2.1) (namely,  $p_{uv}^e = 0$  or  $p_{vu}^e = 1$ ), until no more vertices can be added.
- (3) By the assumption in (1), we know that  $\mathcal{V}$  does not contain the receiver  $R$ , otherwise we would have found the desired path. That is, the receiver belongs to  $\bar{\mathcal{V}} = V \setminus \mathcal{V}$ . Let  $\partial\mathcal{V} = \{e | e = (u, v) \in E \text{ with } u \in \mathcal{V}, v \in \bar{\mathcal{V}}\}$  denote the set of all edges  $e$  that connect  $\mathcal{V}$  with  $\bar{\mathcal{V}}$ . These edges form a cut. By the construction of  $\mathcal{V}$ ,  $p_{uv}^e = 1$  and  $p_{vu}^e = 0$  for all edges  $e \in \partial\mathcal{V}$ . But from (1),  $\sum_{e \in \partial\mathcal{V}} p_{uv}^e \leq k - 1$ , and thus there exists a cut of value at most  $k - 1 < h$ , which contradicts the premise of the theorem. □

## 2.2 The Main Network Coding Theorem

We now consider a multicast scenario over a network  $G = (V, E)$  where  $h$  unit rate sources  $S_1, \dots, S_h$  located on the same network node  $S$  (source) simultaneously transmit information to  $N$  receivers

$R_1, \dots, R_N$ . We assume that  $G$  is an *acyclic directed* graph with *unit capacity* edges, and that the value of the min-cut between the source node and each of the receivers is  $h$ . For the moment, we also assume *zero delay*, meaning that during each time slot all nodes simultaneously receive all their inputs and send their outputs.

**What we mean by unit capacity edges:** The unit capacity edges assumption models a transmission scenario in which time is slotted, and during each time-slot we can reliably (with no errors) transmit through each edge a symbol from some finite field  $\mathbb{F}_q$  of size  $q$ . Accordingly, each unit rate source  $S_i$  emits  $\sigma_i$ ,  $1 \leq i \leq h$ , which is an element of the same field  $\mathbb{F}_q$ . In practice, this model corresponds to the scenario in which every edge can reliably carry one bit and each source produces one bit per time-unit. Using an alphabet of size, say,  $q = 2^m$ , simply means that we send the information from the sources in packets of  $m$  bits, with  $m$  time-units being defined as one time-slot. The  $m$  bits are treated as one symbol of  $\mathbb{F}_q$  and processed using operations over  $\mathbb{F}_q$  by the network nodes. We refer to such transmission mechanisms as transmission schemes over  $\mathbb{F}_q$ . When we say that a scheme exists “over a large enough finite field  $\mathbb{F}_q$ ,” we effectively mean that there exists “a large enough packet length.”

### 2.2.1 Statement of the Theorem

We state the main theorem in network coding as follows:

---

**Theorem 2.2.** Consider a directed acyclic graph  $G = (V, E)$  with unit capacity edges,  $h$  unit rate sources located on the same vertex of the graph and  $N$  receivers. Assume that the value of the min-cut to each receiver is  $h$ . Then there exists a multicast transmission scheme over a large enough finite field  $\mathbb{F}_q$ , in which intermediate network nodes linearly combine their incoming information symbols over  $\mathbb{F}_q$ , that delivers the information from the sources simultaneously to each receiver at a rate equal to  $h$ .

---

From the min-cut max-flow theorem, we know that there exist exactly  $h$  edge-disjoint paths between the sources and each of the

receivers. Thus, if any of the receivers, say,  $R_j$ , is using the network by itself, the information from the  $h$  sources can be routed to  $R_j$  through a set of  $h$  edge disjoint paths. When multiple receivers are using the network simultaneously, their sets of paths may overlap. The conventional wisdom says that the receivers will then have to share the network resources, (e.g., share the overlapping edge capacity or share the access to the edge in time), which leads to reduced rates. However, Theorem 2.2 tells us that, if we allow intermediate network nodes to not only forward but also combine their incoming information flows, then each of the receivers will be getting the information at the same rate as if it had sole access to network resources.

The theorem additionally claims that it is sufficient for intermediate nodes to perform linear operations, namely, additions and multiplications over a finite field  $\mathbb{F}_q$ . We will refer to such transmission schemes as *linear network coding*. Thus the theorem establishes the existence of linear network codes over some large enough finite field  $\mathbb{F}_q$ . To reduce computational complexity, the field  $\mathbb{F}_q$  should be chosen as small as possible. In order to provide a very simple proof of Theorem 2.2, we will not worry, for a moment, about being efficient in terms of the field size. This question will be addressed in Chapter 7, where we discuss resources required for network coding.

### 2.2.2 An Equivalent Algebraic Statement of the Theorem

In order to route the  $h$  information sources to a particular receiver, we first have to find  $h$  edge-disjoint paths that connect the source node to this receiver. We can do that by using, for example, the algorithm given within the proof of the min-cut max-flow theorem, which is in fact a special case of the Ford–Fulkerson algorithm. In the multicast case, we have to find one set of such paths for each of the  $N$  receivers. Note that the paths from different sets may overlap. The example in Figure 2.2 shows a network with two sources and three receivers. Each subfigure shows a set of two edge disjoint paths for different receivers. Observe that the paths toward different receivers overlap over edges  $BD$  and  $GH$ .

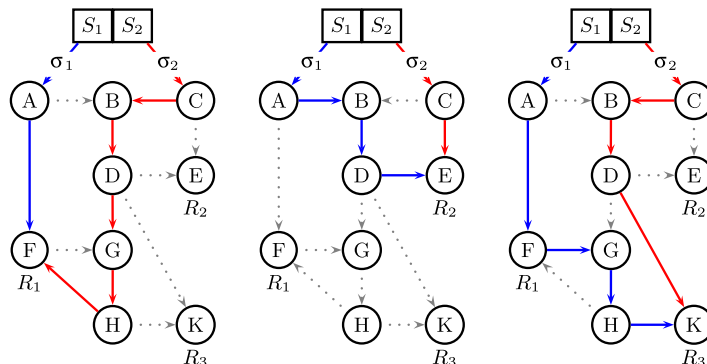


Fig. 2.2 The paths to the three receivers overlap for example over edges  $BD$  and  $GH$ .

If we were restricted to routing, when two paths bringing symbols  $\sigma_i$  and  $\sigma_j$  from sources  $S_i$  and  $S_j$ ,  $i \neq j$ , overlap over a unit capacity edge, we could forward only one of these two symbols (or timeshare between them). In linear network coding, instead, we can transmit through the shared edge a linear combination of  $\sigma_i$  and  $\sigma_j$  over  $\mathbb{F}_q$ . Such operations may be performed several times throughout the network, that is, if paths bringing different information symbols use the same edge  $e$ , a linear combination of these symbols is transmitted through  $e$ . The coefficients used to form this linear combination constitute what we call a *local coding vector*  $c^\ell(e)$  for edge  $e$ .

---

**Definition 2.2.** The *local coding vector*  $c^\ell(e)$  associated with an edge  $e$  is the vector of coefficients over  $\mathbb{F}_q$  with which we multiply the incoming symbols to edge  $e$ . The dimension of  $c^\ell(e)$  is  $1 \times |\text{In}(e)|$ , where  $\text{In}(e)$  is the set of incoming edges to the parent node of  $e$ .

---

Since we do not know what values should the coefficients in the local coding vectors take, we assume that each used coefficient is an unknown variable, whose value will be determined later. For the example in Figure 2.2, the linear combination of information is shown in Figure 2.3. The local coding vectors associated with edges  $BD$  and  $GH$  are

$$c^\ell(BD) = [\alpha_1 \ \alpha_2] \quad \text{and} \quad c^\ell(GH) = [\alpha_3 \ \alpha_4].$$

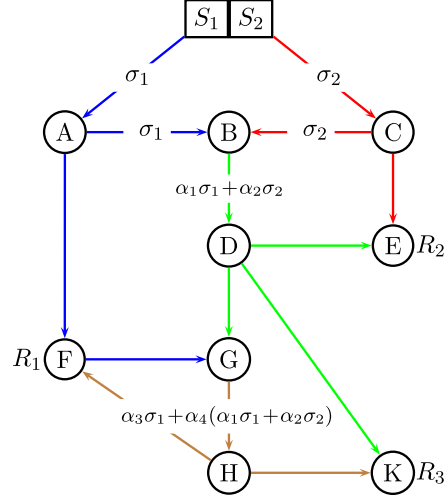


Fig. 2.3 The linear network coding solution sends over edges  $BD$  and  $GH$  linear combinations of their incoming flows.

We next discuss how this linear combining can enable the receivers to get the information at rate  $h$ .

Note that, since we start from the source symbols and then at intermediate nodes only perform linear combining of the incoming symbols, through each edge of  $G$  flows a linear combination of the source symbols. Namely, the symbol flowing through some edge  $e$  of  $G$  is given by

$$c_1(e)\sigma_1 + c_2(e)\sigma_2 + \dots + c_h(e)\sigma_h = \underbrace{[c_1(e) \ c_2(e) \ \dots \ c_h(e)]}_{c(e)} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_h \end{bmatrix},$$

where the vector  $c(e) = [c_1(e) \ c_2(e) \ \dots \ c_h(e)]$  belongs to an  $h$ -dimensional vector space over  $\mathbb{F}_q$ . We shall refer to the vector  $c(e)$  as the *global coding vector* of edge  $e$ , or for simplicity as the *coding vector*.

---

**Definition 2.3.** The *global coding vector*  $c(e)$  associated with an edge  $e$  is the vector of coefficients of the source symbols that flow (linearly combined) through edge  $e$ . The dimension of  $c(e)$  is  $1 \times h$ .

---

The coding vectors associated with the input edges of a receiver node define a system of linear equations that the receiver can solve to determine the source symbols. More precisely, consider receiver  $R_j$ . Let  $\rho_i^j$  be the symbol on the last edge of the path  $(S_i, R_j)$ , and  $\mathbf{A}_j$  the matrix whose  $i$ th row is the coding vector of the last edge on the path  $(S_i, R_j)$ . Then the receiver  $R_j$  has to solve the following system of linear equations:

$$\begin{bmatrix} \rho_1^j \\ \rho_2^j \\ \vdots \\ \rho_h^j \end{bmatrix} = \mathbf{A}_j \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_h \end{bmatrix} \quad (2.2)$$

to retrieve the information symbols  $\sigma_i$ ,  $1 \leq i \leq h$ , transmitted from the  $h$  sources. Therefore, choosing global coding vectors so that all  $\mathbf{A}_j$ ,  $1 \leq j \leq N$ , are full rank will enable all receivers to recover the source symbols from the information they receive. There is one more condition that these vectors have to satisfy: the global coding vector of an output edge of a node has to lie in the linear span of the coding vectors of the node's input edges. For example, in Figure 2.3, the coding vector  $c(GH)$  is in the linear span of  $c(DG)$  and  $c(FG)$ .

Alternatively, we can deal with local coding vectors. Clearly, given all the local coding vectors for a network, we can compute global coding vectors, and vice versa. The global coding vectors associated with edges  $BD$  and  $GH$  in Figure 2.3 are

$$c(BD) = [\alpha_1 \ \alpha_2] \quad \text{and} \quad c(GH) = [\alpha_3 + \alpha_1\alpha_4 \ \alpha_2\alpha_4].$$

Consequently, matrices  $\mathbf{A}_j$  can be expressed in terms of the components of the local coding vectors  $\{\alpha_k\}$ . For our example in Figure 2.3, the three receivers observe the linear combinations of source symbols defined by the matrices

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 0 \\ \alpha_3 + \alpha_1\alpha_4 & \alpha_2\alpha_4 \end{bmatrix}, \quad \mathbf{A}_2 = \begin{bmatrix} 0 & 1 \\ \alpha_1 & \alpha_2 \end{bmatrix}, \quad \text{and} \\ \mathbf{A}_3 = \begin{bmatrix} \alpha_1 & \alpha_2 \\ \alpha_3 + \alpha_1\alpha_4 & \alpha_2\alpha_4 \end{bmatrix}.$$

The network code design problem is to select values for the coefficients  $\{\alpha_k\} = \{\alpha_1, \dots, \alpha_4\}$ , so that all matrices  $\mathbf{A}_j$ ,  $1 \leq j \leq 3$ , are full rank. The main multicast theorem can be expressed in algebraic language as follows:

---

**Algebraic Statement of the Main Theorem:** In linear network coding, there exist values in some large enough finite field  $\mathbb{F}_q$  for the components  $\{\alpha_k\}$  of the local coding vectors, such that all matrices  $\mathbf{A}_j$ ,  $1 \leq j \leq N$ , defining the information that the receivers observe, are full rank.

---

In the following section, we prove this equivalent claim of the main coding theorem for multicast.

### 2.2.3 Proof of the Theorem

The requirement that all  $\mathbf{A}_j$ ,  $1 \leq j \leq N$ , be full rank is equivalent to the requirement that the product of the determinants of  $\mathbf{A}_j$  be different from zero:

$$f(\{\alpha_k\}) \triangleq \det(\mathbf{A}_1) \det(\mathbf{A}_2) \cdots \det(\mathbf{A}_N) \neq 0, \quad (2.3)$$

where  $f(\{\alpha_k\})$  is used to denote this product as a function of  $\{\alpha_k\}$ .

Note that, because we assumed that the graph  $G$  is acyclic, i.e., there are no feedback loops, and because we only perform linear operations at intermediate nodes of the network, each entry of matrix  $\mathbf{A}_j$  is a polynomial in  $\{\alpha_k\}$ . We will also show this rigorously in the next chapter. Therefore, the function  $f(\{\alpha_k\})$  is also a polynomial in  $\{\alpha_k\}$  of finite total degree.

Next, we show that  $f(\{\alpha_k\})$  is not identically equal to zero on some field. Since  $f(\{\alpha_k\})$  is a product, it is sufficient to show that none of its factors, namely, none of the  $\det \mathbf{A}_j$ , is identically equal to zero. This follows directly from the min-cut max-flow theorem: simply find  $h$ -edge disjoint paths from the sources to receiver  $R_j$ , and then use value one for the coefficients corresponding to the edges of these paths, and value zero for the remaining coefficients. This coefficient assignment makes  $\mathbf{A}_j$  to be the  $h \times h$  identity matrix.

When dealing with polynomials over a finite field, one has to keep in mind that even a polynomial not identically equal to zero over that field can evaluate to zero on all of its elements. Consider, for example, the polynomial  $x(x+1)$  over  $\mathbb{F}_2$ , or the following set of matrices:

$$\mathbf{A}_1 = \begin{bmatrix} x^2 & x(1+x) \\ x(1+x) & x^2 \end{bmatrix} \quad \text{and} \quad \mathbf{A}_2 = \begin{bmatrix} 1 & x \\ 1 & 1 \end{bmatrix}. \quad (2.4)$$

Over  $\mathbb{F}_2$ , at least one of the determinants of these matrices is equal to zero (namely, for  $x=0$ ,  $\det(\mathbf{A}_1)=0$  and  $x=1$ ,  $\det(\mathbf{A}_2)=0$ ).

The following lemma tells us that we can find values for the coefficients  $\{\alpha_k\}$  over a large enough finite field such that the condition  $f(\{\alpha_k\}) \neq 0$  in (2.3) holds, and thus concludes our proof. For example, for the matrices in (2.4) we can use  $x=2$  over  $\mathbb{F}_3$  so that both  $\det(\mathbf{A}_1) \neq 0$  and  $\det(\mathbf{A}_2) \neq 0$ .

---

**Lemma 2.3 (Sparse Zeros Lemma).** Let  $f(\alpha_1, \dots, \alpha_\eta)$  be a multivariate polynomial in variables  $\alpha_1, \dots, \alpha_\eta$ , with maximum degree in each variable of at most  $d$ . Then, in every finite field  $\mathbb{F}_q$  of size  $q > d$  on which  $f(\alpha_1, \dots, \alpha_\eta)$  is not identically equal to zero, there exist values  $p_1, \dots, p_\eta$ , such that  $f(\alpha_1 = p_1, \dots, \alpha_\eta = p_\eta) \neq 0$ .

---

*Proof.* We prove the lemma by induction in the number of variables  $\eta$ :

- (1) For  $\eta = 1$ , we have a polynomial in a single variable of degree at most  $d$ . The lemma holds immediately since such polynomials can have at most  $d$  roots.
- (2) For  $\eta > 1$ , the inductive hypothesis is that the claim holds for all polynomials with fewer than  $\eta$  variables. We can express  $f(\alpha_1, \dots, \alpha_\eta)$  as a polynomial in variable  $\alpha_\eta$  with coefficients  $\{f_i\}$  that are polynomials in the remaining variables  $\alpha_1, \dots, \alpha_{\eta-1}$ :

$$f(\alpha_1, \dots, \alpha_\eta) = \sum_{i=0}^d f_i(\alpha_1, \dots, \alpha_{\eta-1}) \alpha_\eta^i. \quad (2.5)$$

Consider a field  $\mathbb{F}_q$  with  $q > d$  on which  $f(\alpha_1, \dots, \alpha_\eta)$  is not identically equal to zero. Then at least one of the polynomials  $\{f_i\}$  is not identically equal to zero on  $\mathbb{F}_q$ , say  $f_j$ .

Since the number of variables in  $f_j$  is smaller than  $\eta$ , by the inductive hypothesis, there exist values  $\alpha_1 = p_1, \dots, \alpha_{\eta-1} = p_{\eta-1}$  such that  $f_j(p_1, \dots, p_{\eta-1}) \neq 0$ . Substituting these values,  $f(p_1, \dots, p_{\eta-1}, \alpha_\eta)$  becomes a polynomial in the single variable  $\alpha_\eta$  with degree of at most  $d$  and at least one coefficient different than zero in  $\mathbb{F}_q$ . The lemma then holds from the argument in (1).  $\square$

Note that we have given a simple proof that network codes exist under certain assumptions, rather than a constructive proof. A number of questions naturally arise at this point, and we will discuss them in the next session. In Chapter 3, we will learn how to calculate the parameterized matrices  $\mathbf{A}_j$  for a given graph and multicast configuration, and give an alternative information-theoretical proof of the main theorem. In Chapter 5, we will look into the existing methods for network code design, i.e., methods to efficiently choose the linear coefficients.

#### 2.2.4 Discussion

Theorem 2.2 claims the existence of network codes for multicast, and immediately triggers a number of important questions. Can we construct network codes efficiently? How large does the operating field  $\mathbb{F}_q$  need to be? What are the complexity requirements of network coding? Do we expect to get significant benefits, i.e., is coding/decoding worth the effort? Do network codes exist for other types of network scenarios, such as networks with noisy links, or wireless networks, or traffic patterns other than multicast? What is the possible impact on applications? We will address some of these issues in later chapters.

In the rest of this chapter, we check how restrictive are the modeling assumptions we made, that is, whether the main theorem would hold or could be easily extended if we removed or relaxed these assumptions.

**Unit capacity edges** – *not restrictive*, provided that the capacity of the edges are rational numbers and we allow parallel edges.

**Collocated sources** – *not restrictive*. We can add to the graph an artificial common source vertex, and connect it to the  $h$  source vertices through unit capacity edges.

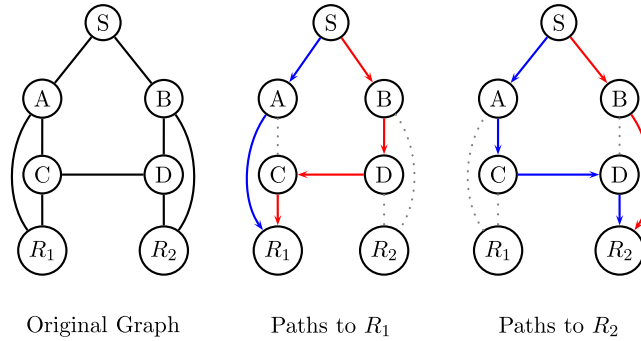


Fig. 2.4 An undirected graph where the main theorem does not hold.

**Directed graph – restrictive.** The gain in network coding comes from combining information carried by different paths whenever they overlap on an edge. However, in undirected graphs, different paths may traverse the same edge in opposite direction. This is the case in the example in Figure 2.4 for edge  $CD$ . For this example, network coding does not offer any benefits: we can achieve the maximal rate of 1.5 to each receiver by time-sharing across edge  $CD$ . On the other hand, there exist undirected graphs where use of network coding offers benefits. Such an example is provided by the butterfly network in Figure 1.2, if we assume that the edges of the network are undirected. Without network coding we can achieve rate 1.5 to each receiver, while using network coding we can achieve rate 2.

**Zero delay – not restrictive.** All practical networks introduce delay. From a theoretical point of view, delay introduces a relationship between network coding and convolutional codes. We discuss networks with delay in Chapter 6.

**Acyclic graph – not restrictive.** Note that the underlying graph for the example in Figure 2.2 contains the cycle  $\{FG, GH, HF\}$ , and this fact did not come up in our discussion. However, there exist network configurations where we need to deal with the underlying cyclic graphs by taking special action, for example by introducing memory, as we discuss in Chapter 6.

**Same min-cut values – restrictive.** If the receivers have different min-cuts, we can multicast at the rate equal to the minimum of the

min-cuts, but cannot always transmit to each receiver at the rate equal to its min-cut. For example, consider again the butterfly network in Figure 1.2 and assume that each vertex of the graph is a receiver. We will then have receivers with min-cut two, and receivers with min-cut one. Receivers with min-cut two require that linear combining is performed somewhere in the network, while receivers with min-cut one require that only routing is employed. These two requirements are not always compatible.

### Notes

The min-cut max-flow theorem was proven in 1927 by Menger [40]. It was also proven in 1956 by Ford and Fulkerson [22] and independently the same year by Elias *et al.* [20]. There exist many different variations of this theorem, see for example [18]. The proof we gave follows the approach in [8], and it is specific to unit capacity undirected graphs. The proof for arbitrary capacity edges and/or directed graphs is a direct extension of the same ideas.

The main theorem in network coding was proved by Ahlswede *et al.* [2, 36] in a series of two papers in 2000. The first origin of these ideas can be traced back to Yeung's work in 1995 [48]. Associating a random variable with the unknown coefficients when performing linear network coding was proposed by Koetter and Médard [32]. The sparse zeros lemma is published by Yeung *et al.* in [49] and by Harvey in [25], and is a variation of a result proved in 1980 by Schwartz [45]. The proof approach of the main theorem also follows the proof approach in [25, 49].

# 3

---

## Theoretical Frameworks for Network Coding

---

Network coding can and has been studied within a number of different theoretical frameworks, in several research communities. The choice of framework a researcher makes most frequently depends on his background and preferences. However, one may also argue that each network coding issue (e.g., code design, throughput benefits, complexity) should be put in the framework in which it can be studied the most naturally and efficiently.

We here present tools from the algebraic, combinatorial, information theoretic, and linear programming frameworks, and point out some of the important results obtained within each framework. At the end, we discuss different types of routing and coding. Multicasting is the only traffic scenario examined in this chapter. However, the presented tools and techniques can be extended to and are useful for studying other more general types of traffic as well.

### 3.1 A Network Multicast Model

#### 3.1.1 The Basic Model

A multicast scenario is described by a directed graph  $G = (V, E)$ , a source vertex  $S \in V$  (on which  $h$  unit rate sources  $S_i$  are collocated),

and a set  $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$  of  $N$  receivers. We will refer to these three ingredients together as a (multicast) *instance*  $\{G, S, \mathcal{R}\}$ . From the main theorem on network coding (Theorem 2.2), we know that a necessary and sufficient condition for feasibility of rate  $h$  multicast is that the min-cut to each receiver be greater or equal to  $h$ . We call this condition the *multicast property for rate  $h$* .

---

**Definition 3.1.** An instance  $\{G, S, \mathcal{R}\}$  satisfies the *multicast property for rate  $h$*  if the min-cut value between  $S$  and each receiver is greater than or equal to  $h$ .

---

Let  $\{(S_i, R_j), 1 \leq i \leq h\}$  be a set of  $h$  edge-disjoint paths from the sources to the receiver  $R_j$ . Under the assumption that the min-cut to each receiver equals at least  $h$ , the existence of such paths is guaranteed by the min-cut max-flow theorem. The choice of the paths is not unique, and, as we discuss later, affects the complexity of the network code. Our object of interest is the subgraph  $G'$  of  $G$  consisting of the  $hN$  paths  $(S_i, R_j)$ ,  $1 \leq i \leq h$ ,  $1 \leq j \leq N$ . Clearly, the instance  $\{G', S, \mathcal{R}\}$  also satisfies the multicast property.

As discussed in Chapter 2, in linear network coding, each node of  $G'$  receives an element of  $\mathbb{F}_q$  from each input edge, and then forwards (possibly different) linear combinations of these symbols to its output edges. The coefficients defining the linear combination on edge  $e$  are collected in a local coding vector  $c^\ell(e)$  of dimension  $1 \times |\text{In}(e)|$ , where  $\text{In}(e)$  is the set of edges incoming to the parent node of  $e$  (see Definition 2.2). As a consequence of this local linear combining, each edge  $e$  carries a linear combination of the source symbols, and the coefficients describing this linear combination are collected in the  $h$ -dimensional global coding vector  $c(e) = [c_1(e) \cdots c_h(e)]$  (see Definition 2.3). The symbol through edge  $e$  is given by

$$c_1(e)\sigma_1 + \cdots + c_h(e)\sigma_h.$$

Receiver  $R_j$  takes  $h$  coding vectors from its  $h$  input edges to form the rows of the matrix  $\mathbf{A}_j$ , and solves the system of linear equations (2.2). Network code design is concerned with assigning local coding vectors, or equivalently, coding vectors, to each edge of the graph.

---

**Definition 3.2.** An assignment of coding vectors is *feasible* if the coding vector of an edge  $e$  lies in the linear span of the coding vectors of the parent edges  $\text{In}(e)$ . A *valid* linear network code is any feasible assignment of coding vectors such that the matrix  $\mathbf{A}_j$  is full rank for each receiver  $R_j$ ,  $1 \leq j \leq N$ .

---

We have seen that for multicast networks satisfying the conditions of the main network coding theorem, a valid linear network code exists over some large enough field. There are, however, some other network traffic scenarios for which there are no valid linear network codes, and then some for which there are neither linear nor nonlinear valid network codes.

---

**Definition 3.3.** A network is *solvable* if there exist operations the nodes of the network can perform so that each receiver experiences the same maximum rate as if it were using all the network resources by itself. A network is *linearly solvable* if these operations are linear.

---

All multicast instances are, therefore, linearly solvable.

Observe now that in linear network coding, we need to perform linear combining at edge  $e$ , only if there exist two or more paths that share  $e$  but use distinct edges of  $\text{In}(e)$ . We then say that edge  $e$  is a *coding point*. In practice, these are the places in the network where we require additional processing capabilities, as opposed to simple forwarding.

---

**Definition 3.4.** *Coding points* are the edges of the graph  $G'$  where we need to perform network coding operations.

---

Finally, we will be interested in minimal graphs, namely, those that do not have redundant edges.

---

**Definition 3.5.** A graph  $G$  is called *minimal* with the multicast property if removing any edge would violate this property.

---

Identifying a minimal graph before multicasting may allow us to use less network resources and reduce the number of coding points as the following example illustrates.

---

**Example 3.1.** Consider the network with two sources and two receivers shown in Figure 3.1(a) that is created by putting one butterfly network on the top of another. A choice of two sets of edge-disjoint paths (corresponding to the two receivers) is shown in Figures 3.1(b) and 3.1(c). This choice results in using two coding points, i.e., linearly combining flows at edges  $AB$  and  $CD$ . Notice, however, that nodes  $A$  and  $B$  in Figure 3.1(a) and their incident edges can be removed without affecting the multicast condition. The resulting graph is then minimal, has a single coding point, and is identical to the butterfly network shown in Figure 1.2.

---

We can extend the same construction by stacking  $k$  butterfly networks to create a non-minimal configuration with  $k$  coding points. On the other hand, as we prove in Section 3.3, minimal configurations with two sources and two receivers have at most one coding point. Thus, identifying minimal configurations can help to significantly reduce the number of coding points.

Identifying a minimal configuration for an instance  $\{G, S, \mathcal{R}\}$  can be always done in polynomial time, as we discuss in Chapter 5. Contrary

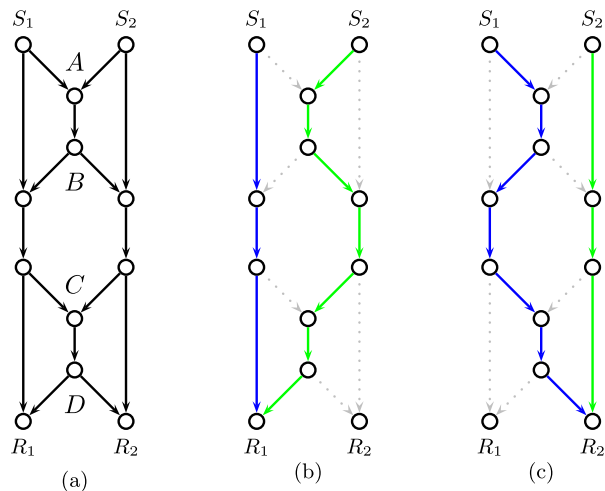


Fig. 3.1 A network with two sources and two receivers: (a) the original graph, (b) two edge-disjoint paths from the sources to the receiver  $R_1$ , and (c) two edge-disjoint paths from the sources to the receiver  $R_2$ .

to that, identifying a *minimum* configuration (that has the minimum number of coding points) is in most cases an NP-hard problem, as we discuss in Chapter 7.

### 3.1.2 The Line Graph Model

To define a network code, we eventually have to specify which linear combination of source symbols each edge carries. Thus, it is often more transparent to work with the graph

$$\gamma = \bigcup_{\substack{1 \leq i \leq h \\ 1 \leq j \leq N}} L(S_i, R_j), \tag{3.1}$$

where  $L(S_i, R_j)$  denotes the *line graph* of the path  $(S_i, R_j)$ , that is, each vertex of  $L(S_i, R_j)$  represents an edge of  $(S_i, R_j)$ , and any two vertices of  $L(S_i, R_j)$  are adjacent if and only if their corresponding edges share a common vertex in  $(S_i, R_j)$ . Figure 3.2 shows the network with two sources and three receivers which we studied in Chapter 2 together with its line graph.

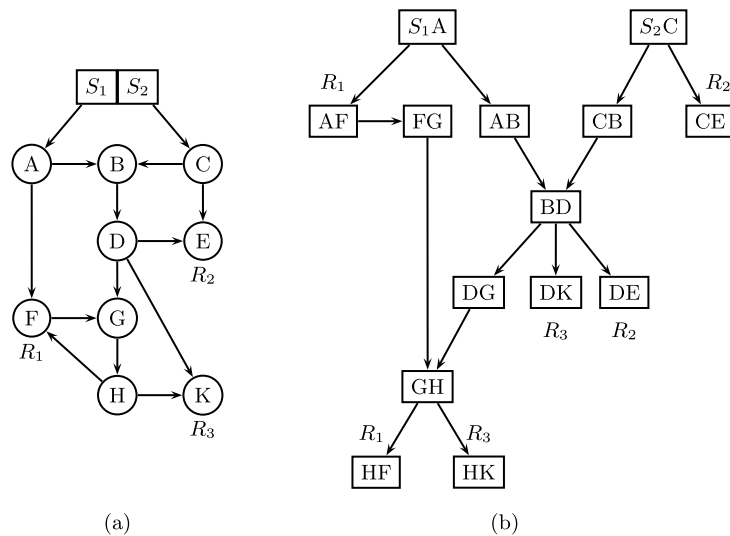


Fig. 3.2 A network with 2 sources and 3 receivers, and its line graph.

Without loss of generality (by possibly introducing an auxiliary node and  $h$  auxiliary edges), we can assume that the source vertex  $S$  in graph  $G'$  has exactly  $h$  outgoing edges, one corresponding to each of the  $h$  co-located sources. As a result, the line graph contains a node corresponding to each of the  $h$  sources. We refer to these nodes as *source nodes*. In Figure 3.2,  $S_1A$  and  $S_2C$  are source nodes.

---

**Definition 3.6.** *Source nodes*  $\{S_1^e, S_2^e, \dots, S_h^e\}$  are the nodes of  $\gamma$  corresponding to the  $h$  sources. Equivalently, they are the  $h$  edges in  $G'$  emanating from the source vertex  $S$ .

---

Each node in  $\gamma$  with a single input edge merely forwards its input symbol to its output edges. Each node with two or more input edges performs a coding operation (linear combining) on its input symbols, and forwards the result to all of its output edges. These nodes are the coding points in Definition 3.4. Using the line graph notation makes the definition of coding points transparent. In Figure 3.2(b),  $BD$  and  $GH$  are coding points.

Finally, we refer to the node corresponding to the last edge of the path  $(S_i, R_j)$  as the *receiver node* for receiver  $R_j$  and source  $S_i$ . For a configuration with  $h$  sources and  $N$  receivers, there exist  $hN$  receiver nodes. In Figure 3.2(b),  $AF$ ,  $HF$ ,  $HK$ ,  $DK$ ,  $DE$ , and  $CE$  are receiver nodes. Our definitions for feasible and valid network codes directly translate for line graphs, as well as our definition for minimality.

Note that vertices corresponding to the edges of  $\text{In}(e)$  are parent nodes of the vertex corresponding to  $e$ . The edges coming into the node  $e$  are labeled by the coefficients of local coding vector  $c^\ell(e)$ . Thus designing a network code amounts to choosing the values  $\{\alpha_k\}$  for the labels of edges in the line graph.

## 3.2 Algebraic Framework

One of the early approaches to network coding (and definitely the one that galvanized the area) was algebraic. Because the approach is very straightforward and requires little background, we adopted it for explaining the fundamentals in Chapter 2. Randomized coding,

believed to be of paramount importance for practical network coding, has been developed within the algebraic framework.

This approach is particularly easy to explain using the notion of the line graph in which each edge carries either label 1 or the unique label corresponding to a variable in  $\{\alpha_k\}$ . The main idea is to think of each vertex of the line graph (edge in the original graph) as a memory element that stores an intermediate information symbol. Then, if we consider a specific receiver  $R_j$ , our line graph acts as a linear system with  $h$  inputs (the  $h$  sources) and  $h$  outputs (that the receiver observes), and up to  $m := |E|$  memory elements. This system is described by the following set of finite-dimensional state-space equations:

$$\begin{aligned}\mathbf{s}_{k+1} &= \mathbf{A}\mathbf{s}_k + \mathbf{B}\mathbf{u}_k, \\ \mathbf{y}_k &= \mathbf{C}_j\mathbf{s}_k + \mathbf{D}_j\mathbf{u}_k,\end{aligned}\tag{3.2}$$

where  $\mathbf{s}_k$  is the  $m \times 1$  state vector,  $\mathbf{y}_k$  is the  $h \times 1$  output vector,  $\mathbf{u}_k$  is the  $h \times 1$  input vector, and  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\mathbf{C}_j$ , and  $\mathbf{D}_j$  are matrices with appropriate dimensions which we discuss below. (The state space equations (3.2) can also be interpreted as describing a convolutional code, as we explain in Chapter 6.) A standard result in linear system theory gives us the transfer matrix  $\mathbf{G}_j(\mathcal{D})$ :

$$\mathbf{G}_j(\mathcal{D}) = \mathbf{D}_j + \mathbf{C}_j(\mathcal{D}^{-1}\mathbf{I} - \mathbf{A})^{-1}\mathbf{B},\tag{3.3}$$

where  $\mathcal{D}$  is the indeterminate delay operator. Using unit delay, we obtain the transfer matrix for receiver  $R_j$ :

$$\mathbf{A}_j = \mathbf{D}_j + \mathbf{C}_j(\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}.\tag{3.4}$$

In (3.2) matrix  $\mathbf{A}$  is common for all receivers and reflects the way the memory elements (states) are connected (network topology). Its elements are indexed by the states (nodes of the line graph), and an element of  $\mathbf{A}$  is nonzero if and only if there is an edge in the line graph between the indexing states. The nonzero elements equal either to 1 or to an unknown variable in  $\{\alpha_k\}$ . Network code design amounts to selecting values for the variable entries in  $\mathbf{A}$ .

Matrix  $\mathbf{B}$  is also common for all receivers and reflects the way the inputs (sources) are connected to our graph. Matrices  $\mathbf{C}_j$  and  $\mathbf{D}_j$ ,

respectively, express how the outputs receiver  $R_j$  observes depend upon the state variables and the inputs. Matrices  $\mathbf{B}$ ,  $\mathbf{C}_j$ , and  $\mathbf{D}_j$  can be chosen to be binary matrices by possibly introducing auxiliary vertices and edges. Recall that the dimension of matrices  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\mathbf{C}_j$ , and  $\mathbf{D}_j$  depends upon the number of edges in our graph, which, in general, can be very large. In the next section, we discuss a method to significantly reduce this number.

By accordingly ordering the elements of the state space vector, matrix  $\mathbf{A}$  becomes strictly upper triangular for acyclic graphs, and therefore, nilpotent ( $\mathbf{A}^n = \mathbf{0}$  for some positive integer  $n$ ). Let  $L$  denote the length of the longest path between the source and a receiver. Then  $\mathbf{A}^{L+1} = \mathbf{0}$ . In other words,

$$(\mathbf{I} - \mathbf{A})^{-1} = \mathbf{I} + \mathbf{A} + \mathbf{A}^2 + \cdots + \mathbf{A}^L. \quad (3.5)$$

This equation immediately implies that the elements of the transfer matrices  $\mathbf{A}_j$  are polynomials in the unknown variables  $\{\alpha_k\}$ , a result that we used in the proof of the main theorem. Moreover, (3.5) offers an intuitive explanation of (3.4). It is easy to see that  $\mathbf{A}$  is effectively an incidence matrix. The series in (3.5) accounts for all paths connecting the network edges. The transfer matrix expresses the information brought along these paths from the sources to the receivers.

We will from now on, without loss of generality, assume that  $\mathbf{D}_j = \mathbf{0}$  (this we can always do by possibly adding auxiliary edges and increasing the size  $m$  of the state space). Observe that substituting  $\mathbf{D}_j = \mathbf{0}$  in (3.4) gives  $\mathbf{A}_j = \mathbf{C}_j(\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$ .

We next present a very useful lemma, helpful, for example, in some code design algorithms. Here, we use it to prove an upper bound on the field size (code alphabet size) sufficient for the existence of a network code.

---

**Lemma 3.1.** Let  $\mathbf{A}_j = \mathbf{C}_j(\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$ . Then for  $\mathbf{A}$  strictly upper triangular

$$|\det \mathbf{A}_j| = |\det \mathbf{N}_j|, \quad \text{where } \mathbf{N}_j = \begin{bmatrix} \mathbf{C}_j & \mathbf{0} \\ \mathbf{I} - \mathbf{A} & \mathbf{B} \end{bmatrix}. \quad (3.6)$$


---

*Proof.* Note that

$$\det \mathbf{N}_j = \det \begin{bmatrix} \mathbf{C}_j & \mathbf{0} \\ \mathbf{I} - \mathbf{A} & \mathbf{B} \end{bmatrix} = \pm \det \begin{bmatrix} \mathbf{0} & \mathbf{C}_j \\ \mathbf{B} & \mathbf{I} - \mathbf{A} \end{bmatrix},$$

since permuting columns of a matrix only affects the sign of its determinant. Now, using a result known as *Schur's formula* for the determinant of block matrices, we further obtain

$$\pm \det \mathbf{N}_j = \det \begin{bmatrix} \mathbf{0} & \mathbf{C}_j \\ \mathbf{B} & \mathbf{I} - \mathbf{A} \end{bmatrix} = \det(\mathbf{I} - \mathbf{A}) \det(\mathbf{C}_j(\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}).$$

The claim (3.6) follows directly from the above equation by noticing that, since  $\mathbf{A}$  is strictly upper triangular, then  $\det(\mathbf{I} - \mathbf{A}) = 1$ .  $\square$

---

**Theorem 3.2.** For a multicast network (under the model assumptions in Section 3.1) an alphabet of size  $q > N$  is always sufficient.

---

*Proof.* Each variable  $\alpha_k$  appears at most once in each matrix  $\mathbf{N}_j$ . As a result, the polynomial

$$f(\alpha_1, \dots, \alpha_\eta) = \det \mathbf{N}_1 \det \mathbf{N}_2 \cdots \det \mathbf{N}_N \quad (3.7)$$

has degree at most  $N$  in each of the  $\eta$  variables  $\{\alpha_k\}$ . From Lemma 5.6, for  $q > N$  there exist values  $p_1, p_2, \dots, p_\eta \in \mathbb{F}_q$  such that  $f(\alpha_1 = p_1, \dots, \alpha_\eta = p_\eta) \neq 0$ .  $\square$

We discuss how tight this bound is and present additional alphabet size bounds in Chapter 7.

### 3.3 Combinatorial Framework

We present an approach within the combinatorial framework that is mainly based on graph theory and to a certain extent on projective geometry. This approach has been useful for understanding computational complexity of network coding in terms of the required code

alphabet size and the number of coding points, which we discuss in Chapter 7. In this section, we show how we can use it to identify structural properties of the multicast configurations. We give this discussion in two levels. First we discuss the results and structural properties informally at a high level using the original graph  $G$ . At a second level, we use the line graph approach for a formal description and proofs.

### 3.3.1 Basic Notions of Subtree Decomposition

The basic idea is to partition the network graph into subgraphs through which the same information flows, as these are subgraphs within which we do not have to code. Each such subgraph is a tree, rooted at a coding point or a source, and terminating either at receivers or other coding points.

For the example in Figure 3.2, coding points are the edges  $BD$  and  $GH$ , and we can partition the graph into four trees. Through  $T_1 = \{S_1A, AB, AF, FG\}$  flows the source symbol  $\sigma_1$ , and through  $T_2 = \{S_2C, CB, CE\}$  flows the source symbol  $\sigma_2$ . Since source symbols flow through these trees, we call them source subtrees. Through  $T_3 = \{BD, DE, DG, DK\}$  flows the linear combination of source symbols that flows through the coding point  $BD$ , and similarly through  $T_4 = \{GH, HF, HK\}$  the linear combination that flows through the coding point  $GH$ . We call  $T_3$  and  $T_4$  coding subtrees.

---

**Definition 3.7.** A subtree  $T_i$  is called a *source subtree* if it starts with a source and a *coding subtree* if it starts with a coding point.

---

Each receiver  $R_j$  observes the linear combination of the sources that flow through the last edges on the paths  $(S_i, R_j)$ ,  $1 \leq i \leq h$ . We call these edges receiver nodes. Each receiver has  $h$  receiver nodes.

---

**Definition 3.8.** The *receiver nodes* for a receiver  $R_j$  are defined to be the last edges on the paths  $(S_i, R_j)$ ,  $1 \leq i \leq h$ .

---

In Figure 3.2 receiver  $R_1$  has the receiver nodes  $AF$  and  $HF$ .

For the network code design problem, we only need to know how the subtrees are connected and which receiver nodes are in each  $T_i$ . Thus we can contract each subtree to a node and retain only the edges that connect the subtrees. The resulting combinatorial object, which we will refer to as the *subtree graph*  $\Gamma$ , is defined by its underlying topology  $(V_\Gamma, E_\Gamma)$  and the association of the receivers with nodes  $V_\Gamma$ . Figure 3.3(b) shows the subtree graph for the network in Figure 3.2.

The network code design problem is now reduced to assigning an  $h$ -dimensional coding vector  $c(T_i) = [c_1(T_i) \cdots c_h(T_i)]$  to each subtree  $T_i$ . Receiver  $j$  observes  $h$  coding vectors from  $h$  distinct subtrees to form the rows of the matrix  $\mathbf{A}_j$ .

---

**Example 3.2.** A valid code for the network in Figure 3.2(a) can be obtained by assigning the following coding vectors to the subtrees in Figure 3.3(b):

$$c(T_1) = [1 \ 0], \quad c(T_2) = [0 \ 1], \quad c(T_3) = [1 \ 1], \quad \text{and} \quad c(T_4) = [0 \ 1].$$

For this code, the field with two elements is sufficient. Nodes  $B$  and  $G$  in the network (corresponding to coding points  $BD$  and  $GH$ ) perform binary addition of their inputs and forward the result of the operation.

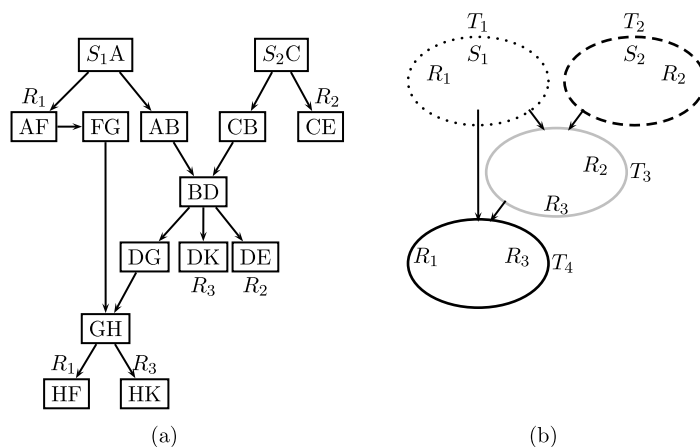


Fig. 3.3 (a) Line graph with coding points  $BD$  and  $GH$  for the network in Figure 3.2, and (b) the associated subtree graph.

The rest of the nodes in the network merely forward the information they receive. The matrices for receivers  $R_1$ ,  $R_2$ , and  $R_3$  are

$$\mathbf{A}_1 = \begin{bmatrix} c(T_1) \\ c(T_4) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{A}_2 = \begin{bmatrix} c(T_2) \\ c(T_3) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad \text{and}$$

$$\mathbf{A}_3 = \begin{bmatrix} c(T_3) \\ c(T_4) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

### 3.3.2 Formal Description of Subtree Decomposition

Using the line graph approach (see Figure 3.3), the subtree decomposition can be formally performed as follows. We partition the vertices of the line graph  $\gamma$  in (3.1) into subsets  $V_i^\ell$  so that the following holds:

- (1) Each  $V_i^\ell$  contains exactly one source node (see Definition 3.6) or a coding point (see Definition 3.4), and
- (2) Each node that is neither a source node nor a coding point belongs to the  $V_i^\ell$  which contains its closest ancestral coding point or source node.

$T_i$  is the subgraph of  $\gamma$  induced by  $V_i^\ell$  (namely,  $V_i^\ell$  together with the edges whose both endpoints are in  $V_i^\ell$ ). Source subtrees start with one of the source nodes  $\{S_1^e, S_2^e, \dots, S_h^e\}$  of the line graph and coding subtrees with one of the coding points.

We shall use the notation  $T_i$  to describe both the subgraph of the line graph  $\gamma$ , and the corresponding node in the subtree graph  $\Gamma$ , depending on the context.

### 3.3.3 Properties of Subtree Graphs

Here we prove a number of properties for subtree graphs and minimal subtree graphs, and use these properties to show that, for a given number of sources and receivers, there exists a finite number of corresponding subtree graphs. For example, for a configuration with two sources and two receivers there exists exactly one multicast configuration requiring network coding, which corresponds to the butterfly network.

---

**Theorem 3.3.** The subgraphs  $T_i$  satisfy the following properties:

- (1) Each subgraph  $T_i$  is a tree starting at either a coding point or a source node.
  - (2) The same linear combination of source symbols flows through all edges in the original graph that belong to the same  $T_i$ .
  - (3) For each receiver  $R_j$ , there exist  $h$  vertex-disjoint paths in the subtree graph from the source nodes to the receiver nodes of  $R_j$ .
  - (4) For each receiver  $R_j$ , the  $h$  receiver nodes corresponding to the last edges on the paths  $(S_i, R_j)$ ,  $1 \leq i \leq h$ , belong to distinct subtrees.
  - (5) Each subtree contains at most  $N$  receiver nodes, where  $N$  is the number of receivers.
- 

*Proof.* We prove the claims one by one.

- (1) By construction,  $T_i$  contains no cycles, the source node (or the coding point) has no incoming edges, and there exists a path from the source node (or the coding point) to each other vertex of  $T_i$ . Thus  $T_i$  is a tree rooted at the source node (or the coding point).
- (2) Since  $T_i$  is a tree, the linear combination that flows through the root (source node or coding point) also has to flow through the rest of the nodes.
- (3) Because the min-cut condition is satisfied in the original network, there exist  $h$  edge-disjoint paths to each receiver. Edge-disjoint paths in the original graph correspond to vertex-disjoint paths in the line graph  $\gamma$ .
- (4) Assume that the receiver nodes corresponding to the last edges of paths  $(S_k, R_j)$  and  $(S_\ell, R_j)$  belong to the same subtree  $T_i$ . Then both paths go through the root vertex (source node or coding point) of subtree  $T_i$ . But the paths for receiver  $R_j$  are vertex-disjoint.

- (5) This claim holds because there are  $N$  receivers and, by the previous claim, all receiver nodes contained in the same subtree are distinct.  $\square$

From definition, the multicast property is satisfied in  $\Gamma$  if and only if the min-cut condition is satisfied for every receiver in  $G$ . Thus the following holds:

---

**Lemma 3.4.** There is no valid codeword assignment (in the sense of Definition 3.2) for a subtree graph which does not satisfy the multicast property.

---

We use this lemma to show properties of minimal subtree graphs. Directly extending Definition 3.5, a subtree graph is called *minimal* with the multicast property if removing any edge would violate the multicast property.

---

**Theorem 3.5.** For a minimal subtree graph, the following holds:

- (1) There does not exist a valid network code where a subtree is assigned the same coding vector as one of its parents.
  - (2) There does not exist a valid network code where the vectors assigned to the parents of any given subtree are linearly dependent.
  - (3) There does not exist a valid network code where the coding vector assigned to a child belongs to a subspace spanned by a proper subset of the vectors assigned to its parents.
  - (4) Each coding subtree has at most  $h$  parents.
  - (5) If a coding subtree has  $2 \leq P \leq h$  parents, then there exist  $P$  vertex-disjoint paths from the source nodes to the subtree.
- 

*Proof.*

- (1) Suppose a subtree is assigned the same coding vector as one of its parents. Then removing the edge(s) between the subtree and the other parent(s) results in a subtree graph with

a valid coding vector assignment. By Lemma 3.4, the multicast property is also satisfied. But we started with a minimal subtree graph and removed edges without violating the multicast property, which contradicts minimality.

- (2) Suppose there is a subtree  $T$  whose parents  $T_1, \dots, T_P$  are assigned linearly dependent vectors  $c(T_1), \dots, c(T_P)$ . Without loss of generality, assume that  $c(T_1)$  can be expressed as a linear combination of  $c(T_2), \dots, c(T_P)$ . Then removing the edge between  $T$  and  $T_1$  results in a subtree graph with a valid coding vector assignment. From Lemma 3.4 the multicast property is also satisfied. But this contradicts minimality.
- (3) Suppose there is a subtree  $T$  whose parents  $T_1, \dots, T_P$  are assigned vectors  $c(T_1), \dots, c(T_P)$ . Without loss of generality, assume that  $T$  is assigned a vector  $c$  that is a linear combination of  $c(T_2), \dots, c(T_P)$ . Then removing the edge between  $T$  and  $T_1$  results in a subtree graph with a valid coding vector assignment. From Lemma 3.4 the multicast property is also satisfied. But this contradicts minimality.
- (4) Since coding vectors are  $h$ -dimensional, this claim is a direct consequence of claim (2).
- (5) By claim (2), the coding vectors assigned to the  $P$  parent subtrees must be linearly independent, which requires the existence of  $P$  vertex disjoint paths.  $\square$

The first three claims of Theorem 3.5 describe properties of valid codes for minimal subtree graphs, while the last two claims describe structural properties of minimal subtree graphs. The additional structural properties listed below for networks with two sources directly follow from Theorems 3.3 and 3.5.

---

**Theorem 3.6.** In a minimal subtree decomposition of a network with  $h = 2$  sources and  $N$  receivers:

- (1) A parent and a child subtree have a child or a receiver (or both) in common.

- (2) Each coding subtree contains at least two receiver nodes.
  - (3) Each source subtree contains at least one receiver node.
- 

*Proof.*

- (1) Suppose that the minimal subtree graph has two source and  $m$  coding subtrees. Consider a network code which assigns coding vectors  $[01]$  and  $[10]$  to the source subtrees, and a different vector  $[1\alpha^k]$ ,  $0 < k \leq m$ , to each of the coding subtrees, where  $\alpha$  is a primitive element of  $\mathbb{F}_q$  and  $q > m$ . As discussed in Appendix, any two different coding vectors form a basis for  $\mathbb{F}_q^2$ , and thus this code is feasible and valid. Let  $T_i$  and  $T_j$  denote a parent and a child subtree, respectively, with no child or receiver in common. Now, alter the code by assigning the coding vector of  $T_i$  to  $T_j$ , and keep the rest of coding vectors unchanged. This assignment is feasible because  $T_i$  and  $T_j$  do not share a child, which would have to be assigned a scalar multiple of the coding vector of  $T_i$  and  $T_j$ . Since  $T_i$  and  $T_j$  do not share a receiver, the code is still valid. Therefore, by claim (1) of Theorem 3.5, the configuration is not minimal, which contradicts the assumption.
- (2) Consider a coding subtree  $T$ . Let  $P_1$  and  $P_2$  be its parents. By claim (1), a parent and a child have either a receiver or a child in common. If  $T$  has a receiver in common with each of its two parents, then  $T$  has two receiver nodes. If  $T$  and one of the parents, say  $P_1$ , do not have a receiver in common, then they have a child in common, say  $C_1$ . Similarly, if  $T$  and  $C_1$  do not have receiver in common, then they have a child in common. And so forth, following the descendants of  $P_1$ , one eventually reaches a child of  $T$  that is a terminal node of the subtree graph, and thus has no children. Consequently,  $T$  has to have a receiver in common with this terminal subtree. Similarly, if  $T$  and  $P_2$  do not have a child in common, there exists a descendant of  $P_2$  and child of  $T$  which must have a receiver in common with  $T$ .

- (3) If the two source subtrees have no children, then each must contain  $N$  receiver nodes. If the source subtree has a child, say  $T_1$ , then by claim (1) it will have a receiver or a child in common with  $T_1$ . Following the same reasoning as in the previous claim, we conclude that each source subtree contains at least one receiver node.  $\square$

---

**Lemma 3.7.** For a network with two sources and two receivers, there exist exactly two minimal subtree graphs shown in Figure 3.4.

---

*Proof.* Figure 3.4(a) shows the network scenario in which each receiver has access to both source subtrees, and thus no network coding is required, i.e., there are no coding subtrees. If network coding is required, then by Theorem 3.6, there can only exist one coding subtree containing two receiver nodes. Figure 3.4(b) shows the network scenario in which each receiver has access to a different source subtree and a common coding subtree. This case corresponds to the familiar butterfly example of a network with two sources and two receivers.  $\square$

Therefore, all instances  $\{G, S, \mathcal{R}\}$  with two sources and two receivers where network coding is required, “contain” the butterfly network in Figure 1.2, and can be reduced to it in polynomial time by removing edges from the graph  $G$ .

Continuing along these lines, it is easy to show that for example there exist exactly three minimal configurations with two sources and

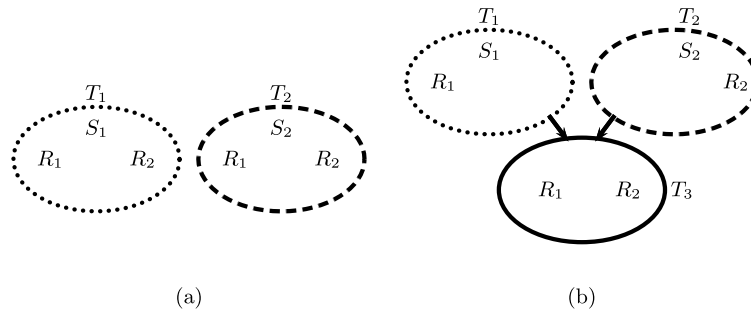


Fig. 3.4 Two possible subtree graphs for networks with two sources and two receivers: (a) no coding required, and (b) network coding necessary.

three receivers, and seven minimal configurations with two sources and four receivers. In fact, one can enumerate all the minimal configurations with a given number of sources and receivers.

### 3.4 Information-Theoretic Framework

The original approach to network coding as well as the original proof of the main theorem was information theoretic. Subsequently, both information theorists and computer scientists used information theoretic tools to establish bounds on achievable rates for the cases where the main theorem does not hold (e.g., to derive bounds for multiple unicast sessions traffic, as we discuss in part II of the review). We here briefly summarize the very beautiful proof idea used for the main theorem, and refer the interested reader to the literature for in depth and rigorous expositions on the information theoretic approach (see Section 3.6.3). The proof outline we give is high level and overlooks a number of technical details.

Consider the instance  $\{G = (V, E), S, \mathcal{R}\}$ . Let  $\text{In}(v)$  and  $\text{Out}(v)$  denote the set of incoming and outgoing edges for vertex  $v$ . For simplicity, we assume binary communication, that is, we can send only bits through the unit capacity edges of the network. Similar analysis goes through in the case of larger finite alphabets.

#### 3.4.1 Network Operation with Coding

The network operates as follows:

- (1) A source  $S$  of rate  $\omega_S$  produces  $B$  packets (messages), each containing  $n\omega_S$  information bits. It also selects  $|\text{Out}(S)|$  functions  $\{f_i^S\}$ ,

$$f_i^S : 2^{n\omega_S} \rightarrow 2^n,$$

which map the  $n\omega_S$  information bits into  $n$  coded bits to be transmitted over its outgoing edges.

- (2) Similarly, each vertex  $v \in V$  selects  $|\text{Out}(v)|$  functions  $\{f_i^v\}$ , one for each of its outgoing edges. Each function  $f_i^v$  has as its inputs  $|\text{In}(v)|$  packets of length  $n$ , and as its output one

packet of length  $n$

$$f_i^v : 2^{n|\text{In}(v)|} \rightarrow 2^n.$$

The functions  $\{f_i^S\}$  and  $\{f_i^v\}$  are chosen uniformly at random among all possible such mappings. For example, if we were to restrict the network nodes operation to linear processing over  $\mathbb{F}_2$ , each function  $f_i^v$  would be defined by a randomly selected binary matrix of dimension  $|\text{In}(v)|n \times n$ .

- (3) The network is clocked: at time  $k$  of the clock, for  $1 \leq k \leq B$ , the source  $S$  maps one of its  $B$  information packets denoted by  $m_k$  to  $|\text{Out}(S)|$  packets, and sends these packets through its outgoing edges. After time  $B$ , the source stops transmission.
- (4) For each information packet  $m_k$ ,  $1 \leq k \leq B$ , each other (non-source) vertex  $v \in V$  waits until it receives all  $|\text{In}(v)|$  incoming packets that depend only on packet  $m_k$ . It then (by the means of functions  $\{f_i^v\}$ ) computes and sends  $|\text{Out}(v)|$  packets through its outgoing edges, also depending only on packet  $m_k$ .
- (5) For each information packet  $m_k$ ,  $1 \leq k \leq B$ , each receiver  $R_j$  gets  $|\text{In}(R_j)|$  packets, which it uses to decode the source packet  $m_k$ , based on its knowledge of the network topology and all the mapping functions that the source and the intermediate nodes employ.
- (6) The receivers decode all  $B$  source packets during at most  $B + |V|$  time slots. Consequently, the received information rate is

$$\frac{n\omega_S B}{n(B + |V|)} \rightarrow \omega_S \quad \text{for } B \gg |V|.$$

### 3.4.2 The Information Rate Achievable with Coding

We now argue that the previously described network operation allows the receivers to successfully decode the source messages, provided that the rate  $\omega_S$  is smaller than the min-cut between the source and each receiver.

- (1) Consider two distinct source messages  $m$  and  $m' \in \{1,0\}^{n\omega_S}$ . We first calculate the probability  $\Pr(m, m')$  that a specific

receiver  $R_j$  is not going to be able to distinguish between  $m$  and  $m'$ . That is, we calculate the pairwise probability of error.

- (2) Define  $\mathcal{V} = \mathcal{V}(m, m')$  to be the set of vertices in  $V$  that cannot distinguish between source messages  $m$  and  $m'$ . This means that, for all  $v \in \mathcal{V}$ , all the  $\text{In}(v)$  incoming packets corresponding to messages  $m$  and  $m'$  are the same. Clearly,  $S$  belongs to  $\bar{\mathcal{V}} = V \setminus \mathcal{V}$  (the complement of  $\mathcal{V}$  in  $V$ ). Moreover, we have an error at receiver  $R_j$  if and only if  $R_j \in \mathcal{V}$ . Let  $\vartheta\mathcal{V}$  denote the edges that connect  $\bar{\mathcal{V}}$  with  $\mathcal{V}$ , that is,  $\vartheta\mathcal{V} = \{e | e = (v, u) \in E \text{ with } v \in \bar{\mathcal{V}}, u \in \mathcal{V}\}$ . When an error occurs,  $\vartheta\mathcal{V}$  defines a cut between the source and the receiver  $R_j$ .
- (3) We now calculate the probability that such a cut occurs. For every edge  $e = (v, u) \in \vartheta\mathcal{V}$ ,  $v$  can distinguish between  $m$  and  $m'$  but  $u$  cannot. Thus,  $v$  receives from its input edges two distinct sets of packets for  $m$  and  $m'$ , and maps them using the randomly chosen  $f_e^v$  to the same packet for edge  $e$ . Since the mappings are chosen uniformly at random, this can happen with probability  $2^{-n}$ . Furthermore, since the mappings for every edge are chosen independently, the probability that, at all the edges in  $\vartheta\mathcal{V}$ , messages  $m$  and  $m'$  are mapped to the same point equals  $2^{-n|\vartheta\mathcal{V}|}$ .
- (4) We have  $2^{n\omega_S}$  distinct messages in total, and thus, by the union bound, the total probability of error does not exceed  $2^{-n|\vartheta\mathcal{V}|} 2^{n\omega_S}$ . Therefore, provided that  $2^{-n(|\vartheta\mathcal{V}| - \omega_S)}$  decays to zero, the probability of error also decays to zero. For this to happen, the condition  $|\vartheta\mathcal{V}| > \omega_S$  is sufficient. In other words, if  $m = \min_{\mathcal{V}} |\vartheta\mathcal{V}|$  is the min-cut value between the source and the destination, the network operation we previously described allows to achieve all rates strictly smaller than this min-cut.
- (5) Using the union bound we can show that the same result holds simultaneously for all receivers.

We now make several important observations. First, note that the network operation *does not use the knowledge* of the min-cut value, at any other part but the source that produces packets at

the information rate. That is, each intermediate node performs the same operation, no matter what the min-cut value is, or where the node is located in the network. This is not the case with routing, where an intermediate node receiving linear independent information streams needs, for example, to know which information stream to forward where and at which rate. This property of network coding proves to be very useful in dynamically changing networks, as we will discuss in the second part of the review. Second, the packet length  $n$  can be thought of as the “alphabet size” required to achieve a certain probability of error when randomly selecting the coding operations. Third, although the proof assumed equal alphabet sizes, the same arguments go through for arbitrary alphabet sizes. Finally, note that the same proof would apply if, instead of unit capacity edges, we had arbitrary capacities, and more generally, if each edge were a Discrete Memoryless Channel (DMC), using the information theoretical min-cut.<sup>1</sup>

### 3.5 Linear-Programming Framework

There is a very rich history in using Linear Programming (LP) for studying flows through networks. Network coding has also been studied within this framework, which turned out to be the most natural for addressing networks with costs. Another notable achievement of this approach is the characterization of throughput benefits of network coding.

We start with some standard formulations for the max-flow problem and for multicast without network coding, and then present the LP formulations for multicast with network coding. In this section, we always assume a capacitated directed graph  $G = (V, E)$ , where edge  $e = (v, u)$  that connects vertex  $v \in V$  to vertex  $u \in V$  has an associated capacity  $c_{vu} \geq 0$ .

#### 3.5.1 Unicast and Multicast Flows

Consider the problem of maximizing the flow from a source  $S \in V$  to a single destination  $R \in V$ . Let  $f_{vu} \in \mathcal{R}^+$  denote the flow through the

<sup>1</sup>As defined for example in [17], Chapter 14.

edge  $e = (v, u) \in E$ . We want to maximize the flow that the source sends to the receiver, subject to two conditions: *capacity constraints* and *flow conservation*. The capacity constraints require that the value of the flow through each edge does not exceed the capacity of the edge. Flow conservation ensures that at every node  $u$  of the graph, apart from the source and the destination, the total incoming flow equals the total outgoing flow. In fact, by adding a directed edge of infinite capacity that connects the destination to the source, flow conservation can also be applied to the source and destination nodes. The associated LP can be stated as follows:

*Max-Flow LP:*  
 maximize  $f_{RS}$   
 subject to

$$\sum_{(v,u) \in E} f_{vu} = \sum_{(u,w) \in E} f_{uw}, \quad \forall u \in V \quad (\text{flow conservation})$$

$$f_{vu} \leq c_{vu}, \quad \forall (v, u) \in E \quad (\text{capacity constraints})$$

$$f_{vu} \geq 0, \quad \forall (v, u) \in E$$

Consider now the problem of maximizing the multicast rate from the (source) vertex  $S \in V$  to a set  $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$  of  $N$  terminals. If we do not use network coding, then this problem is equivalent to the problem of *Packing Steiner Trees*. A Steiner tree is a tree rooted at the source vertex that spans (reaches) all receivers. A partial Steiner tree may not reach all receivers. Figure 3.5 depict a Steiner tree rooted at  $S_2$  and a partial Steiner tree rooted at  $S_1$ .

With each tree  $t$ , we associate a random variable  $y_t$  that expresses the information rate conveyed through the tree  $t$  to the receivers. Let  $\tau$  be the set of all Steiner trees in  $\{G, S, \mathcal{R}\}$ , and  $n_t$  the number of terminals in  $t$ . The maximum fractional<sup>2</sup> packing of Steiner trees can be calculated by solving the following linear program.

<sup>2</sup>Fractional means that we do not restrict  $y_t$  to integer values.

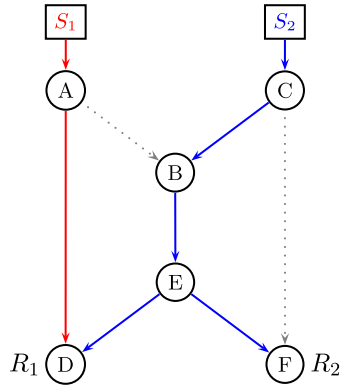


Fig. 3.5 The butterfly network with a Steiner tree rooted at  $S_2$  and a partial Steiner tree rooted at  $S_1$ .

*Packing Steiner Trees LP:*

$$\text{maximize } \sum_{t \in \tau} y_t$$

subject to

$$\sum_{t \in \tau: e \in t} y_t \leq c_e, \quad \forall e \in E \quad (\text{capacity constraints})$$

$$y_t \geq 0, \quad \forall t \in \tau$$

This is a hard problem to solve, as the set of all possible Steiner trees can be exponentially large in the number of vertices of the graph.

### 3.5.2 Multicast with Network Coding with and without Cost

If we use network coding, the problem of maximizing the throughput when multicasting has a polynomial time solution. The new idea we can use is the following. With each receiver  $R_i$ , we associate its own flow  $f^i$ . Each individual flow  $f^i$  obeys flow conservation equations. However, the individual flows are allowed to “overlap,” for example using linear combination, resulting in a *conceptual flow*  $f$ , which simply summarizes what information rate actually flows through each edge.

It is now this flow that needs to satisfy the capacity constraints. Note that the conceptual flow  $f$  *does not* need to satisfy flow conservation. Again we add virtual edges  $(R_i, S)$  of infinite capacity that connect each receiver  $R_i$  to the source vertex.

*Network Coding LP:*

maximize  $\chi$

subject to

$$\begin{aligned}
 & f_{R_i S}^i \geq \chi, \quad \forall i \\
 & \sum_{(v,u) \in E} f_{vu}^i = \sum_{(u,w) \in E} f_{uw}^i, \quad \forall u \in V, \quad \forall i \quad (\text{flow conservation}) \\
 & f_{vu}^i \leq f_{vu}, \quad \forall (v,u) \in E \quad (\text{conceptual flow constraints}) \\
 & f_{vu} \leq c_{vu}, \quad \forall (v,u) \in E \quad (\text{capacity constraints}) \\
 & f_{vu}^i \geq 0 \text{ and } f_{vu} \geq 0, \quad \forall (v,u) \in E, \quad \forall i
 \end{aligned}$$

The first constraint ensures that each receiver has min-cut at least  $\chi$ . Once we solve this program and find the optimal  $\chi$ , we can apply the main theorem in network coding to prove the existence of a network code that can actually deliver this rate to the receivers, no matter how the different flows  $f_i$  overlap. The last missing piece is to show that we can design these network codes in polynomial time, and this will be the subject of Chapter 5. Note that, over directed graphs, instead of solving the LP, we could simply identify the min-cut from the source to each receiver, and use the minimum such value. The merit of this program is that it can easily be extended to apply over undirected graphs as well, and that it introduces an elegant way of thinking about network coded flows through conceptual flows that can be applied to problems with similar flavor, such as the one we describe next.

In many information flow instances, instead of maximizing the rate, we may be interested in minimizing a cost that is a function of the information rate and may be different for each network edge. For example, the cost of power consumption in a wireless environment can be modeled in this manner. Consider an instance  $\{G, S, \mathcal{R}\}$  where the min-cut to each receiver is larger or equal to  $h$ , and assume

that the cost of using edge  $(v,u)$  is proportional to the flow  $f_{vu}$  with a weight  $w_{vu} \geq 0$  (i.e., the cost equals  $w_{vu}f_{vu}$ ). The problem of minimizing the overall cost in the network is also computationally hard if network coding is not allowed. On the other hand, by using network coding, we can solve it in polynomial time by slightly modifying the previous LP, where we associate zero cost with the virtual edges  $(R_i, S)$ :

*Network Coding with Cost LP:*

$$\begin{aligned} & \text{minimize} && \sum_{(x,y) \in E} w_{xy} f_{xy} \\ & \text{subject to} && \\ & && f_{R_i S}^i \geq h, \quad \forall i \\ & && \sum_{(v,u) \in E} f_{vu}^i = \sum_{(u,w) \in E} f_{uw}^i, \quad \forall u \in V, \quad \forall i \quad (\text{flow conservation}) \\ & && f_{vu}^i \leq f_{vu}, \quad \forall (v,u) \in E \quad (\text{conceptual flow constraints}) \\ & && f_{vu} \leq c_{vu}, \quad \forall (v,u) \in E \quad (\text{capacity constraints}) \\ & && f_{vu}^i \geq 0 \text{ and } f_{vu} \geq 0, \quad \forall (v,u) \in E, \quad \forall i \end{aligned}$$

In general, although network coding allows us to send information to each receiver at a rate equal to its min-cut, i.e., the same rate the receiver would achieve by using all the network resources exclusively to support its information transmission, if there is a cost associated with using the edges of the graph, network coding does not necessarily allow us to achieve the optimal cost that the exclusive use of the network would offer. We will see a specific such case in Example 6.2.

## 3.6 Types of Routing and Coding

We here describe several types of routing and coding we have or will encounter in this review.

### 3.6.1 Routing

Routing refers to the case where flows are sent uncoded from the source to the receivers, and intermediate nodes simply forward their incoming

information flows. We distinguish the following types of routing:

- (1) *Integral routing*, which requires that through each unit capacity edge we route at most one unit rate source.
- (2) *Fractional routing*, which allows multiple fractional rates from different sources to share the same edge as long as they add up to at most the capacity of the edge.
- (3) *Vector routing*, which can be either integral or fractional but is allowed to change at the beginning of each time slot.

Fractional (integral) routing is equivalent to the problem of packing fractional (integral) Steiner trees. Vector routing amounts to packing Steiner trees not only over space, but also over time.

### 3.6.2 Network Coding

Network coding refers to the case where the intermediate nodes in the network are allowed to perform coding operations. We can think of each source as producing a length  $L$  binary packet, and assume we can send through each edge one bit per time slot. There are several possible approaches to network coding.

- (1) In *linear network coding*, sets of  $m$  consecutive bits are treated as a symbol of  $\mathbb{F}_q$  with  $q = 2^m$ . Coding amounts to linear operations over the field  $\mathbb{F}_q$ . The same operations are applied symbol-wise to each of the  $L/m$  symbols of the packet. Thus, each incoming packet in a coding point is multiplied with a symbol in  $\mathbb{F}_q$  and added to other incoming packets.
- (2) In *vector linear network coding*, we treat each packet as a vector of length  $L/m$  with symbols in  $\mathbb{F}_{2^m}$  for some  $m$ . Encoding amounts to linear transformations in the vector space  $\mathbb{F}_{2^m}^{L/m}$ : each incoming packet (that is, the corresponding vector) is multiplied by an  $\frac{L}{m} \times \frac{L}{m}$  matrix over  $\mathbb{F}_{2^m}$ , and then added to other packets.
- (3) In *nonlinear network coding*, intermediate nodes are allowed to combine packets using any operation (e.g., bitwise AND).

For traffic configurations where the main theorem in network coding does not apply, there exist networks that are solvable using vector linear network coding, but not solvable using linear network coding. Moreover, there exist networks which are solvable only using nonlinear operations and are not solvable using linear operations. Finally, the alphabet size of the solution can be significantly reduced as we move from linear, to vector linear, and to nonlinear solutions.

### 3.6.3 Coding at the Source

This is a hybrid scheme between routing and network coding: source nodes are allowed to perform coding operations and send out encoded information streams. Intermediate nodes in the network however, are only allowed to forward their incoming information streams.

In networks where there are no losses, to maximize the multicast rate, we can combine routing (or vector routing) with erasure correcting coding. For example:

- Pack through the network and over  $n$ -time slots partial Steiner trees, i.e., trees that are rooted at the source nodes that span a subset of the receivers such that each receiver appears in at least  $f$  trees.
- Encode the information using an erasure correcting code at the source and send a different coded symbol through each partial tree.

The idea behind this scheme is to assume that a receiver experiences an “erasure” if it is not spanned by a partial Steiner tree. Thus each receiver essentially observes the source information through an erasure channel.

We now formulate the problem of creating an appropriate routing schedule that supports such a coding scheme as a linear program. Consider an instance  $\{G, S, \mathcal{R}\}$ . Let  $\tau$  denote the set of partial Steiner trees in  $G$  rooted at  $S$  with terminal set  $\mathcal{R}$ . We will be using the number of time slots,  $n$ , as a parameter. In each time slot we seek a feasible fractional packing of partial Steiner trees. The goal is to maximize the total number of trees that each receiver occurs in, across

the time slots. We express this as a linear program as follows. For a tree  $t \in \tau$  and a time slot  $k$ , we have a non-negative variable  $y(t, k)$  that indicates the amount of  $t$  that is packed in time slot  $k$ .

*Coding at the Source LP:*

maximize  $f$

subject to

$$\sum_k \sum_{t \in \tau: R_i \in t} y(t, k) \geq f, \quad \forall R_i$$

$$\sum_{t \in \tau: e \in t} y(t, k) \leq c_e, \quad \forall e \in E, \quad 1 \leq k \leq n$$

$$y(t, k) \geq 0, \quad \forall t \in \tau, \quad 1 \leq k \leq n$$

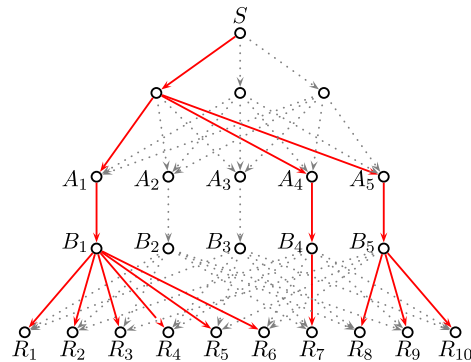
Given a solution  $y^*$  to this linear program, let  $m = \sum_k \sum_{t \in \tau} y^*(t, k)$ . We can use an erasure code that employs  $m$  coded symbols to convey the same  $f$  information symbols to all receivers, i.e., to achieve rate  $T_i = f/m$ . (To be precise, we need  $f/m$  to be a rational and not a real number.) For integer edge capacities  $c_e$ , there is an optimum solution with rational coordinates.

---

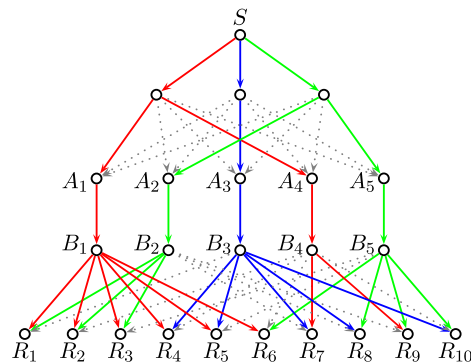
**Example 3.3.** The network in Figure 3.6 has  $N = 10$  receivers, and each receiver has min-cut three from the source. If we are restricted to integral routing over a single time-slot, we can pack one Steiner tree, which would lead to  $T_i = 1$ . Alternatively, we can pack three partial Steiner trees, where each receiver is taking part in two distinct trees. Using an erasure code of rate  $2/3$  at the source, we can then achieve  $T_i = 2$ .

---

This scheme is very related to the rate-less codes approach over lossy networks, such as LT codes and Raptor codes. In lossy networks, packet erasures can be thought of as randomly pruning trees. At each time slot, the erasures that occur correspond to a particular packing of partial Steiner trees. Rate-less codes allow the receivers to decode as soon as they collect a sufficient number of coded symbols.



Case 1: Packing steiner trees.



Case 2: Packing partial steiner trees.

Fig. 3.6 A network with unit capacity edges where, if we are restricted to integral routing over one time-slot, we can pack (Case 1) one unit rate Steiner tree, or (Case 2) three unit rate partial Steiner trees.

## Notes

The algebraic framework for network coding was developed by Koetter and Médard in [32], who translated the network code design to an algebraic problem which depends on the structure of the underlying graph. Lemma 3.1 is proved by Ho *et al.* in [26] and by Harvey in [25]. The presented combinatorial framework follows the approach of Fragouli and Soljanin in [24]. The information theoretic framework comes from the original paper by Ahlswede *et al.* [2]. Introduction to LP can be found in numerous books, see for example [10, 44]. The network coding LP

that achieved flow maximization was proposed by Li *et al.* in [37], while the cost minimization LPs and decentralized distributed algorithms for solving them were proposed by Lun *et al.* in [38]. Vector routing was examined by Cannons *et al.* in [11]. Linear vs. nonlinear solvability was discussed by Riis in [42] and by Dougherty *et al.* in [19]. Coding at the source using partial tree packing was proposed by Chekuri *et al.* in [15].

# 4

---

## Throughput Benefits of Network Coding

---

The multicast examples considered in the previous chapters demonstrated that network coding can offer throughput benefits when compared to routing; we will here look into how large such benefits can be. We consider both directed and undirected networks, under two types of routing: integral routing, which requires that through each unit capacity edge we route at most one unit rate source, and fractional routing, which allows multiple fractional rates from different sources that add up to at most one on any given edge. We consider two throughput measures: symmetric throughput, which refers to the common information rate guaranteed to all receivers, and average throughput, which refers to the average of the rates that the individual receivers experience.

We will see how throughput benefits of network coding can be related to the integrality gap of a standard LP formulation for the Steiner tree problem. This approach applies to both directed and undirected graphs. For the special case of undirected networks, we will outline a simple proof that coding can at most double the throughput. For directed networks, we will give examples where coding offers

large throughput benefits as well as examples where these benefits are limited.

#### 4.1 Throughput Measures

We denote by  $T_c$  the maximum rate that the receivers experience when network coding is used, and refer to it as *coding throughput*. For the routing throughput, we use the following notation:

- $T_i^j$  and  $T_f^j$  denote the rate that receiver  $R_j$  experiences with fractional and integral routing, respectively, under some routing strategy.
- $T_i$  and  $T_f$  denote the maximum integral and fractional rate we can route to all receivers, where the maximization is over all possible routing strategies. We refer to this throughput as *symmetric* or *common*.
- $T_i^{av} = \frac{1}{N} \max \sum_{j=1}^N T_i^j$  and  $T_f^{av} = \frac{1}{N} \max \sum_{j=1}^N T_f^j$  denote the maximum integral and fractional *average* throughput.

The benefits of network coding in the case of the common throughput measure are described by

$$\frac{T_i}{T_c} \quad \text{and} \quad \frac{T_f}{T_c}$$

and the benefits of network coding in the case of the average throughput measure are described by

$$\frac{T_i^{av}}{T_c} \quad \text{and} \quad \frac{T_f^{av}}{T_c}.$$

For a multicast configuration with  $h$  sources and  $N$  receivers, it holds that

$$T_c = h,$$

from the main network multicast theorem. Also, because there exists a tree spanning the source and the receiver nodes, the uncoded throughput is at least 1. We, therefore, have

$$1 \leq T_i^{av} \leq T_f^{av} \leq h,$$

and thus

$$\frac{1}{h} \leq \frac{T_i^{av}}{T_c} \leq \frac{T_f^{av}}{T_c} \leq 1. \quad (4.1)$$

The upper bound in (4.1) is achievable by the configurations in which network coding is not necessary for multicast at rate  $h$ .

## 4.2 Linear Programming Approach

We here use the Linear Programming (LP) framework to show that the throughput benefits network coding offers equal the integrality gap of a standard formulation for the Steiner tree problem. We prove this result for the symmetric throughput in Theorem 4.1 and give without proof the respective result for the average throughput in Theorem 4.2. We focus our discussion on directed graphs, as we will treat undirected graphs separately later on. However, the described approach applies to both directed and undirected graphs.

### 4.2.1 Symmetric Throughput Coding Benefits

Consider an instance  $\{G, S, \mathcal{R}\}$  where  $G = (V, E)$  is a directed graph,  $S \in V$  is a root (source) vertex, and  $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$  a set of  $N$  terminals (receivers). For the instance  $\{G, S, \mathcal{R}\}$ , a Steiner tree is a subset of  $G$  that connects  $S$  with the set of receivers  $\mathcal{R}$ .

With every edge  $e$  of the graph, we can in general associate two parameters: a capacity  $c_e \geq 0$  and a cost (weight)  $w_e \geq 0$ . Let  $c = [c_e]$  and  $w = [w_e]$ ,  $e \in E$ , denote vectors that collect the set of edge capacities and edge weights, respectively. Either the edge weights or the edge capacities or both may be relevant in a particular problem. We consider essentially two problems: the Steiner tree packing problem and the minimum Steiner tree problem.

In the *Steiner tree packing problem*, we are given  $\{G, S, \mathcal{R}\}$  and a set of non-negative edge capacities  $c$ . The maximum throughput achievable with routing for this instance can be calculated using the following LP

(see Chapter 3):

<p style="text-align: center;"><i>Packing Steiner Trees – Primal:</i></p> <p>maximize <math>\sum_{t \in \tau} y_t</math></p> <p>subject to</p> $\sum_{t \in \tau: e \in t} y_t \leq c_e, \quad \forall e \in E$ $y_t \geq 0, \quad \forall t \in \tau$	(4.2)
--	-------

Recall that the variable  $y_t$  expresses how much fractional rate we route from the source to the receivers through the Steiner tree  $t$ , for all possible Steiner trees  $t \in \tau$  for our instance. Let  $T_f(G, S, \mathcal{R}, c)$  denote the optimal solution of this problem, and  $T_c(G, S, \mathcal{R}, c)$  the rate network coding achieves. For further analysis, it is useful to look at the dual problem of the above LP:

<p style="text-align: center;"><i>Packing Steiner Trees – Dual:</i></p> <p>minimize <math>\sum_{e \in E} c_e z_e</math></p> <p>subject to</p> $\sum_{e \in t} z_e \geq 1, \quad \forall t \in \tau$ $z_e \geq 0, \quad \forall e \in E$	(4.3)
---	-------

In the dual program, we can think of the variable  $z_e$  as expressing a cost of edge  $e$ . We are asking what cost  $z_e$  should be associated with edge  $e$  (for all  $e \in E$ ) so that *each* Steiner tree has cost at least one and the total cost  $\sum_{e \in E} c_e z_e$  is minimized.

In the *minimum Steiner tree problem*, we are given  $\{G, S, \mathcal{R}\}$  and a set of non-negative edge weights  $w$ . We are asked to find the minimum weight tree that connects the source to all the terminals. Here edge capacities are not relevant: the Steiner tree either uses or

does not use an edge. To state this problem, we need the notion of a separating set of vertices. We call a set of vertices  $\mathcal{D} \subset V$  *separating* if  $\mathcal{D}$  contains the source vertex  $S$  and  $V \setminus \mathcal{D}$  contains at least one of the terminals in  $\mathcal{R}$ . Let  $\delta(\mathcal{D})$  denote the set of edges from  $\mathcal{D}$  to  $V \setminus \mathcal{D}$ , that is,  $\delta(\mathcal{D}) = \{(u, v) \in E : u \in \mathcal{D}, v \notin \mathcal{D}\}$ . We consider the following integer programming (IP) formulation for the minimum Steiner tree problem:

<p><i>Steiner Tree Problem:</i></p> <p>minimize <math>\sum_{e \in E} w_e x_e</math></p> <p>subject to</p> $\sum_{e \in \delta(\mathcal{D})} x_e \geq 1, \quad \forall \mathcal{D}: \mathcal{D} \text{ is separating}$ $x_e \in \{0, 1\}, \quad \forall e \in E$	(4.4)
---	-------

where there is a binary variable  $x_e$  for each edge  $e \in E$  to indicate whether the edge is contained in the tree. We denote by  $\text{OPT}(G, w, S, \mathcal{R})$  the value of the optimum solution for the given instance.

A linear relaxation of the above IP is obtained by replacing the constraint  $x_e \in \{0, 1\}$  by  $0 \leq x_e \leq 1$  for all  $e \in E$ . We can further simplify this constraint to  $x_e \geq 0$ ,  $e \in E$  (noticing that if a solution is feasible with  $x_e \geq 1$ , then it remains feasible by setting  $x_e = 1$ ), and obtain the following formulation:

<p><i>Linear Relaxation for Steiner Tree Problem:</i></p> <p>minimize <math>\sum_{e \in E} w_e x_e</math></p> <p>subject to</p> $\sum_{e \in \delta(\mathcal{D})} x_e \geq 1, \quad \forall \mathcal{D}: \mathcal{D} \text{ is separating}$ $x_e \geq 0, \quad \forall e \in E$	(4.5)
---	-------

Let  $\text{LP}(G, w, S, \mathcal{R})$  denote the optimum value of this linear program for the given instance. The value  $\text{LP}(G, w, S, \mathcal{R})$  lower bounds the cost of the integer program solution  $\text{OPT}(G, w, S, \mathcal{R})$ . The *integrality gap* of the relaxation on  $G$  is defined as

$$\alpha(G, S, \mathcal{R}) = \max_{w \geq 0} \frac{\text{OPT}(G, w, S, \mathcal{R})}{\text{LP}(G, w, S, \mathcal{R})},$$

where the maximization is over all possible edge weights. Note that  $\alpha(G, S, \mathcal{R})$  is invariant to scaling of the optimum achieving weights.

The theorem tells us that, for a given configuration  $\{G, S, \mathcal{R}\}$ , the maximum throughput benefits we may hope to get with network coding equals the largest integrality gap of the Steiner tree problem possible on the same graph. This result refers to fractional routing; if we restrict our problem to integral routing on the graph, we may get even larger throughput benefits from coding.

---

**Theorem 4.1.** Given an instance  $\{G, S, \mathcal{R}\}$

$$\max_w \frac{\text{OPT}(G, w, S, \mathcal{R})}{\text{LP}(G, w, S, \mathcal{R})} = \max_c \frac{T_c(G, S, \mathcal{R}, c)}{T_f(G, S, \mathcal{R}, c)}.$$


---

*Proof.* We first show that there exist weights  $w$  so that

$$\max_c \frac{T_c(G, S, \mathcal{R}, c)}{T_f(G, S, \mathcal{R}, c)} \leq \frac{\text{OPT}(G, w, S, \mathcal{R})}{\text{LP}(G, w, S, \mathcal{R})}. \quad (4.6)$$

- (1) For the instance  $\{G, S, \mathcal{R}\}$ , find  $c = c^*$  such that the left-hand side ratio is maximized, i.e.,

$$\frac{T_c(G, S, \mathcal{R}, c^*)}{T_f(G, S, \mathcal{R}, c^*)} = \max_c \frac{T_c(G, S, \mathcal{R}, c)}{T_f(G, S, \mathcal{R}, c)}.$$

Assume  $c^*$  is normalized so that the minimum min-cut from the source to the receivers (and as a result the network coding rate) equals one, that is,

$$\frac{T_c(G, S, \mathcal{R}, c^*)}{T_f(G, S, \mathcal{R}, c^*)} = \frac{1}{T_f(G, S, \mathcal{R}, c^*)}. \quad (4.7)$$

Note that normalization does not affect fractional routing.

- (2) Now solve the dual program in (4.3). From strong duality,<sup>1</sup> we get that

$$T_f(G, S, \mathcal{R}, c^*) = \sum y_t^* = \sum c_e z_e^*. \quad (4.8)$$

Then the dual program equivalently says that there exist costs  $z_e^*$  such that each Steiner tree has cost at least one. From the slackness conditions,<sup>2</sup> each tree  $t$  used for fractional routing (with  $y_t > 0$ ), gets cost of exactly one  $\sum_{e \in t} z_e^* = 1$ .

- (3) We now use our second set of optimization problems. Consider the same instance  $\{G, S, \mathcal{R}\}$ , and associate weights  $w_e = z_e^*$  with every edge of the graph, and solve on this graph the Steiner tree problem in (4.4). Since for this set of weights, each Steiner tree in our graph has weight at least one, and (as previously discussed) the slackness conditions imply that there exist trees of weight one, the minimum value the Steiner tree IP can achieve equals one:

$$\text{OPT}(G, w = z^*, S, \mathcal{R}) = 1. \quad (4.9)$$

- (4) Consider the LP relaxation of this problem stated in (4.5). Note that we can get a feasible solution by choosing  $x = c^*$  (the capacities in the original graph). Indeed, for these values the min-cut to each receiver equals at least one. Moreover,  $\sum z_e^* c_e = T_f(G, S, \mathcal{R}, c^*)$ . Therefore, the LP minimization will lead to a solution

$$\text{LP}(G, w = z^*, S, \mathcal{R}) \leq T_f(G, S, \mathcal{R}, c^*). \quad (4.10)$$

Putting together (4.7), (4.9), and (4.10), we get (4.6).

We next show that there exist capacities  $c$  so that

$$\frac{T_c(G, S, \mathcal{R}, c)}{T_f(G, S, \mathcal{R}, c)} \geq \max_w \frac{\text{OPT}(G, w, S, \mathcal{R})}{\text{LP}(G, w, S, \mathcal{R})}. \quad (4.11)$$

<sup>1</sup>Strong duality holds since there exists a feasible solution in the primal.

<sup>2</sup>See [10, 44] for an overview of LP.

- (1) We start from the Steiner tree problem in (4.4) and (4.5).

Let  $w^*$  be a weight vector such that

$$\alpha(G, S, \mathcal{R}) = \frac{\text{OPT}(G, w^*, S, \mathcal{R})}{\text{LP}(G, w^*, S, \mathcal{R})} = \max_w \frac{\text{OPT}(G, w, S, \mathcal{R})}{\text{LP}(G, w, S, \mathcal{R})}.$$

Let  $x^*$  be an optimum solution for the LP on the instance  $(G, w^*, S, \mathcal{R})$ . Hence

$$\text{LP}(G, w^*, S, \mathcal{R}) = \sum_e w_e^* x_e^*.$$

Note that vectors with positive components, and thus the vectors  $x = \{x_e, e \in E\}$  satisfying the constraints of (4.5), can be interpreted as a set of capacities for the edges of  $G$ . In particular, we can then think of the optimum solution  $x^*$  as associating a capacity  $c_e = x_e^*$  with each edge  $e$  so that the min-cut to each receiver is greater or equal to one, and the cost  $\sum_e w_e^* x_e^*$  is minimized.

- (2) Consider now the instance with these capacities, namely,
- $\{G, c = x^*, S, \mathcal{R}\}$
- and examine the coding throughput benefits we can get. Since the min-cut to each receiver is at least one, with network coding we can achieve throughput

$$T_c(G, c = x^*, S, \mathcal{R}) \geq 1. \quad (4.12)$$

With fractional routing we can achieve

$$T_f(G, c = x^*, S, \mathcal{R}) = \sum y_t^* \quad (4.13)$$

for some  $y_t^*$ .

- (3) We now need to bound the cost
- $\text{OPT}(G, w^*, S, \mathcal{R})$
- that the solution of the IP Steiner tree problem will give us. We will use the following averaging argument. We will show that the average cost per Steiner tree will be less or equal to

$$\frac{\sum w_e^* x_e^*}{T_f(G, c = x^*, S, \mathcal{R})}.$$

Therefore, there exists a Steiner tree with the weight smaller than the average, which upper bounds the solution of (4.5):

$$\text{OPT}(G, w^*, S, \mathcal{R}) \leq \frac{\sum w_e^* x_e^*}{T_f(G, x^*, S, \mathcal{R})} = \frac{\text{LP}(G, w^*, S, \mathcal{R})}{T_f(G, x^*, S, \mathcal{R})}$$

But from (4.12), we have

$$\frac{\text{LP}(G, w^*, S, \mathcal{R})}{T_f(G, c = x^*, S, \mathcal{R})} \leq \text{LP}(G, w^*, S, \mathcal{R}) \frac{T_c(G, c = x^*, S, \mathcal{R})}{T_f(G, c = x^*, S, \mathcal{R})}$$

and we are done.

- (4) Let  $w_t = \sum_{e \in t} w_e^*$  denote the weight of a tree  $t$ , and consider  $\sum_{t \in \tau} w_t y_t^*$  (the total weight of the packing  $y^*$ ). We have

$$\sum_{t \in \tau} w_t y_t^* \geq \min_{t \in \tau} \{w_t\} \sum_{t \in \tau} y_t^*.$$

Thus there exists a tree  $t_1 \in \arg \min_{t \in \tau} \{w_t\}$  of weight  $w_{t_1}$  such that

$$w_{t_1} \leq \frac{\sum_{t \in \tau} w_t y_t^*}{\sum_{t \in \tau} y_t^*}. \quad (4.14)$$

- (5) Finally, we need to show that  $\sum_{t \in \tau} w_t y_t^* \leq \sum_{e \in E} w_e^* x_e^*$ . Indeed, by changing the order of summation, we get

$$\sum_{t \in \tau} w_t y_t^* = \sum_{t \in \tau} y_t^* \sum_{e \in t} w_e^* = \sum_{e \in E} w_e^* \sum_{t: e \in t} y_t^*.$$

By the feasibility of  $y^*$  for the capacity vector  $c = x^*$ , the quantity  $\sum_{t: e \in t} y_t^*$  is at most  $x_e^*$ . Hence we have that

$$\sum_{t \in \tau} w_t y_t^* \leq \sum_{e \in E} w_e^* x_e^*. \quad (4.15)$$

Putting together (4.6) and (4.11) proves the theorem.  $\square$

### 4.2.2 Average Throughput Coding Benefits

We now consider the coding advantage for average throughput over a multicast configuration  $\{G, S, \mathcal{R}\}$  and a set of non-negative capacities  $c$  on the edges of  $G$ . We will assume for technical reasons that the min-cut from  $S$  to each of the terminals is the same. This can be easily arranged by adding dummy terminals. That is, if the min-cut to a receiver  $R_i$  is larger than required, we connect the receiver node to a new dummy terminal through an edge of capacity equal to the min-cut. Then the network coding throughput is equal to the common min-cut value.

The maximum achievable average throughput with routing is given by the maximum fractional packing of *partial* Steiner trees. A partial Steiner tree  $t$  stems from the source  $S$  and spans all or only a subset of the terminals. With each tree  $t$ , we associate a variable  $y_t$  denoting a fractional flow through the tree. Let  $\tau$  be the set of all partial Steiner trees in  $\{G, S, \mathcal{R}\}$ , and  $n_t$  the number of terminals in  $t$ . Then the maximum fractional packing of partial Steiner trees is given by the following linear program:

*Packing Partial Steiner Trees:*

$$\text{maximize } \sum_{t \in \tau} \frac{n_t}{N} y_t$$

subject to

$$\sum_{t \in \tau: e \in t} y_t \leq c_e, \quad \forall e \in E$$

$$y_t \geq 0, \quad \forall t \in \tau$$

Let  $T_f^{av}(G, S, \mathcal{R}, c)$  denote the value of the above linear program on a given instance. The coding advantage for average throughput on  $G$  is given by the ratio

$$\beta(G, S, \mathcal{R}) = \max_c \frac{T_c(G, c, S, \mathcal{R})}{T_f^{av}(G, c, S, \mathcal{R})}.$$

Note that  $\beta(G)$  is invariant to scaling capacities. It is easy to see that  $\beta(G, S, \mathcal{R}) \geq 1$ , since we assumed that the min-cut to each receiver is the same, and thus network coding achieves the maximum possible sum rate. It is also straightforward that  $\beta(G, S, \mathcal{R}) \leq \alpha(G, S, \mathcal{R})$ , since for any given configuration  $\{G, c, S, \mathcal{R}\}$ , the average throughput is at least as large as the common throughput we can guarantee to all receivers, namely,  $T_f^{av} \geq T_f$ .

Let  $\beta(G, S, \mathcal{R}^*)$  denote the maximum average throughput benefits we can get on graph  $G$  when multicasting from source  $S$  to *any possible subset* of the receivers  $\mathcal{R}' \subseteq \mathcal{R}$ :

$$\beta(G, S, \mathcal{R}^*) = \max_{\mathcal{R}' \subseteq \mathcal{R}} \beta(G, S, \mathcal{R}').$$

---

**Theorem 4.2.** For a configuration  $\{G, S, \mathcal{R}\}$  where the number of receivers is  $|\mathcal{R}| = N$  and the min-cut to each receiver is the same, we have

$$\beta(G, S, \mathcal{R}^*) \geq \max \left\{ 1, \frac{1}{H_N} \alpha(G, S, \mathcal{R}) \right\},$$

where  $H_N$  is the  $N$ th harmonic number  $H_N = \sum_{j=1}^N 1/j$ .

---

Note that the maximum value of  $T_f$  and  $T_f^{av}$  is not necessarily achieved for the same capacity vector  $c$ , or for the same number of receivers  $N$ . What this theorem tells us is that, for a given  $\{G, S, \mathcal{R}\}$ , with  $|\mathcal{R}| = N$ , the maximum common rate we can guarantee to all receivers will be at most  $H_N$  times smaller than the maximum average rate we can send from  $S$  to any subset of the receivers  $\mathcal{R}$ . The theorem quantitatively bounds the advantage in going from the stricter measure  $\alpha(G, S, \mathcal{R})$  to the weaker measure  $\beta(G, S, \mathcal{R}^*)$ . Furthermore, it is often the case that for particular instances  $(G, S, \mathcal{R})$ , either  $\alpha(G, S, \mathcal{R})$  or  $\beta(G, S, \mathcal{R}^*)$  is easier to analyze and the theorem can be useful to get an estimate of the other quantity.

We now comment on the tightness of the bounds in the theorem. There are instances in which  $\beta(G, S, \mathcal{R}^*) = 1$ , take for example the case when  $G$  is a tree rooted at  $S$ . On the other hand there are instances in which  $\beta(G, S, \mathcal{R}^*) = O(1/\ln N)\alpha(G, S, \mathcal{R})$ . Examples include network graphs discussed in the next section. In general, the ratio  $\alpha(G, S, \mathcal{R})/\beta(G, S, \mathcal{R}^*)$  can take a value in the range  $[1, H_N]$ .

### 4.3 Configurations with Large Network Coding Benefits

The first example that demonstrated the benefits of network coding over routing for symmetric throughput was the combination network  $B(h, k)$  depicted in Figure 4.1.

---

#### Example 4.1. The Combination Network

The combination network  $B(h, k)$ , where  $k \geq h$ , has  $h$  sources,  $N = \binom{k}{h}$  receivers, and two layers of  $k$  intermediate nodes  $A$  and  $B$ . All sources

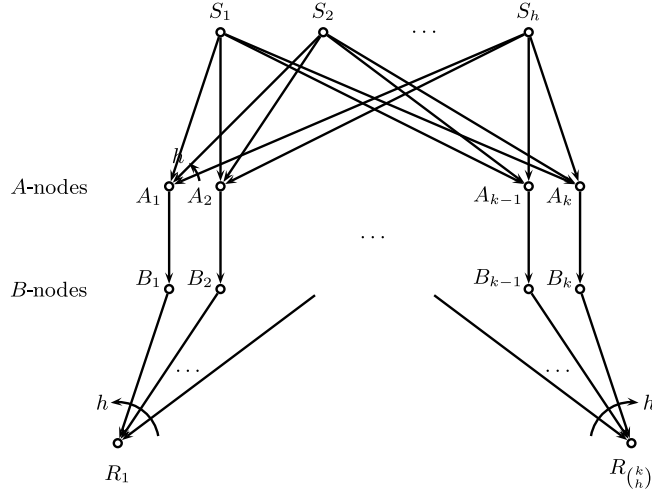


Fig. 4.1 The combination  $B(h, k)$  network has  $h$  sources and  $N = \binom{k}{h}$  receivers, each receiver observing a distinct subset of  $h$   $B$ -nodes. Edges have unit capacity and sources have unit rate.

are connected to  $k$   $A$  nodes. Each  $A$  node is connected to a corresponding  $B$  node. Each of the  $\binom{k}{h}$  receivers is connected to a different subset of size  $h$  of the  $B$  nodes, as Figure 4.1 depicts.  $B(h, k)$  has  $k$  coding points, that are the edges  $(A_i, B_i)$ ,  $1 \leq i \leq k$ , between the  $A$  and  $B$  nodes. A specific instance  $B(h = 3, k = 5)$  is depicted in Figure 3.6.

The benefits of network coding compared to integral routing for the network  $B(h, k)$  can be bounded as

$$\frac{T_i}{T_c} \leq \frac{1}{h}, \quad \text{for } k \geq h^2. \quad (4.16)$$

To see that, note that the min-cut condition is satisfied for every receiver, and thus  $T_c = h$ . Choose any integral routing scheme, that sends one source through every edges that is a coding point. From the pigeonhole principle, because there exist  $h$  sources and  $k > h^2$  points, there will exist at least one source, say  $S_1$ , that will be allocated to at least  $h$  coding points. Consider the receiver that observes these  $h$  coding points where  $S_1$  flows. These receiver will experience rate equal to one, and thus the bound (4.16) follows.

It is also straightforward to upper-bound the fractional throughput of  $B(k, h)$ . Note that each Steiner tree needs to use  $k - (h - 1)$  out of the  $k$   $A_i B_i$  edges, to reach all the receivers. Therefore, the fractional packing number is at most  $k/(k - h + 1)$ , and consequently

$$\frac{T_f}{T_c} \leq \frac{k}{h(k - h + 1)}. \quad (4.17)$$

On the other hand, as we will see the following section, the average throughput benefits for  $B(h, k)$  are upper bounded by a factor of two. In addition, we will see that by using a coding scheme at the source and vector routing, the symmetric throughput can approach the average, and thus the coding benefits are upper bounded by the same factor. It is, therefore, natural to ask if there exist configurations where network coding offers significant benefits with respect to the average throughput. We next describe a class of networks with  $N$  receivers for which coding can offer up to  $\mathcal{O}(\sqrt{N})$ -fold increase of the average throughput achievable by routing. We refer to this class as ZK networks, because they were introduced by Zosin and Khuller, who constructed them to demonstrate a possible size of the integrality gap for the directed Steiner tree problem.

---

**Example 4.2. The  $ZK(p, N)$  Network**

Let  $N$  and  $p$ ,  $p \leq N$ , be two integers and  $\mathcal{I} = \{1, 2, \dots, N\}$  be an index set. We define two more index sets:  $\mathcal{A}$  as the set of all  $(p - 1)$ -element subsets of  $\mathcal{I}$  and  $\mathcal{B}$  as the set of all  $p$ -element subsets of  $\mathcal{I}$ . A  $ZK(N, p)$  network, illustrated in Figure 4.2, is defined by two parameters  $N$  and  $p$  as follows: Source  $S$  transmits information rate  $\omega = h = \binom{N-1}{p-1}$  to  $N$  receiver nodes  $R_1 \cdots R_N$  through a network of three sets of nodes  $A$ ,  $B$ , and  $C$ .  $A$ -nodes are indexed by the elements of  $\mathcal{A}$ , and  $B$ - and  $C$ -nodes, by the elements of  $\mathcal{B}$ . An  $A$  node is connected to a  $B$  node if the index of  $A$  is a subset of the index of  $B$ . A  $B$  node is connected to a  $C$  node if and only if they correspond to the same element of  $\mathcal{B}$ . A receiver node is connected to the  $C$  nodes whose indices contain the index of the receiver. All edges in the graph have unit capacity. The out-degree of the source node is  $\binom{N}{p-1}$ , the out-degree of  $A$  nodes is  $N - (p - 1)$ , the in-degree of  $B$  nodes is  $p$ , the out-degree of  $C$  nodes is  $p$ , and the

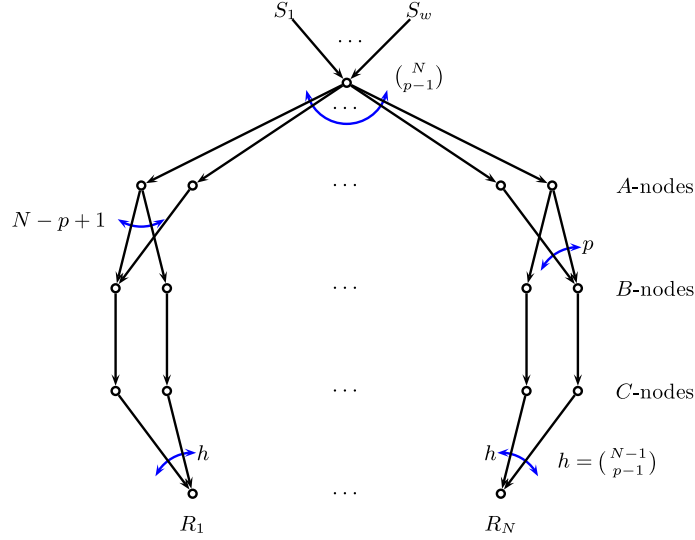


Fig. 4.2  $\text{ZK}(N, p)$  network.

in-degree of the receiver nodes is  $\binom{N-1}{p-1}$ . In fact, it is easy to show that there exist exactly  $\binom{N-1}{p-1}$  edge disjoint paths between the source and each receiver, and thus the min-cut to each receiver equals  $\binom{N-1}{p-1}$ .

The following theorem provides an upper bound to the average throughput benefits achievable by network coding in  $\text{ZK}(N, p)$  networks.

**Theorem 4.3.** In a  $\text{ZK}(N, p)$  network (Figure 4.2) with  $h = \binom{N-1}{p-1}$  sources and  $N$  receivers,

$$\frac{T_f^{av}}{T_c} \leq \frac{p-1}{N-p+1} + \frac{1}{p}. \quad (4.18)$$

*Proof.* If only routing is permitted, the information is transmitted from the source node to the receiver through a number of trees, each carrying a different information source. Let  $a_t$  be the number of  $A$ -nodes in tree  $t$ , and  $c_t$ , the number of  $B$ - and  $C$ -nodes. Note that  $c_t \geq a_t$ , and that the  $c_t$   $C$ -nodes are all descendants of the  $a_t$   $A$ -nodes. Therefore, we can

count the number of the receivers spanned by the tree as follows: Let  $n_t(A_j)$  be the number of  $C$ -nodes connected to  $A_j$ , the  $j$ th  $A$ -node in the tree. Note that

$$\sum_{j=1}^{a_t} n_t(A_j) = c_t.$$

The maximum number of receivers the tree can reach through  $A_j$  is  $n_t(A_j) + p - 1$ . Consequently, the maximum number of receivers the tree can reach is

$$\sum_{j=1}^{a_t} [n_t(A_j) + p - 1] = a_t(p - 1) + c_t.$$

To find an upper bound to the routing throughput, we need to find the number of receivers that can be reached by a set of disjoint trees. Note that for any set of disjoint trees, we have

$$\sum_t a_t \leq \binom{N}{p-1} \quad \text{and} \quad \sum_t c_t \leq \binom{N}{p}.$$

Thus,  $T_i$  can be upper-bounded as

$$\begin{aligned} T_i &\leq \frac{1}{N} \sum_t (a_t(p-1) + c_t) = \frac{1}{N}(p-1) \sum_t a_t + \sum_t c_t \\ &\leq (p-1) \binom{N}{p-1} + \binom{N}{p}. \end{aligned} \quad (4.19)$$

The network coding rate  $T_c$  is equal to  $\binom{N-1}{p-1}$ . Therefore,

$$\frac{T_i^{av}}{T_c} \leq \frac{p-1}{N-p+1} + \frac{1}{p}. \quad (4.20)$$

We can apply the exact same arguments to upper bound  $T_f^{av}$ , by allowing  $a_t$  and  $c_t$  to take fractional values, and interpreting these values as the fractional rate of the corresponding trees.  $\square$

For a fixed  $N$ , the right-hand side of (4.20) is minimized for

$$p = \frac{N+1}{\sqrt{N}+1} \cong \sqrt{N},$$

and for this value of  $p$ ,

$$\frac{T_f^{av}}{T_c} \leq 2 \frac{\sqrt{N}}{1+N} \lesssim \frac{2}{\sqrt{N}}. \quad (4.21)$$

#### 4.4 Configurations with Small Network Coding Benefits

For a number of large classes of networks, coding can at most double the *average* rate achievable by using very simple routing schemes.<sup>3</sup> We next describe several such classes of networks, which include several examples considered in the network coding literature up to date. The simplest examples of such networks are configurations with two sources.

---

**Example 4.3.** For a network with two sources the bounds in (4.1) give

$$\frac{1}{2} \leq \frac{T_i^{av}}{T_c} \leq 1$$

by setting  $h = 2$ . We can achieve the lower bound by simply finding a spanning tree. In general, a tighter bound also holds:

$$\frac{T_i^{av}}{T_c} \geq \frac{1}{2} + \frac{1}{2N},$$

and there are networks for which this bound holds with equality.

---

There are networks with two sources with even smaller coding throughput advantage.

---

**Example 4.4.** In any combination network with two unit rate sources (network coding throughput  $T_c = 2$ ) a simple routing scheme can achieve an average throughput of at least  $3T_c/4$  as follows: We route  $S_1$  through one half of the  $k$  coding points  $A_i B_i$  and  $S_2$  through the other half. Therefore, the average routing throughput, for even  $k$ , is given by

$$T_i^{av} = \frac{1}{\binom{k}{2}} \left\{ 2 \binom{\frac{k}{2}}{2} + 2 \left[ \binom{k}{2} - 2 \binom{\frac{k}{2}}{2} \right] \right\} = 2 \left( \frac{3}{4} + \frac{1}{4(k-1)} \right) > \frac{3}{4} T_c.$$

---

<sup>3</sup>Recall that computing an optimum routing is in general NP-hard.

The term  $2\binom{k}{2}$  is the sum rate for the receivers that only observe one of the two sources, while the term  $2[\binom{k}{2} - 2\frac{k}{2}]$  is the sum rate for the receivers that observe both sources. A similar bound holds for odd  $k$ .

---

The subtree decomposition can prove useful here in enabling us to estimate coding advantages for large classes of networks. For example, network coding at most doubles the throughput in all networks with  $h$  sources and  $N$  receivers whose information flow graphs are bipartite and each coding point has two parents. The same is true for configurations where all coding points have min-cut  $h$  from the sources.

---

**Example 4.5.** The combination network provides an interesting example where coding offers significant benefits for the symmetric throughput (as discussed in the previous section), while it cannot even double the average throughput achievable by routing. To see that, we consider a simple randomized routing scheme and compute the expected throughput observed by a single receiver. In this scheme, for each edge going out of the source node, one of the  $h$  information sources is chosen uniformly at random and routed through the edge. The probability that a receiver does not observe a particular source is thus equal to

$$\epsilon = \left(\frac{h-1}{h}\right)^h. \quad (4.22)$$

Therefore, the number of sources that a receiver observes on the average is given by

$$T_i^{av} = h(1 - \epsilon) = h\left[1 - \left(1 - \frac{1}{h}\right)^h\right] \gtrsim h(1 - e^{-1}) > \frac{T_c}{2}. \quad (4.23)$$

Some thought is required to realize that this routing scenario can be cast into a classic urns and balls experiment in which  $h$  balls (corresponding to the receiver's  $h$  incoming edges) are thrown independently and uniformly into  $h$  urns (corresponding to the  $h$  sources). The established theory of such occupancy models can tell us not only

the expected value of the random variable representing the number of observed sources but also its entire probability distribution. Moreover, we can say that as the number of sources increases, the probability that the normalized difference between the observed throughput is greater than some value  $x$  decreases exponentially fast with  $x^2$ . Similar concentration results hold when the number of receivers is large (the rates they experience tend to concentrate around a much larger value than the minimum) and give us yet another reason for looking at the average throughput.

---

The next example illustrates what can be achieved by coding at the source that was described in Section 3.6.3.

---

**Example 4.6.** If in Example 4.5, in addition to the described randomized routing scheme, the source is allowed to apply channel coding (over time), then the integral throughput can achieve the average asymptotically over time. To see that, we first note that under the randomized routing, which is repeated at each time instant, each receiver observes the sequence of each source outputs as if it had passed through an erasure channel with the probability of erasure  $\epsilon$  given by (4.22). The capacity of such a channel is equal to  $1 - \epsilon$ . Information theory tells us that there exists a sequence of codes with rates  $k/n$  such that the probability of incorrect decoding goes to 0 and  $k/n \rightarrow 1 - \epsilon$  as  $n \rightarrow \infty$ . Such codes can be applied at each source, enabling each receiver to get the same throughput. Therefore, since there are  $h$  sources, we have  $T_i(n) \rightarrow h \cdot k/n$  as  $n \rightarrow \infty$ . Since  $k/n$  can be taken arbitrary close to the capacity, we have  $T_i(n) \rightarrow h(1 - \epsilon) = T_i$ .

These results, using uniform at random allocation of the sources, hold in the case of  $B(h, k)$  combination networks, and additionally, over networks with  $h$  sources,  $k$  coding point  $A_i B_i$  and an arbitrary number of receivers observing the coding points. When the configuration is symmetric, as in the case of  $B(h, k)$  networks, the random routing can be replaced by deterministic, and the integral throughput  $T_i$  can achieve the average in a finite number of time units.

---

## 4.5 Undirected Graphs

We now consider undirected networks, where through each edge  $e = (x, y)$  we can simultaneously route flows  $f_{xy}$  and  $f_{yx}$  in opposite directions, provided their sum does not exceed the capacity of the edge, i.e.,  $f_{xy} + f_{yx} \leq c_e$ . In undirected graphs, the coding advantage of network coding is bounded by a constant factor of two.

---

**Theorem 4.4.** In multicasting over undirected graphs, the throughput benefits network coding offers as compared to routing are upper bounded by a factor of two

$$T_c \leq 2T_f.$$


---

To prove the theorem, we use the following result for directed graphs.

---

**Theorem 4.5 (Bang-Jensen, 1995).** Let  $G = (V, E)$  be a directed graph with the set of receivers  $\mathcal{R} \subset V$ , such that at all vertices in  $V$  other than the source and the receivers, the in-degree is greater or equal than the out-degree. Assume that receiver  $R_i \in \mathcal{R}$  has min-cut  $m_i$  from the source, and let  $k = \max_{R_i \in \mathcal{R}} m_i$ . Then there are  $k$  edge-disjoint partial Steiner trees in  $G$  such that each node  $R_i \in \mathcal{R}$  occurs in  $m_i$  of the  $k$  trees.

---

*Proof.* Consider an instance  $\{G, S, \mathcal{R}\}$  where now  $G$  is an undirected graph with unit capacity edges, and where the min-cut to each receiver equals  $h$ . Therefore,  $T_c = h$ . Replace each undirected edge with two opposite directed edges, each of capacity  $1/2$ . This graph has the following properties:

- (1) The min-cut to each receiver is at least  $h/2$ .
- (2) The in-degree at each intermediate vertex equals the out-degree.

Clearly, the conditions of Theorem 4.5 are satisfied. Therefore, there are  $h/2$  edge-disjoint Steiner trees such that each receiver occurs in

$h/2$  of them. In other words, these trees span all receivers, and thus the routing throughput can be made to be at least  $h/2$ . We conclude that  $T_c \leq 2T_f$ .  $\square$

Theorem 4.5 actually provides a more general claim than what we need to prove Theorem 4.4. It additionally tells us that, even if the min-cut to each receiver is different, that is, we have a *non-uniform demand* network, over undirected graphs we can still send to each receiver rate larger or equal to half its min-cut value using routing.

One should be aware that although network coding does not offer significant throughput benefits for multicasting over undirected graphs, it does offer benefits in terms of the transmission scheme design complexity. Network coding schemes achieving throughput  $T_c$  can be designed in polynomial time (solving the LP in Section 3.5 and then using any network code design algorithm), while finding the optimal routing (packing Steiner trees) is a computationally hard problem. We also note however that empirical studies that use polynomial time approximation algorithms to solve the routing problem showed a comparable performance to network coding.

## Notes

The first multicast throughput benefits in network coding referred to the symmetric integral throughput in directed networks, and were reported by Sanders *et al.* in [43]. Theorem 4.1 was subsequently proved by Agarwal and Charikar in [1], and it was extended to Theorem 4.2 by Chekuri *et al.* in [15]. The interested reader is referred to [44] for details on integer and linear programs, and to [4, Ch. 9] for details on the Steiner tree problem formulations. The ZK networks were introduced by Zosin and Khuller in [50], and were used to demonstrate network coding throughput benefits in [1, 15]. Theorem 4.5 was proved Bang-Jensen *et al.* in [5]. Throughput benefits over undirected graphs were examined by Li *et al.* in [37], and by Chekuri *et al.* in [14]. Information theoretic rate bounds over undirected networks with two-way channels were provided by Kramer and Savari in [33]. Experimental results by Wu *et al.* reported in [47] showed small throughput bene-

fits over undirected network graphs of six Internet service providers. Throughput benefits that network coding can offer for other types of traffic scenarios were examined by Rasala-Lehman and Lehman in [41], and by Dougherty *et al.* in [19]. Non-uniform demand networks were examined by Cassuto and Bruck in [13] and later in [14].

# 5

---

## Network Code Design Methods for Multicasting

---

We now look at network code design algorithms for multicasting under the assumptions of the main network coding theorem (Theorem 2.2). We assume the network multicast model as established in Section 3.1: namely, a directed acyclic graph with unit capacity edges where the min-cut to each of the  $N$  receivers equals the number of unit-rate sources  $h$ .

Perhaps the most important point of this chapter is that, provided we can use an alphabet of size at least as large as the number of receivers, we can design network codes in *polynomial time*.<sup>1</sup> This is what makes network coding appealing from a practical point of view. Maximizing the multicast rate using routing is, as we discussed in Section 3.5, NP-hard. On the other hand, coding solutions that achieve higher rates than routing (namely, the multicast capacity) can be found in polynomial time. Moreover, a very simple and versatile randomized approach to coding allows to achieve very good performance in terms of rate, and has been considered for use in practice.

---

<sup>1</sup>Polynomial in the number of edges of the network graph.

We classify network code design algorithms based on the type of information they require as their input: we say that an algorithm is *centralized* if it relies on global information about the entire network structure, and *decentralized* if it requires only local and no topological information. We start this chapter by describing a common initial procedure that all algorithms execute. We then present examples of both centralized and decentralized algorithms, and discuss their various properties, such as scalability to network changes.

## 5.1 Common Initial Procedure

Given a multicast instance  $\{G = (V, E), S, \mathcal{R}\}$ , the first common steps in all the algorithms described in this chapter are to

- (1) find  $h$  edge-disjoint paths  $\{(S_i, R_j), 1 \leq i \leq h\}$  from the sources to the receivers, the associated graph  $G' = \bigcup_{\substack{1 \leq i \leq h \\ 1 \leq j \leq N}} (S_i, R_j)$ , the set of coding points  $\mathcal{C}$ , and
- (2) find the associated minimal configuration.

Recall that a configuration is minimal if removing any edge would violate the min-cut condition for at least one receiver (Definition 3.5), and coding points are edges in the network where two or more incoming information symbols have to be combined (Definition 3.4).

Step (1) is necessary in all the algorithms described in this chapter. To implement it we can use any max flow algorithm, such as the one given within the proof of the min-cut max-flow theorem (Theorem 2.1). A flow-augmenting path can be found in time  $\mathcal{O}(|E|)$ , thus the total complexity is  $\mathcal{O}(|E|hN)$ .

Step (2) is not necessary, but it can significantly reduce the required network resources, such as the number of coding points and the number of employed edges by the information flow. Algorithm 5.1 describes a brute force implementation, which sequentially attempts to remove each edge and check if the min-cut condition to each receiver is still satisfied. This approach requires  $\mathcal{O}(|E|^2hN)$  operations.

**Algorithm 5.1:** INITIAL PROCESSING( $G, S, \mathcal{R}$ )Find  $(S_i, R_j)$  for all  $i, j$ 

$$G' = (V', E') \leftarrow \bigcup_{\substack{1 \leq i \leq h \\ 1 \leq j \leq N}} (S_i, R_j)$$

$$\forall e \in E' \left\{ \begin{array}{l} \text{if } (V', E' \setminus \{e\}) \text{ satisfies the multicast property} \\ \text{then } \left\{ \begin{array}{l} E' \leftarrow E' \setminus \{e\} \\ G' \leftarrow (V', E') \end{array} \right. \end{array} \right.$$
**return**  $(G')$ 

## 5.2 Centralized Algorithms

### 5.2.1 Linear Information Flow Algorithm

Consider an instance  $\{G, S, \mathcal{R}\}$ . A given network code (choice of coding vectors) conveys to receiver  $R_j$  rate at most equal to the number of independent linear combinations that the receiver observes. In that sense, we can (with some abuse of terminology) talk about the min-cut value a receiver experiences under a given network code, and of whether a specific network code preserves the multicast property. In other words, we can think of the network code as being part of the channel.

The Linear Information Flow (LIF) is a greedy algorithm based exactly on the observation that the choice of coding vectors should preserve the multicast property of the network. The algorithm sequentially visits the coding points in a topological order<sup>2</sup> (thus ensuring that no coding point is visited before any of its ancestors in the graph), and assigns coding vectors to them. Each assigned coding vector preserves the multicast property for all downstream receivers up to the coding point to which it is assigned. Intuitively, the algorithm preserves  $h$  “degrees of freedom” on the paths from the sources to each receiver.

<sup>2</sup>A topological order is simply a partial order. Such an order exists for the edges of any graph  $G$  that is acyclic.

To precisely describe the algorithm, we need some additional notation. Let  $\mathcal{C}$  denote the set of all coding points and  $R(\delta)$  the set of all receivers that employ a coding point  $\delta$  in one of their paths. Each coding point  $\delta$  appears in at most one path  $(S_i, R_j)$  for each receiver  $R_j$ . Let  $f_{\leftarrow}^j(\delta)$  denote the predecessor coding point to  $\delta$  along this path  $(S_i, R_j)$ .

For each receiver  $R_j$ , the algorithm maintains a set  $\mathcal{C}_j$  of  $h$  coding points, and a set  $B_j = \{c_1^j, \dots, c_h^j\}$  of  $h$  coding vectors. The set  $\mathcal{C}_j$  keeps track of the most recently visited coding point in each of the  $h$  edge disjoint paths from the source to  $R_j$ . The set  $B_j$  keeps the associated coding vectors.

Initially, for all  $R_j$ , the set  $\mathcal{C}_j$  contains the source nodes  $\{S_1^e, S_2^e, \dots, S_h^e\}$ , and the set  $B_j$  contains the orthonormal basis  $\{e_1, e_2, \dots, e_h\}$ , where the vector  $e_i$  has one in position  $i$  and zero elsewhere. Preserving the multicast property amounts to ensuring that the set  $B_j$  forms a basis of the  $h$ -dimensional space  $\mathbb{F}_q^h$  at all steps of the algorithm and for all receivers  $R_j$ . At step  $k$ , the algorithm assigns a coding vector  $c(\delta_k)$  to the coding point  $\delta_k$ , and replaces, for all receivers  $R_j \in R(\delta_k)$ :

- the point  $f_{\leftarrow}^j(\delta_k)$  in  $\mathcal{C}_j$  with the point  $\delta_k$ ,
- the associated vector  $c(f_{\leftarrow}^j(\delta_k))$  in  $B_j$  with  $c(\delta_k)$ .

The algorithm selects the vector  $c(\delta_k)$  so that for every receiver  $R_j \in R(\delta_k)$ , the set  $(B_j \setminus \{c(f_{\leftarrow}^j(\delta_k))\}) \cup c(\delta_k)$  forms a basis of the  $h$ -dimensional space. Such a vector  $c(\delta_k)$  always exists, provided that the field  $\mathbb{F}_q$  has size larger than  $N$ , as Lemma 5.2 proves. When the algorithm terminates, the set  $B_j$  contains the set of linear equations the receiver  $R_j$  needs to solve to retrieve the source symbols.

---

**Lemma 5.1.** Consider a coding point  $\delta$  with  $m \leq h$  parents and a receiver  $R_j \in R(\delta)$ . Let  $\mathcal{V}(\delta)$  be the  $m$ -dimensional space spanned by the coding vectors of the parents of  $\delta$ , and  $\mathcal{V}(R_j, \delta)$  be the  $(h - 1)$ -dimensional space spanned by the elements of  $B_j$  after removing

$c(f_{\leftarrow}^j(\delta))$ . Then

$$\dim\{\mathcal{V}(\delta) \cap \mathcal{V}(R_j, \delta)\} = m - 1.$$

---

*Proof.* Since the dimension of the intersection of two subspaces  $A$  and  $B$  is given by

$$\dim\{A \cap B\} = \dim A + \dim B - \dim\{A \cup B\},$$

we only need to show that  $\dim\{\mathcal{V}(\delta) \cup \mathcal{V}(R_j, \delta)\} = h$ . This is true because  $\mathcal{V}(\delta)$  contains  $c(f_{\leftarrow}^j(\delta))$  and  $\mathcal{V}(R_j, \delta)$  contains the rest of the basis  $B_j$ .  $\square$

---

**Lemma 5.2.** The LIF algorithm successfully identifies a valid network code using any alphabet  $\mathbb{F}_q$  of size  $q > N$ .

---

*Proof.* Consider a coding point  $\delta$  with  $m \leq h$  parents and a receiver  $R_j \in R(\delta)$ . The coding vector  $c(\delta)$  has to be a non-zero vector in the  $m$ -dimensional space  $\mathcal{V}(\delta)$  spanned by the coding vectors of the parents of  $\delta$ . There are  $q^m - 1$  such vectors, feasible for  $c(\delta)$ . To make the network code valid for  $R_j$ ,  $c(\delta)$  should not belong to the intersection of  $\mathcal{V}(\delta)$  and the  $(h - 1)$ -dimensional space  $\mathcal{V}(R_j, \delta)$  spanned by the elements of  $B_j$  after removing  $c(f_{\leftarrow}^j(\delta))$ . The dimension of this intersection is  $m - 1$ , and thus the number of the vectors it excludes from  $\mathcal{V}(\delta)$  is  $q^{m-1} - 1$ . Therefore, the number of vectors excluded by  $|R(\delta)|$  receivers in  $\delta$  is at most  $|R(\delta)|q^{m-1} - 1 \leq Nq^{m-1} - 1$ . (Here, the all-zero vector is counted only once.) Therefore, provided that

$$q^m > Nq^{m-1} \Leftrightarrow q > N,$$

we can find a valid value for  $c(\delta)$ . Applying the same argument successively to all coding points concludes the proof.  $\square$

Note that that this bound may not be tight, since it was obtained by assuming that the only intersection of the  $|R(\delta)|$   $(h - 1)$ -dimensional spaces  $\mathcal{V}(R_j, \delta)$ ,  $R_j \in R(\delta)$  is the all-zero vector, which was counted

only once. However, each pair of these spaces intersects on an  $(h - 2)$ -dimensional subspace. Tight alphabet size bounds are known for networks with two sources and for some other special class of networks, which we discuss in Chapter 7.

The remaining point to discuss is how the algorithm identifies a valid coding vector  $c(\delta_k)$  at step  $k$ . It is possible to do so either deterministically (e.g., by exhaustive search), or by randomly selecting a value for  $c(\delta_k)$ .

---

**Lemma 5.3.** A randomly selected coding vector  $c(\delta_k)$  at step  $k$  of the LIF preserves the multicast property with probability at least  $1 - N/q$ .

---

*Proof.* Receiver  $R_i \in R(\delta_k)$  requires that  $c(\delta_k)$  does not belong to the  $(m - 1)$ -dimensional space spanned by the elements of  $B_j \setminus \{c(f_{\leftarrow}^j(\delta_k))\}$ . Randomly selecting a vector in  $\mathbb{F}_q^m$  does not satisfy this property with probability  $q^{m-1}/q^m = 1/q$ . Using the fact that  $|R(\delta_k)| \leq N$  and applying the union bound the result follows.  $\square$

From Lemma 5.3, if for example we choose a field  $\mathbb{F}_q$  of size  $q > 2N$  the probability of success for a single coding vector is at least  $\frac{1}{2}$ . Thus repeating the experiment on the average two times per coding point we can construct a valid network code.

A final observation is that we can efficiently check whether the selected vector  $c(\delta_k)$  belongs to the space spanned by  $B_j \setminus \{c(f_{\leftarrow}^j(\delta_k))\}$  by keeping track of, and performing an inner product with, a vector  $\alpha_j$ , that is orthogonal to this subspace.<sup>3</sup> The LIF algorithm is summarized in the following Algorithm 5.2. The algorithm can be implemented to run in expected time  $\mathcal{O}(|E|Nh^2)$ .

---

<sup>3</sup>Note that  $B_j \setminus \{c(f_{\leftarrow}^j(\delta_k))\}$  forms an hyperplane, i.e., an  $h - 1$  dimensional subspace, of the  $h$ -dimensional space that can be uniquely described by the associated orthogonal vector.

**Algorithm 5.2:** LIF( $\{G, S, \mathcal{R}\}, \mathbb{F}_q$  with  $q > N$ )

$\mathcal{C} \leftarrow$  coding points of  $\{G, S, \mathcal{R}\}$  in topological order  
 $\forall \delta \in \mathcal{C}, \mathcal{R}(\delta) \leftarrow \{R_j \text{ such that } \delta \text{ is in a path } (S_i, R_j)\}$   
 $\forall R_j \in \mathcal{R}, \begin{cases} B_j \leftarrow \{e_1, e_2, \dots, e_h\}, \\ C_j \leftarrow \{S_1^e, S_2^e, \dots, S_h^e\} \\ \forall \delta \in C_j, \quad a_j(\delta) = e_j \end{cases}$   
• Repeat for  $k$  steps,  $1 \leq k \leq |\mathcal{C}|$   
At step  $k$  access  $\delta_k \in \mathcal{C}$   
and  $\begin{cases} \text{Find } c(\delta_k) \text{ such that } c(\delta_k) \cdot a_j(\delta_k) \neq 0 \forall R_j \in \mathcal{R}(\delta_k) \\ \begin{cases} C_j \leftarrow (C_j \setminus \{f_{-}^j(\delta_k)\}) \cup \delta_k \\ B_j \leftarrow (B_j \setminus \{c(f_{-}^j(\delta_k))\}) \cup c(\delta_k) \\ \alpha_j(\delta_k) \leftarrow \frac{1}{c(\delta_k) \cdot a_j(f_{-}^j(\delta_k))} a_j(f_{-}^j(\delta_k)) \\ \forall \delta \in C_j \setminus \delta_k : \\ \quad \alpha_j(\delta) \leftarrow \alpha_j(\delta) - (c(\delta_k) \cdot \alpha_j(\delta)) \alpha_j(\delta_k) \end{cases} \end{cases}$   
**return** (the coding vectors  $c(\delta), \forall \delta \in \mathcal{C}$ )

### 5.2.2 Matrix Completion Algorithms

The problem of network code design can be reduced to a problem of completing mixed matrices. Mixed matrices have entries that consist both of unknown variables and numerical values. The problem of matrix completion is to select values for the variables so that the matrix satisfies desired properties. For the network code design, the desired property is that  $N$  matrices sharing common variables are simultaneously full rank. These are the  $N$  transfer matrices from the source to each receiver, as discussed in Section 3.2.

We now discuss a very nice connection between bipartite graph matching and matrix completion. Such ideas have been extended to deterministic algorithms for network code design.

Consider a matrix  $\mathbf{A}$  whose every entry is either zero or an unknown variable. The problem of selecting the variable values so that the matrix is full rank over the binary field (if possible) can be reduced to a bipartite matching problem as follows. Construct a bipartite graph with the

left nodes corresponding to the rows and the right nodes the columns of matrix  $\mathbf{A}$ . For a non-zero entry in the position  $(i, j)$  in  $\mathbf{A}$ , place an edge between left node  $i$  and right node  $j$ . A perfect matching is a subset of the edges such that each node of the constructed bipartite graph is adjacent to exactly one of the edges. Identifying a perfect matching can be accomplished in polynomial time. It is also easy to see that the identified matching corresponds to an assignment of binary values to the matrix entries (one for an edge used, zero otherwise) such that the matrix is full rank.

### 5.3 Decentralized Algorithms

#### 5.3.1 Random Assignment

The basic idea in this algorithm is to operate over a field  $\mathbb{F}_q$  with  $q$  large enough for even random choices of the coding vector coefficients to offer with high probability a valid solution. This algorithm requires no centralized or local information, as nodes in the network simply choose the non-zero components of local coding vectors independently and uniformly from  $\mathbb{F}_q$ . The associated probability of error can be made arbitrarily small by selecting a suitably large alphabet size, as the following theorem states.

---

**Theorem 5.4.** Consider an instance  $\{G = (V, E), S, \mathcal{R}\}$  with  $N = |\mathcal{R}|$  receivers, where the components of local coding vectors are chosen uniformly at random from a field  $\mathbb{F}_q$  with  $q > N$ . The probability that all  $N$  receivers can decode all  $h$  sources is at least  $(1 - N/q)^{\eta'}$ , where  $\eta' \leq |E|$  is the maximum number of coding points employed by any receiver.

---

That is,  $\eta'$  is the number of coding points encountered in the  $h$  paths from the source node to any receiver, maximized over all receivers. Recall that a network code is valid if it assigns values to variables  $\alpha_1, \dots, \alpha_{\eta'}$ , ( $\eta' < \eta$ ) such that

$$f(\alpha_1, \dots, \alpha_{\eta'}) = \det \mathbf{A}_1 \det \mathbf{A}_2 \cdots \det \mathbf{A}_N \neq 0, \quad (5.1)$$

where  $\mathbf{A}_j$  is the transfer matrix for receiver  $j$ ,  $1 \leq j \leq N$ . We have seen in Chapters 2 and 3 that for acyclic directed networks,  $f$  is a polynomial with maximum degree in each variable at most  $N$ . With this observation, the following lemma is sufficient to prove Theorem 5.4.

---

**Lemma 5.5.** Let  $f(\alpha_1, \dots, \alpha_\eta)$  be a multivariate polynomial in variables  $\alpha_1, \dots, \alpha_\eta$ , with maximum degree in each variable of at most  $N$  and total degree at most  $N\eta'$ . If the values for  $\alpha_1, \dots, \alpha_\eta$  are chosen uniformly at random from a finite field  $\mathbb{F}_q$  of size  $q > N$  on which  $f$  is not identically equal to zero, then the probability that  $f(\alpha_1, \dots, \alpha_\eta)$  equals zero is at most  $1 - (1 - N/q)^{\eta'}$ .

---

We omit the proof of this lemma, and instead present a simple proof of a slightly weaker but more often used result which, in Theorem 5.4, replaces  $\eta'$  by the total number of local coding coefficients  $\eta$ . For this result, the following lemma is sufficient.

---

**Lemma 5.6.** Let  $f(\alpha_1, \dots, \alpha_\eta)$  be a multivariate polynomial in  $\eta$  variables with the maximum degree in each variable of at most  $N$ . If the values for  $\alpha_1, \dots, \alpha_\eta$  are chosen uniformly at random from a finite field  $\mathbb{F}_q$  of size  $q > N$  on which  $f$  is not identically equal to zero, then the probability that  $f(\alpha_1, \dots, \alpha_\eta)$  equals zero is at most  $1 - (1 - N/q)^\eta$ .

---

*Proof.* We prove the lemma by induction in the number of variables  $\eta$ :

- (1) For  $\eta = 1$ , we have a polynomial in a single variable of degree at most  $N$ . Since such polynomials can have at most  $N$  roots, a randomly chosen element from a field  $\mathbb{F}_q$  satisfying the conditions of the theorem will be a root of  $f$  with probability of at most  $N/q$ .
- (2) For  $\eta > 1$ , the inductive hypothesis is that the claim holds for all polynomials with fewer than  $\eta$  variables. We express  $f$  in the form

$$f(\alpha_1, \dots, \alpha_\eta) = \alpha_\eta^d f_1(\alpha_1, \dots, \alpha_{\eta-1}) + f_2(\alpha_1, \dots, \alpha_\eta),$$

where  $d \leq N$ ,  $f_1$  is a not-identically zero polynomial in  $\eta - 1$  variables, and  $f_2$  is a polynomial in  $\eta$  variables with degree in each variable of at most  $N$ , that may contain all other powers  $\alpha_\eta$  except for  $\alpha_\eta^d$ . Assume that we uniformly at random chose values for the variables  $\alpha_1, \dots, \alpha_\eta$  from a finite field  $\mathbb{F}_q$  of size  $q > N$ . Then

$$\begin{aligned} \Pr[f = 0] &= \Pr[f_1 = 0] \cdot \Pr[f = 0|f_1 = 0] \\ &\quad + (1 - \Pr[f_1 = 0]) \cdot \Pr[f = 0|f_1 \neq 0]. \end{aligned}$$

We now bound the above three probabilities as follows:

- (a)  $\Pr[f_1 = 0] \leq 1 - (1 - N/q)^{(\eta-1)}$  by the inductive hypothesis.
- (b)  $\Pr[f = 0|f_1 = 0] \leq 1$ .
- (c)  $\Pr[f = 0|f_1 \neq 0] \leq N/q$ , since when we evaluate  $f$  at the values of  $\alpha_1, \dots, \alpha_{\eta-1}$  it becomes a polynomial in  $\alpha_\eta$  of degree at least  $d$  and at most  $N$ .

Therefore

$$\begin{aligned} \Pr[f = 0] &\leq \Pr[f_1 = 0] + (1 - \Pr[f_1 = 0]) \frac{N}{q} \quad \text{by (b) and (c)} \\ &= \Pr[f_1 = 0] \left(1 - \frac{N}{q}\right) + \frac{N}{q} \\ &\leq \left[1 - \left(1 - \frac{N}{q}\right)^{(\eta-1)}\right] \left(1 - \frac{N}{q}\right) + \frac{N}{q} \quad \text{by (a)} \\ &= 1 - \left(1 - \frac{N}{q}\right)^\eta. \quad \square \end{aligned}$$

The randomized coding approach may use a larger alphabet than necessary. However, it is decentralized, scalable and yields to a very simple implementation. In fact, as we discuss in the second part of this review, it is very well suited to a number of practical applications, such as dynamically changing networks. We next briefly discuss how this approach, and more generally, network coding, can be implemented in practice.

### 5.3.2 Network Coding in Practice: Use of Generations

In practical networks, information is sent in packets. Each packet consists of  $L$  information bits, and sets of  $m$  consecutive bits are treated as a symbol of  $\mathbb{F}_q$  with  $q = 2^m$ . Coding amounts to linear operations over the field  $\mathbb{F}_q$ . The same operations are applied symbol-wise to each of the  $L/m$  symbols of the packet.

Over the Internet packets between a source–destination pair are subject to random delays, may get dropped, and often follow different routes. Moreover, sources may produce packets asynchronously. It is thus difficult to implement a centralized network coding algorithm. To deal with the lack of synchronization, the packets are grouped into *generations*. Packets are combined only with other packets in the same generation. A generation number is also appended to the packet headers to make this possible (one byte is sufficient for this purpose).

Assume that a generation consists of  $h$  information packets, which we will call source packets. A feasible approach to perform network coding in a distributed and asynchronous manner, is to append to *each* packet header an  $h$ -dimensional coding vector that describes how the packet is encoded, that is, which linear combination of the source packets it carries.

Intermediate nodes in the network buffer the packets they receive. To create a packet to send, they uniformly at random select coefficients, and symbol-wise combine buffered packets that belong in the same generation. They use the packet’s header information to calculate the new coding vector to append. The receivers use the appended coding vectors to decode.

Note that use of the coding vector in the header incurs a small additional overhead. For example, for a packet that contains 1400 bytes, where every byte is treated as a symbol over  $\mathbb{F}_{2^8}$ , if we have  $h = 50$  sources, then the overhead is approximately  $50/1400 \approx 3.6\%$ .

The size of a generation can be thought of as the number of sources  $h$  in synchronized networks: it determines the size of matrices the receivers need to invert to decode the information. Since inverting an  $h \times h$  matrix requires  $\mathcal{O}(h^3)$  operations, and also affects the delay, it is desirable to keep the generation size small. On the other hand, the

size of the generation affects how well packets are “mixed,” and thus it is desirable to have a fairly large generation size. Indeed, if we use a large number of small-size generations, intermediate nodes may receive packets destined to the same receivers but belonging to different generations. Experimental studies have started investigating this trade-off.

### 5.3.3 Permute-and-Add Codes

Randomized network code requires each intermediate node to perform  $\mathcal{O}(h^2)$  operations for every packet it encodes. The codes we describe in this section reduce this complexity to  $\mathcal{O}(h)$ . These codes can be thought of as performing, instead of the randomized network coding over  $\mathbb{F}_q$  described previously, randomized *vector* network coding over  $\mathbb{F}_2^L$ , with  $q = 2^L$ . However, to achieve low encoding complexity, these codes do not randomly select  $L \times L$  matrices over  $\mathbb{F}_2^L$ , but restrict the random selection over permutation matrices. Each source produces a length  $L$  binary packet. As their name indicates, the permute-and-add codes have intermediate nodes permute each of their incoming binary packets and then xor them, to generate the new packet to send. Intermediate nodes choose uniformly at random which of the  $L!$  possible permutation to apply to each of their incoming packets. Each receiver solves a set of  $hL \times hL$  binary equations to retrieve the information data.

---

**Theorem 5.7.** Let  $h$  be the binary min-cut capacity. For any  $\epsilon > 0$ , the permute-and-add code achieves rate  $R = h - (|E| + 1)\epsilon$  with probability greater than  $1 - 2^{L\epsilon + \log(N \frac{|E|+h}{L+1})}$ .

---

We here only give a high-level outline of the proof. The basic idea is an extension of the polynomial time algorithms presented in Section 5.2.1. To show that the transfer matrix from the source to each receiver is invertible, it is sufficient to make sure that the information crossing specific cut-sets allows to infer the source information. These cut-sets can be partially ordered, so that no cut-set is visited before its “ancestor” cut-sets. The problem is then reduced to ensuring that the transfer matrix between successive cut-sets is, with high probability, full rank.

### 5.3.4 Algorithms for Certain Classes of Networks

These algorithms use some high level a priori information about the class of networks the multicast configuration belongs to. The following example gives simple decentralized algorithms for networks with two receivers.

---

#### Example 5.1. Codes for networks with two receivers

If we have a minimal configuration with  $h$  sources and two receivers, each coding point needs to simply xor its incoming information symbols to get a valid network code. This is because the binary alphabet is sufficient for networks with two receivers, and for a minimal configuration, each input to a coding point have to be multiplied by non-zero coefficients.

---

Codes for networks with two sources are also simple to construct.

---

#### Example 5.2. Codes for networks with two sources

As discussed in Section 3.3, to label the nodes of a subtree graph of a network with two sources, we can use the points on the projective line  $\mathbb{P}\mathbb{G}(1, q)$

$$[01], [10], \quad \text{and} \quad [1\alpha^i] \quad \text{for } 0 \leq i \leq q - 2. \quad (5.2)$$

Recall that for a valid network code, it is sufficient and necessary that the coding vector associated with a subtree lies in the linear span of the coding vectors associated with its parent subtrees, and the coding vectors of any two subtrees having a receiver in common are linearly independent. Since any two different points of (5.2) are linearly independent in  $\mathbb{F}_q^2$ , and thus each point is in the span of any two different points of (5.2), both coding conditions for a valid network code are satisfied if each node in a subtree graph of a network is assigned a unique point of the projective line  $\mathbb{P}\mathbb{G}(1, q)$ . We here present two algorithms which assign distinct coding vectors to the nodes in the subtree graph.

In Chapter 7, we will discuss a connection between the problem of designing codes for networks with two sources and the problem of

graph coloring. Using this connection we can additionally employ all graph coloring algorithms for network code design in this special case. Such algorithms may result in codes that are more efficient in terms of the alphabet size. However, those presented here are decentralized and mostly scalable to network changes.

The first method is inspired by the Carrier Sense Multiple Access (CSMA) systems, where access to the network is governed by the possession of a *token*. The token is circulating around the network, and when a device needs to transmit data, it seizes the token when it arrives at the device. The token remains at the possession of the transmission device until the data transfer is finished. In our algorithm, a token will be generated for each coding vector. Each coding point (associated terminal) seizes a token and uses the associated coding vector. If a change in the network occurs, for example, receivers leave the multicast session, tokens that are no longer in use may be released back in the network to be possibly used by other coding points.

An alternative simple way to organize a mapping from coding vectors to coding points is described below. Recall that at each subtree, we locally know which receivers it contains and which sources are associated with each receiver (at the terminal before the coding point). In networks with two sources, each coding subtree contains at least one receiver node associated with  $S_1$  and at least one receiver node associated with  $S_2$ . Let  $\mathfrak{R}(T; S_1) = \{R_{i_1}, R_{i_2}, \dots, R_{i_u}\}$ , where  $i_1 < i_2 < \dots < i_u$  be the set of receivers associated with  $S_1$  in a given subtree  $T$ . We choose  $[1 \alpha^{i_1-1}]$  to be the label of that subtree. This way no other subtree can be assigned the same label since the receiver  $R_{i_1}$  can be associated with the source  $S_1$  in at most one subtree. Note that this is not the most efficient mapping as it may require alphabet size of  $q = N + 1$ , as opposed to  $q = N$ . This is because for  $N$  receivers, we will use coding vectors from the set  $[1 \alpha^i]$  for  $0 \leq i \leq q - 2$  with  $q - 2 = N - 1$ .

---

We now discuss how the code design problem can be simplified at the cost of using some additional network resources. These algorithms are well suited to applications such as overlay and ad-hoc networks, where

we have at our disposal large graphs rather than predetermined sets of paths from sources to receivers.

---

**Example 5.3.** One way to take advantage of the increased connectivity is to (if possible) partition the network  $G$  with  $h$  sources ( $h$  even) into  $h/2$  independent two-source configurations, and then code each subnetwork separately in a possibly decentralized manner. Algorithm 5.3 outlines this procedure.

---

**Algorithm 5.3:** PAIRED-SOURCES CODE( $G$ )

Group sources into pairs, and  
**for each** pair of sources  $(S_{i_1}, S_{i_2})$   
 { Find paths  $(S_{i_1}, R_j), (S_{i_2}, R_j), 1 \leq j \leq N$  in  $G$   
 { Design a network code based on these paths.  
 { Update  $G$  by removing the used paths.

---

The following example provides the natural generalization of Example 5.2 to networks with  $h$  sources.

---

**Example 5.4. Codes that employ vectors in general position**

In a network with  $|\mathcal{C}|$  coding points, a simple decentralized code design is possible if we are willing to use an alphabet of size  $|\mathcal{C}| + h - 1$  as well as additional network resources (edges and terminals) to ensure that the min-cut to each coding point be  $h$ . The basic idea is that the  $h$  coding vectors of the parents of each coding point  $T$ , form a basis of the  $h$ -dimensional space. Thus, any coding vector in the  $h$ -dimensional space is eligible as  $c(T)$ . Additionally, selecting the coding vectors to be points in general position (see Appendix) ensures that each receiver observes  $h$  linearly independent coding vectors.

More specifically, for a minimal configuration with  $h$  sources and  $|\mathcal{C}|$  coding points where the min-cut to each coding point is  $h$ , an alphabet of size  $|\mathcal{C}| + h - 1$  is sufficiently large for decentralized coding. Indeed, we can use the  $|\mathcal{C}| + h$  points in general position in a normal rational

curve in  $\mathbb{P}\mathbb{G}(h - 1, |\mathcal{C}| + h - 1)$  (see Appendix) to assign  $h$  vectors to the source subtrees and  $|\mathcal{C}|$  vectors to the coding subtrees.

Here, we can think of coding points as edges incident to special network terminals, that have not only enhanced computational power, but also enhanced connectivity. Note that, even in this case, multicast with network coding requires lower connectivity than multicast without coding, because multicast without coding is possible iff the min-cut toward *every* node of the graph is  $h$ , not just the coding points.

---

## 5.4 Scalability to Network Changes

A desirable characteristic of network codes is that codes do not change when receivers join or leave the network. For the following discussion, we will use the subtree decomposition described in Section 3.3. An advantage of decentralized codes that employ vectors in general position is that they do not have to be changed with the growth of the network as long as the network's subtree decomposition retains the same topology regardless of the distribution of the receivers, or the new subtree graph contains the original subtree graph.

In a network using decentralized coding, the coding vectors associated with any  $h$  subtrees provide a basis of the  $h$ -dimensional space. We can think of subtrees as “secondary sources” and allow the new receivers to connect to any  $h$  different subtrees. Thus we can extend the multicast network, without any coding/decoding changes for existing users. Subtree decomposition enables scalability in the described manner even when the code is not decentralized, since we can have new receivers contact subtrees, much like today terminals contact servers, until they connect to  $h$  subtrees that allow them to retrieve the source information.

In the above scenario, the receivers are allowed to join the network only in a way that will not change the topology of the subtree graph. Alternatively, if for example in a network with two sources, addition of new receivers results in new subtrees without disturbing the existing ones, then the existing code can be simply extended without any coding/decoding changes for existing users. Note that the projective line  $\mathbb{P}\mathbb{G}(1, q)$  (see Appendix) can be thought of as a subset of the projective

line  $\mathbb{P}\mathbb{G}(1, q')$ , where  $\mathbb{F}_{q'}$  is an extension field of  $\mathbb{F}_q$ . Thus, if we need to create additional coding vectors to allocate to new subtrees, we can employ unused points from the projective line  $\mathbb{P}\mathbb{G}(1, q')$ .

## Notes

LIF, proposed by Sanders *et al.* in [43], and independently by Jaggi *et al.* in [29] was the first polynomial algorithms for network code design (see also [30]). These algorithms were later extended to include procedures that attempt to minimize the required field size by Barbero and Ytrehus in [6]. Randomized algorithms were proposed by Ho *et al.* in [26], and also by Sanders *et al.* in [43], and their asynchronous implementation over practical networks using generations by Chou *et al.* in [16]. Lemma 5.5 instrumental for Theorem 5.4 was proved by Ho *et al.* in [26]. Codes that use the algebraic structure were designed by Koetter and Médard [32], while the matrix completion codes were investigated by Harvey in [25]. Permute-and-add codes were recently proposed by Jaggi *et al.* in [28]. Decentralized deterministic code design was introduced by Fragouli and Soljanin in [24]. Minimal configurations and the brute force algorithm to identify them were also introduced in [24]. Improved algorithms for identifying minimal configurations were constructed in [34].

# 6

---

## Networks with Delay and Cycles

---

For most of this review we have assumed that all nodes in the network simultaneously receive all their inputs and produce their outputs, and that networks have no cycles. We will now relax these assumptions. We first look at how to deal with delay over acyclic graphs. We then formally define networks with cycles, and discuss networks that have both cycles and delay. In fact, we will see that one method to deal with cycles is by artificially introducing delay.

### 6.1 Dealing with Delay

Network links typically introduce variable delay which may cause problems of synchronization in the network code operations. There are two known ways to deal with networks with delay: the asynchronous approach, briefly discussed in Chapter 5, does not allocate to network nodes predetermined operations, but instead, divides information packets into generations, and appends a “generation identification tag” to the packet. Intermediate network nodes store the received packets of the same generation, opportunistically combine them, and append the associated coding vector to the outgoing packets. Here network nodes

only need to know at which rate to inject coded packets into their outgoing links. This approach is ideally suited to dynamically changing networks, where large variations in delay may occur.

In contrast, the approach discussed in this chapter deals with the lack of synchronization by using time-windows at intermediate network nodes to absorb the variability of the network links delay. This second approach is better suited for networks with small variability in delay. Intermediate nodes wait for a predetermined time interval (larger than the expected delay of the network links) to receive incoming packets before combining them. Under this scenario, we can model the network operation by associating one unit delay  $\mathcal{D}$  (or more) either with each edge or only with edges that are coding points. We next discuss how to design network codes while taking into account the introduced (fixed) delay.

---

**Example 6.1.** Consider the butterfly network in Figure 1.2 and assume that there is a unit delay associated with every edge of the graph. Let  $\sigma_1(t)$  and  $\sigma_2(t)$  denote the bits transmitted by the source at time  $t$ ,  $t \geq 1$ . Assume that node  $C$  still simply **xors** its incoming information bits. Then through edge  $AD$ , the receiver  $R_1$  gets the sequence of bits

$$\{*, \sigma_1(1), \sigma_1(2), \sigma_1(3), \sigma_1(4), \dots\} = \sum_{t=1} \mathcal{D}^{t+1} \sigma_1(t),$$

and through edge  $ED$ , the sequence of bits

$$\{*, *, *, \sigma_1(1) + \sigma_2(1), \sigma_1(2) + \sigma_2(2) + \dots\} = \sum_{t=1} \mathcal{D}^{t+3} (\sigma_1(t) + \sigma_2(t)),$$

where “\*” means that nothing was received through the edge during that time slot. Equivalently, receiver  $R_1$  observes sequences  $y_1(\mathcal{D})$  and  $y_2(\mathcal{D})$  given as

$$\begin{bmatrix} y_1(\mathcal{D}) \\ y_2(\mathcal{D}) \end{bmatrix} = \underbrace{\begin{bmatrix} \mathcal{D} & 0 \\ \mathcal{D}^3 & \mathcal{D}^3 \end{bmatrix}}_{\mathbf{A}_1} \begin{bmatrix} \sigma_1(\mathcal{D}) \\ \sigma_2(\mathcal{D}) \end{bmatrix},$$

where  $\sigma_1(\mathcal{D}) = \sum_t \sigma_1(t) \mathcal{D}^t$  and  $\sigma_2(\mathcal{D}) = \sum_t \sigma_2(t) \mathcal{D}^t$ , and we use  $\mathbf{A}_1$  to denote the transfer matrix. For acyclic graphs, the elements of the

transfer matrix are polynomials in  $\mathcal{D}$ . Receiver  $R_1$  can decode at time  $t + 3$  the symbols  $\sigma_1(t)$  and  $\sigma_2(t)$ . Namely, it can process its received vectors by multiplying with the inverse transfer matrix  $\mathbf{B}_1$ .

$$\underbrace{\begin{bmatrix} \mathcal{D}^2 & 0 \\ \mathcal{D}^2 & 1 \end{bmatrix}}_{\mathbf{B}_1} \begin{bmatrix} y_1(\mathcal{D}) \\ y_2(\mathcal{D}) \end{bmatrix} = \mathcal{D}^3 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma_1(\mathcal{D}) \\ \sigma_2(\mathcal{D}) \end{bmatrix}.$$

In practice, fixed delay only means that instead of dealing with matrices and vectors whose elements belong to  $\mathbb{F}_q$ , we now have to deal with those whose elements are polynomials in the delay operator  $\mathcal{D}$  with coefficients over  $\mathbb{F}_q$ , i.e., belong in the ring of polynomials  $\mathbb{F}_q[\mathcal{D}] = \{\sum_i \alpha_i \mathcal{D}^i, \alpha_i \in \mathbb{F}_q\}$ . More generally, as we will discuss in the section about cycles, we will be dealing with vector spaces over the field of rational functions of polynomials in  $\mathbb{F}_q[\mathcal{D}]$ , that is, elements of  $\mathbb{F}_q(\mathcal{D}) = \{\frac{a(\mathcal{D})}{b(\mathcal{D})}, a(\mathcal{D}), b(\mathcal{D}) \in \mathbb{F}_q[\mathcal{D}], b(\mathcal{D}) \neq 0\}$ . In this framework, we can still talk about full rank and invertible matrices, where a matrix  $\mathbf{B}$  is the inverse of a matrix  $\mathbf{A}$  if  $\mathbf{AB} = \mathbf{BA} = \mathbf{I}$ . The goal of network code design is to select coding vectors so that each receiver  $R_j$  observes an invertible transfer matrix  $\mathbf{A}_j$ . It is easy to see that the main theorem in network coding (Theorem 2.2) still holds. Indeed, we can follow a very similar proof, merely replacing elements of  $\mathbb{F}_q$  with elements of  $\mathbb{F}_q(\mathcal{D})$ .

### 6.1.1 Algorithms

There are basically two ways to approach network code design for networks for delay:

- (1) Coding vectors have elements over  $\mathbb{F}_q$ . That is, intermediate nodes are only allowed to linearly combine over  $\mathbb{F}_q$  the incoming information streams, but do not incur additional delay. The delay is introduced only by the network links.
- (2) Coding vectors have elements over  $\mathbb{F}_q[\mathcal{D}]$  or  $\mathbb{F}_q(\mathcal{D})$ . That is, intermediate nodes may store their incoming information streams, and transmit linear combinations of their past and current inputs, i.e., introduce additional delay.

The benefits we may get in this case as compared to the previous approach, is operation over a small field  $\mathbb{F}_q$ , such as the binary field.

In both cases, all algorithms described in Chapter 5 for network code design can effectively still be applied. For example, consider the LIF algorithm in Section 5.2.1. This algorithm sequentially visits the coding points in a topological order and assigns coding vectors to them. Each assigned coding vector is such that all downstream receivers can still decode  $h$  source symbols produced during the same time-slot. In the case of networks with delay, it is sufficient to select coding vectors so that all downstream receivers get  $h$  linear independent combinations that allow them to decode  $h$  information symbols, one from each source, but where the  $h$  symbols are not necessarily produced during the same time slot.

### 6.1.2 Connection with Convolutional Codes

If we associate delay with the network links, we can think of the network as a convolutional code where memory elements are connected as dictated by the network topology, inputs are connected at the sources and outputs are observed at the receivers. This code is described by the finite-dimensional state-space equations (3.2) and the transfer matrix (3.3).

If we associate delay only the coding points in the graph, we obtain a convolutional code corresponding to the subtree graph of the network. An example subtree configuration is shown in Figure 6.1(a) and its corresponding convolutional code in Figure 6.1(b).

Convolutional codes are decoded using a trellis and trellis decoding algorithms, such as the Viterbi algorithm. Given that the receivers observe the convolutional code outputs with no noise, in trellis decoding we can proceed step by step, at each step decoding  $h$  information symbols and keeping track of one state only. In trellis decoding terminology, the decoding trace-back depth is one. Thus the complexity of decoding is determined by the complexity of the trellis diagram, namely, the number of states and the way they are connected.

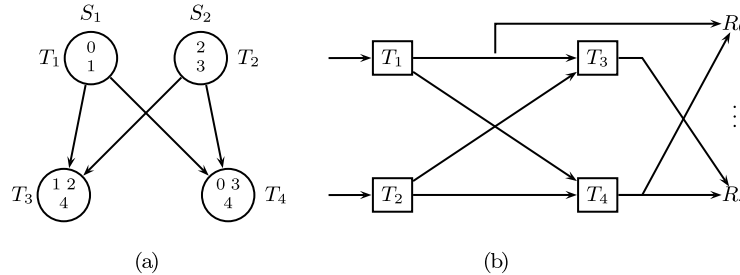


Fig. 6.1 Configuration with 2 sources and 5 receivers: (a) the subtree graph; (b) the corresponding convolutional encoder.

One way to reduce the decoder complexity is to identify, among all encoders that satisfy the multicast condition and the constraints of a given topology, the one that has the smallest number of memory elements, and thus the smallest number of states. Note that the minimization does not need to preserve the same set of outputs, as we are not interested in error-correcting properties, but only the min-cut condition for each receiver.

Another way to reduce the decoder complexity is to use for decoding the trellis associated with the minimal strictly equivalent encoder to  $\mathbf{G}_i(\mathcal{D})$ . Two codes are *strictly equivalent* if they have the same mapping of input sequences to output sequences. Among all strictly equivalent encoders, that produce the same mapping of input to output sequences, the encoder that uses the smallest number of memory elements is called *minimal*.

Typically for convolutional encoders, we are interested in equivalent encoders, that produce the same set of output sequences. Only recently, with the emergence of turbo codes where the mapping from input to output sequences affects the code's performance, the notion of strictly equivalent encoders has become important. Here we provide another example where this notion is useful. Note that in the conventional use of convolutional codes, there is no need to use a different encoder to encode and decode. In our case, we are restricted by the network configuration for the choice of the encoder  $\mathbf{G}_i(\mathcal{D})$ . However, given these constraints, we still have some freedom at the receiver to optimize for decoding complexity.

## 6.2 Optimizing for Delay

Consider the following scenario. A source needs to multicast  $h$  information bits (or packets),  $\{b_1, b_2, \dots, b_h\}$ , to a set of  $N$  receivers with as low delay as possible over a network with a delay associated with traversing every link of the network. For multimedia applications, a delay measure of interest is the inter-arrival delay. Define delay  $\delta_{ij}$  to be the number of time slots between the moment when receiver  $R_j$  successfully decodes bit  $b_{i-1}$  to the moment when it successfully decodes bit  $b_i$ . We are interested in identifying the optimal routing and network coding strategy, such that the overall *average inter-arrival delay*

$$\frac{\sum_j \sum_i \delta_{ij}}{Nh}$$

is minimized. Note that use of network coding may allow us to achieve higher information rates, thus reducing the delay. On the other hand, use of network coding may increase the delay, because a receiver may need to wait for the arrival of several coded bits before decoding.

As the following example illustrates, either routing or network coding or a combination of the two may enable us to achieve the optimal delay when multicasting. But neither routing nor network coding may allow us to achieve the delay that a receiver would experience were it to use all the network resources by itself. When in Part II we discuss network coding applications to wireless, we will see other examples showing how use of network coding may enable delay reduction.

---

**Example 6.2.** Consider the butterfly network in Figure 1.2 and assume that the source would like to deliver  $m$  bits to both receivers. Assume that edges  $AD$  and  $BF$  have unit delay, while edge  $CE$  has an associated delay of  $\alpha$  delay units.

Then with network coding, the average interarrival delay equals<sup>1</sup>

$$\frac{2(\alpha - 1) + \frac{m-2(\alpha-1)}{2}}{m} = \frac{1}{2} + \frac{\alpha - 1}{m}$$

---

<sup>1</sup>For simplicity, we ignore the fact that the network coding solution does not preserve the ordering of the bits.

timeslots. Indeed, for the first  $\alpha - 1$  timeslots each receiver gets one uncoded bit through  $AD$  and  $BF$ , respectively, then for  $(m - 2(\alpha - 1))/2$  timeslots each receiver can successfully decode two bits per timeslot, and finally for  $\alpha - 1$  timeslots the receivers only receive and decode one bit per timeslot through  $CE$ .

On the other hand, by using routing along  $AD$  and  $BF$  and time-sharing along  $CE$ , the source can deliver the information in

$$\frac{\alpha - 1 + \frac{2(m-\alpha+1)}{3}}{m} = \frac{2}{3} + \frac{1}{3} \frac{\alpha - 1}{m}$$

timeslots. Depending on the value of  $\alpha$  and  $m$ , the network coding solution might lead to either less or more delay than the routing solution.

Finally, if a receiver exclusively uses the network resources, the resulting delay would be

$$\frac{1}{2} + \frac{1}{2} \frac{\alpha - 1}{m}$$

that outperforms both previous approaches.

---

### 6.3 Dealing with Cycles

In this section, we consider network topologies that have cycles. We distinguish two cases, depending on whether a partial order constraint, which we describe below, is satisfied. The problem of cycles has to be addressed only for networks which do not satisfy this constraint.

We observe that each path  $(S_i, R_j)$  from source  $S_i$  to receiver  $R_j$  induces a partial order on the set of edges: if edge  $a$  is a child of edge  $b$  then we say that  $a < b$ . The source node edge is the maximal element. Each different path imposes a different partial order on the same set of edges. We distinguish the graphs depending on whether the partial orders imposed by the different paths are consistent. Consistency implies that for all pairs of edges  $a$  and  $b$ , if  $a < b$  in some path, there does not exist a path where  $b < a$ . A sufficient, but not necessary, condition for consistency is that the underlying graph  $G$  is acyclic.

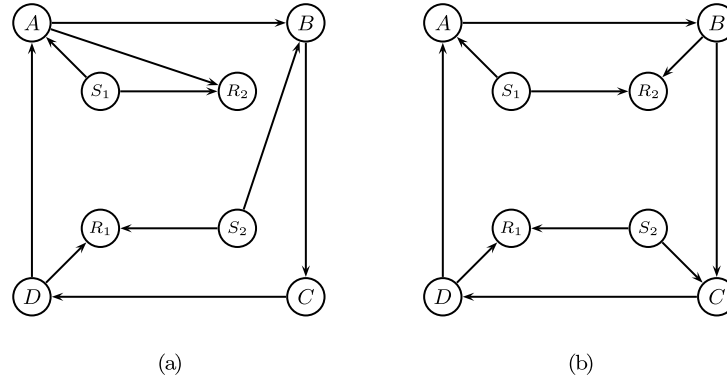


Fig. 6.2 Two networks with a cycle  $ABCD$ : (a) paths  $(S_1, R_1) = S_1A \rightarrow AB \rightarrow BC \rightarrow CD \rightarrow DR_1$  and  $(S_2, R_2) = S_2B \rightarrow BC \rightarrow CD \rightarrow DA \rightarrow AR_2$  impose consistent partial orders to the edges of the cycle; (b) paths  $(S_1, R_1) = S_1A \rightarrow AB \rightarrow BC \rightarrow CD \rightarrow DR_1$  and  $(S_2, R_2) = S_2C \rightarrow CD \rightarrow DA \rightarrow AB \rightarrow BR_2$  impose inconsistent partial orders to the edges of the cycle.

---

**Example 6.3.** Consider the two networks in Figure 6.2. Sources  $S_1$  and  $S_2$  use the cycle  $ABCD$  to transmit information to receivers  $R_1$  and  $R_2$ , respectively. In the network shown in Figure 6.2(a), the paths from sources  $S_1$  and  $S_2$  impose consistent partial orders on the edges of the cycle. In the network shown in Figure 6.2(a), for the path from source  $S_1$ , we have  $AB > CD$ , whereas for the path from source  $S_2$ , we have  $CD > AB$ .

---

When the partial orders imposed by the different paths are consistent, although the underlying network topology contains cycles, the line graph associated with the information flows does not. Thus, from the network code design point of view, we can treat the network as acyclic. When the partial orders imposed by the different paths are not consistent, then the network code design needs to be changed accordingly. It is this set of networks we call networks with cycles.

### 6.3.1 Delay and Causality

Our first observation is that, in networks with cycles, we need to introduce delay in each cycle to guarantee consistent and causal information propagation around the cycle, as the following example illustrates.

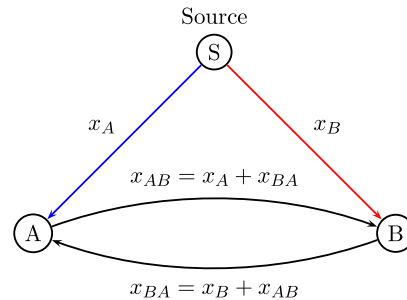


Fig. 6.3 A networks with a cycle  $ABA$ . Unless we introduce delay in the cycle, the information flow is not well defined.

---

**Example 6.4.** Consider nodes  $A$  and  $B$  that insert the independent information flows  $x_A$  and  $x_B$  in the two-edge cycle  $(A, B, A)$ , as depicted in Figure 6.3. Consider the network coding operation at node  $A$ . If we assume that all nodes instantaneously receive messages from incoming links and send them to their outgoing links, then node  $A$  simultaneously receives  $x_A$  and  $x_{BA}$  and sends  $x_{AB}$ . Similarly for node  $B$ . This gives rise to the equation

$$x_{AB} = x_A + x_{BA} = x_A + x_B + x_{AB} \Rightarrow x_A + x_B = 0,$$

which is not consistent with the assumption that  $x_A$  and  $x_B$  are independent. It is also easy to see that local and global coding vectors are no longer well defined.

---

### 6.3.2 Code Design

There are two ways to approach designing network codes for networks with cycles. The first approach observes that networks with cycles correspond to systems with feedback, or, in the convolutional code framework, recursive convolutional codes. That is, the transfer matrices are rational functions of polynomials in the delay operator. The convolutional code framework enables us to naturally take into account cycles without changing the network code design: we select coding operations so that each receiver observes a full rank transfer matrix. Note

that decoding can still be performed using the trellis diagram of the recursive convolutional code, a trellis decoding algorithm like Viterbi, and trace-back depth of one. The decoding complexity mainly depends on the size of the trellis, exactly as in the case of feed-forward convolutional codes. As a result, cycles per se do not increase the number of states and the complexity of trellis decoding.

The second network code design approach attempts to remove the cycles. This approach applies easily to networks that have *simple cycles*, i.e., cycles that do not share edges with other cycles. Observe that an information source needs to be transmitted through the edges of a cycle at most once, and afterwards can be removed from the circulation by the node that introduced it. We illustrate this approach through the following example.

---

**Example 6.5.** Consider the cycle in Figure 6.2(b), and for simplicity assume that each edge corresponds to one memory element. Then the flows through the edges of the cycle are

$$\begin{aligned}
 AB: & \sigma_1(t-1) + \sigma_2(t-3) \\
 BC: & \sigma_1(t-2) + \sigma_2(t-4) \\
 CD: & \sigma_1(t-3) + \sigma_2(t-1) \\
 DA: & \sigma_1(t-4) + \sigma_2(t-2),
 \end{aligned} \tag{6.1}$$

where  $\sigma_i(t)$  is the symbol transmitted from source  $S_i$  at time  $t$ . Operations in (6.1) can be easily implemented by employing a block of memory elements as shown in Figure 6.4. Thus, we can still have a feed-forward encoder by representing the cycle with a block of memory elements.

---

The approach in Example 6.5 generalizes to arbitrary graphs with simple cycles.

A straightforward algorithm is to start with the graph  $\gamma = \bigcup(S_i, R_j)$  (or the subtree graph) where vertices correspond to edges in the original graph  $G$ , and treat each cycle separately. Every vertex  $k$  in  $\gamma$  (edge in the original graph) corresponds to a state variable  $\mathbf{s}_k$ . We need to express how the information that goes through each  $\mathbf{s}_k$  evolves with

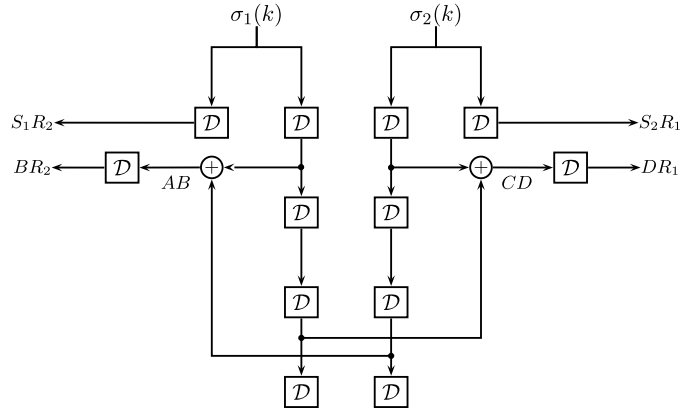


Fig. 6.4 Block representation of the cycle in Figure 6.2(b).

time. To do that, we can look at the  $I$  inputs (incoming edges) of the cycle (for example in Figure 6.4, we have  $I = 2$  inputs). Each input follows a path of length  $L_i$  (in Figure 6.4,  $L_1 = L_2 = 3$ ). For each input, we create  $L_i$  memory elements. Through each edge of the cycle flows a linear combination of a subset of these  $\sum L_i$  memory elements. This subset is defined by the structure of the paths and the structure of the cycle, i.e., it is not selected by the code designer. The code designer can only select the coefficients for the linear combinations. Using this approach, we create an expanded matrix  $\mathbf{A}$ , that contains, for every cycle, an additional set of  $\sum L_i$  memory elements. For example, in Figure 6.4, we use six additional memory elements, thus obtaining a convolutional code with a total of 12 memory elements. We can think of this approach as “expanding in time” when necessary, and along specific paths.

This approach does not extend to networks with overlapping cycles, as the following example illustrates.

---

**Example 6.6.** Consider the network depicted in Figure 6.5 that has a *knot*, that is, an edge (in this case  $r_2r_3$ ) shared by multiple cycles. There are three receivers  $t_1, t_2$ , and  $t_3$  and four sources  $A, B, C$ , and  $D$ . All four information streams  $\sigma_A, \sigma_B, \sigma_C$ , and  $\sigma_D$  have to flow through the edge  $r_2r_3$ . Consider flow  $x_A$ : to reach receiver  $t_1$  it has to circulate

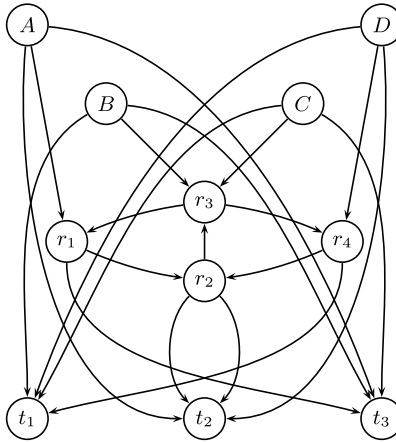


Fig. 6.5 A network configuration with multiple cycles sharing the common edge  $r_2r_3$  (knot).

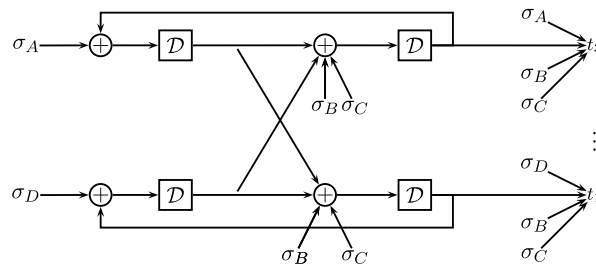


Fig. 6.6 Convolutional code corresponding to the network configuration in Figure 6.5 if we associate unit delay with the edges  $r_1r_2$ ,  $r_4r_2$ ,  $r_3r_1$ , and  $r_3r_4$ .

inside the cycle  $(r_2, r_3, r_4)$ . Since none of the nodes  $r_2$ ,  $r_3$ , and  $r_4$  has  $\sigma_A$ ,  $\sigma_A$  cannot be removed from the cycle. Thus, the second approach would not work. To apply the first approach, assume we associate a unit delay element with each of the edges  $r_1r_2$ ,  $r_4r_2$ ,  $r_3r_1$ , and  $r_3r_4$ . The resulting convolutional code is depicted in Figure 6.6.

### Notes

Coding in networks with cycles and delay was first considered in the seminal papers on network coding by Ahlswede *et al.* [2, 36]. Using polynomials and rational functions over the delay operator  $\mathcal{D}$  to take delay

into account was proposed, together with the algebraic approach, by Koetter and Médard in [32]. The connection with convolutional codes was observed by Fragouli and Soljanin in [23] and independently by Erez and Feder in [21]. Using a partial order to distinguish between cyclic and acyclic graphs was proposed by Fragouli and Soljanin in [24] and later extended by Barbero and Ytrehus in [7]. The “knot example” comes from [7]. Methods to code over cyclic networks were also examined by Ho *et al.* in [27]. A very nice tutorial on convolutional codes is [39].

# 7

---

## Resources for Network Coding

---

Now that we have learned how to design network codes and how much of throughput increase to expect in networks using network coding, it is natural to ask how much it costs to operate such networks. We focus our discussion on resources required to linear network coding for multicasting, as this is the better understood case today. We can distinguish the complexity of deploying network coding to the following components:

- (1) *Set-up complexity*: This is the complexity of designing the network coding scheme, which includes selecting the paths through which the information flows, and determining the operations that the nodes of the network perform. In a time-invariant network, this phase occurs only once, while in a time-varying configuration, the complexity of adapting to possible changes becomes important. Coding schemes for network coding and their associated design complexity are discussed in Chapter 5.
- (2) *Operational complexity*: This is the running cost of using network coding, that is, the amount of computational and

network resources required per information unit successfully delivered. Again this complexity is strongly correlated to the employed network coding scheme.

In linear network coding, we look at information streams as sequences of elements of some finite field  $\mathbb{F}_q$ , and coding nodes linearly combine these sequences over  $\mathbb{F}_q$ . To recover the source symbols, each receiver needs to solve a system of  $h \times h$  linear equations, which requires  $\mathcal{O}(h^3)$  operations over  $\mathbb{F}_q$  if Gaussian elimination is used. Linearly combination of  $h$  information streams requires  $\mathcal{O}(h^2)$  finite field operations. The complexity is further affected by

- (a) *The size of the finite field* over which we operate. We call this the *alphabet size* of the code. The cost of finite field arithmetics grows with the field size. For example, typical algorithms for multiplications/inversions over a field of size  $q = 2^n$  require  $\mathcal{O}(n^2)$  binary operations, while the faster implemented algorithms (using the Karatsuba method for multiplication) require  $\mathcal{O}(n^{\log 3}) = \mathcal{O}(n^{1.59})$  binary operations. Moreover, the alphabet size also affects the required storage capabilities at intermediate nodes of the network. We discuss alphabet size bounds in Section 7.1.
- (b) *The number of coding points* in the network where we need to perform the encoding operations. Note that network nodes capable of combining incoming information streams are naturally more expensive than those merely capable of duplicating and forwarding incoming packets, and can possibly cause delay. Therefore, from this point of view as well, it is in our interest to minimize the number of coding points. We look into this issue in Section 7.2.

There are also costs pertinent specifically to wireless networks, which we discuss in the chapter on wireless networks, in part II of the review.

## 7.1 Bounds on Code Alphabet Size

We are here interested in the maximum alphabet size required for a multicast instance  $\{G, S, \mathcal{R}\}$  with  $h$  sources and  $N$  receivers. That is, the alphabet size that is sufficient but not necessary for all networks with  $h$  sources and  $N$  receivers. Recall that the binary alphabet is sufficient for networks which require only routing.

### 7.1.1 Networks with Two Sources and $N$ Receivers

To show that an alphabet of size  $q$  is sufficient, we can equivalently prove that we can construct a valid code by using as coding vectors the  $k = q + 1$  different vectors over  $\mathbb{F}_q^2$  in the set

$$\{[1\ 0], [0\ 1], [1\ \alpha], \dots, [1\ \alpha^{q-1}]\}, \quad (7.1)$$

where  $\alpha$  is a primitive element of  $\mathbb{F}_q$ . Any two such coding vectors form a basis of the two-dimensional space (see Example A.2 in Appendix).

For the rest of this section, we use the combinatorial framework of Section 3.3. We restrict our attention to minimal subtree graphs, since a valid network code for a minimal subtree graph  $\Gamma$  directly translates to a valid network code for any subtree graph  $\Gamma'$  that can be reduced to  $\Gamma$  (the code for  $\Gamma$  simply does not use some edges of  $\Gamma'$ ). Thus, an alphabet size sufficient for all possible minimal subtree graphs is sufficient for all possible non-minimal subtree graphs as well.

Let  $\Gamma$  be a minimal subtree graph with  $n > 2$  vertices (subtrees);  $n - 2$  is the number of coding subtrees (note that when  $n = 2$ ,  $\Gamma$  has only source subtrees and no network coding is required). We relate the problem of assigning vectors to the vertices of  $\Gamma$  to the problem of vertex coloring a suitably defined graph  $\Omega$ . Vertex coloring is an assignment of colors to the vertices of a graph so that adjacent vertices have different colors. In our case, the “colors” are the coding vectors, which we can select from the set (7.1), and the graph to color  $\Omega$  is a graph with  $n$  vertices, each vertex corresponding to a different subtree in  $\Gamma$ . We connect two vertices in  $\Omega$  with an edge when the corresponding subtrees cannot be allocated the same coding vector.

If two subtrees have a common receiver node, they cannot be assigned the same coding vector. Thus, we connect the corresponding

vertices in  $\Omega$  with an edge which we call *receiver edge*. Similarly, if two subtrees have a common child, by Theorem 3.5, they cannot be assigned the same coding vector. We connect the corresponding vertices in  $\Omega$  with an edge which we call a *flow edge*. These are the only independence conditions we need to impose: all other conditions follow from these and minimality. For example, by Theorem 3.5, parent and child subtrees cannot be assigned the same coding vector. However, we need not worry about this case separately since by Theorem 3.6, a parent and a child subtree have either a child or a receiver in common. Figure 7.1 plots  $\Omega$  for an example subtree graph (which we considered in Chapter 3).

---

**Lemma 7.1.** For a minimal configuration with  $n > 2$ , every vertex  $i$  in  $\Omega$  has degree at least 2.

---

*Proof.* We prove the claim separately for source and coding subtrees.

- (1) *Source subtrees:* If  $n = 3$ , the two source subtrees have exactly one child which shares a receiver with each parent. If  $n > 3$ , the two source subtrees have at least one child which shares a receiver or a child with each parent.
- (2) *Coding subtrees:* Each coding subtree has two parents. Since the configuration is minimal, it cannot be allocated the same

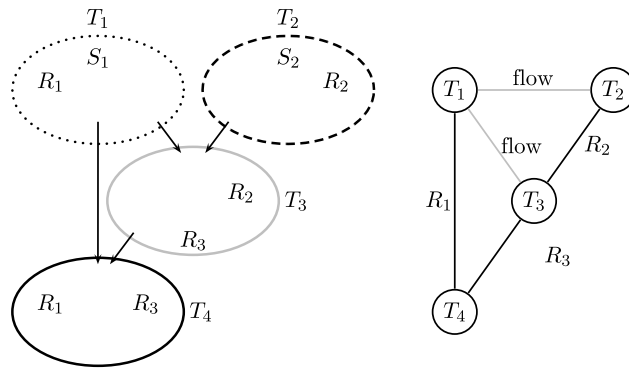


Fig. 7.1 A subtree graph  $\Gamma$  and its associated graph  $\Omega$ . The receiver edges in  $\Omega$  are labeled by the corresponding receivers.

coding vector as either of its parents. This implies that in  $\Omega$  there should exist edges between a subtree and its parents, that may be either flow edges, or receiver edges, and the corresponding vertex has degree at least two.  $\square$

---

**Definition 7.1.** The *chromatic number* of a graph is the minimum number of colors required to color the vertices of the graph, so that no two adjacent vertices are assigned the same color. A graph is *k-chromatic* if its chromatic number is exactly  $k$ .

---



---

**Lemma 7.2.** ([9, Ch. 9]) Every  $k$ -chromatic graph has at least  $k$  vertices of degree at least  $k - 1$ .

---



---

**Theorem 7.3.** For any minimal configuration with  $N$  receivers, the code alphabet  $\mathbb{F}_q$  of size

$$\lfloor \sqrt{2N - 7/4} + 1/2 \rfloor$$

is sufficient. There exist configurations for which it is necessary.

---

*Proof.* Assume that our graph  $\Omega$  has  $n$  vertices and chromatic number  $\chi(\Omega) = k \leq n$ . Let  $m - k$ , where  $m$  is a nonnegative integer. We are going to count the number of edges in  $\Omega$  in two different ways:

- (1) From Lemmas 7.1 and 7.2, we know that each vertex has degree at least 2, and at least  $k$  vertices have degree at least  $k - 1$ . Consequently, we can lower bound the number of edges of  $\Omega$  as

$$E(\Omega) \geq \frac{k(k - 1) + 2m}{2}. \quad (7.2)$$

- (2) Since there are  $N$  receivers and  $n - 2$  coding subtrees, we have at most  $N$  receiver edges and at most  $n - 2$  distinct flow edges (we count parallel edges only once). Thus,

$$E(\Omega) \leq N + n - 2 = N + k + m - 2. \quad (7.3)$$

From (7.2) and (7.3), we obtain

$$N \geq \frac{k(k-1)}{2} - k + 2. \quad (7.4)$$

Equation (7.4) provides a lower bound on the number of receivers we need in order to have chromatic number  $k$ . Solving for  $q = k - 1$  we get the bound

$$q \leq \lfloor \sqrt{2N - 7/4} + 1/2 \rfloor.$$

This proves the first claim of the theorem that, for any minimal configuration with  $N$  receivers, an alphabet of size  $\lfloor \sqrt{2N - 7/4} + 1/2 \rfloor$  is sufficient.

To show that there exist configurations for which an alphabet of this size is necessary, we are going to construct a subtree graph where (7.4) becomes equality, i.e., we will construct a subtree graph that has  $N = \frac{k(k-1)}{2} - k + 2$  receivers and the corresponding graph  $\Omega$  has chromatic number  $k$ . We start with a minimal subtree graph  $\Gamma$  that has  $k$  vertices and  $k - 1$  receivers. Such a graph can be constructed as depicted in Figure 7.2. The corresponding graph  $\Omega$  has  $k - 2$  flow edges and  $k - 1$  receiver edges. Add  $\frac{k(k-1)}{2} - [(k-2) + (k-1)]$  receivers, so that  $\Omega$  becomes a complete graph with  $E(\Omega) = \frac{k(k-1)}{2}$  edges. Thus  $\Omega$  cannot be colored with less than  $k$  colors. The corresponding subtree graph has  $N = \frac{k(k-1)}{2} - k + 2$  receivers, and requires an alphabet of size  $q = k - 1$ .  $\square$

### 7.1.2 Networks with $h$ Sources and $N$ Receivers

Based on the results of the previous section, a lower bound on the alphabet size a network with  $h$  sources and  $N$  receivers may require is  $\lfloor \sqrt{2N - 7/4} + 1/2 \rfloor$ , since such a network may contain a two-source subnetwork that requires this alphabet size. The interesting question is whether there are networks that require larger alphabet.

---

**Theorem 7.4.** For all configurations with  $h$  sources and  $N$  receivers, an alphabet of size greater than  $N$  is always sufficient.

---

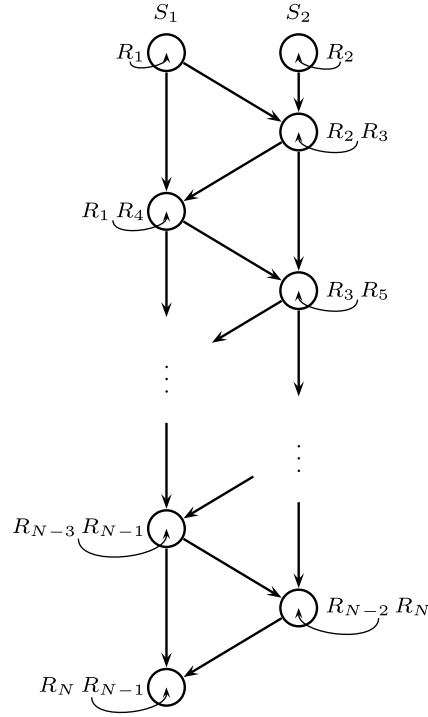


Fig. 7.2 A minimal subtree graph for a network with two sources,  $N$  receivers, and  $N - 1$  coding subtrees.

We have already given two different proofs of this result, in Theorem 3.2 and Lemma 5.2. Here we discuss whether this bound is tight or not.

The proof of Theorem 3.2 does not use the fact that we only care about minimal configurations, where paths cannot overlap more than a certain number of times, and thus the entries of the matrices  $\mathbf{A}_j$  are not arbitrary. For example, there does not exist a minimal configuration with two sources and two receivers corresponding to the matrices

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 0 \\ 0 & x \end{bmatrix}, \quad \mathbf{A}_2 = \begin{bmatrix} 1 & 1 \\ 1 & x \end{bmatrix}. \quad (7.5)$$

In fact, as we saw in Theorem 3.5, there exist two minimal configuration with two sources and two receivers. Therefore, the proof of Lemma 3.1 is more general than it needs to be, and does not necessarily give a tight bound. Similarly, recall that in the proof of Lemma 5.2, we overesti-

mated the number of vectors that cannot be used at a certain coding point. Therefore, this lemma may not give a tight alphabet size either.

In fact, up to today, we do not have examples of networks where the alphabet size needs to be larger than  $\mathcal{O}(\sqrt{N})$ . It has also been shown that this alphabet size is sufficient, under some regularity conditions or special network structure. The conjecture that this is indeed the sufficient alphabet size for all networks is as far as we know today open.

### 7.1.3 Complexity

It is well known that the problem of finding the chromatic number of a graph is NP-hard. We saw in Section 7.1.1 that the problem of finding the minimum alphabet size can be reduced to the chromatic number problem. Here we show that the reverse reduction also holds: we can reduce the problem of coloring an arbitrary graph  $G$  with the minimum number of colors, to the problem of finding a valid network code with the smallest alphabet size, for an appropriately chosen network (more precisely, an appropriately chosen subtree graph  $\Gamma$ , that corresponds to a family of networks). This instance will have two sources, and the coding vectors (corresponding to colors) will be the vectors in (7.1).

Let  $G = (V, E)$  be the graph whose chromatic number we are seeking. Create a subtree graph  $\Gamma = (V_\gamma = V, E_\gamma)$ , that has as subtrees the vertices  $V$  of  $G$ . We will first select the subtrees (vertices) in  $\Gamma$  that act as source subtrees. Select an edge  $e \in E$  that connects vertices  $v_1$  and  $v_2$ , with  $v_1, v_2 \in V$ . Let  $v_1$  and  $v_2$  in  $\Gamma$  act as source subtrees, and the remaining vertices as coding subtrees, that is,  $E_\gamma = \{(v_1, v), (v_2, v) | v \in V \setminus \{v_1, v_2\}\}$ . For each  $e = (v_1, v_2) \in E$ , create a receiver that observes  $v_1$  and  $v_2$  in  $\Gamma$ . It is clear that finding a coding scheme for  $\Gamma$  is equivalent to coloring  $G$ . We thus conclude that finding the minimum alphabet size is also an NP-hard problem.

## 7.2 Bounds on the Number of Coding Points

As we discussed in Example 3.1, non-minimal configurations can have a number of coding points that grows with the number of edges in the network. In minimal configurations, this is no longer the case. Intuitively,

if we have  $h$  sources and  $N$  receivers, the paths from the sources to the receivers can only intersect in a fixed number of ways. The bounds we give below for minimal configurations depend only on  $h$  and  $N$ .

The problem of finding the minimum number of coding points is NP-hard for the majority of cases. However, it is polynomial time provided that  $h = \mathcal{O}(\infty)$ ,  $N = \mathcal{O}(\infty)$ , and the underlying graph is acyclic.

### 7.2.1 Networks with Two Sources and $N$ Receivers

For networks with two sources, we can calculate a tight upper bound on the number of coding points using combinatorial tools.

---

**Theorem 7.5.** In a minimal subtree decomposition of a network with two sources and  $N$  receivers, the number of coding subtrees is upper-bounded by  $N - 1$ . There exist networks which have a minimal subtree decomposition that achieves this upper bound.

---

*Proof.* Recall that there are exactly  $2N$  receiver nodes. The first part of the claim then follows directly from Theorem 3.6. Figure 7.2 demonstrates a minimal subtree graph for a network with two sources and  $N$  receivers that achieves the upper bound on the maximum number of subtrees.  $\square$

### 7.2.2 Networks with $h$ Sources and $N$ Receivers

Here we restrict our attention to acyclic networks, but similar bounds exist for cyclic networks.

---

**Theorem 7.6.** Consider a multicast instance  $\{G, S, \mathcal{R}\}$ . Then we can efficiently find a feasible network code with  $|\mathcal{C}| \leq h^3 N^2$ , where  $h$  is the number of sources,  $N$  the number of receivers, and  $|\mathcal{C}|$  denotes the number of coding points for the network code.

---

The proof idea of this theorem is to first examine configurations with  $h$  sources and two receivers (see Figure 7.3). For such minimal configurations, one can show that the number of coding points is  $\mathcal{O}(h^3)$ .

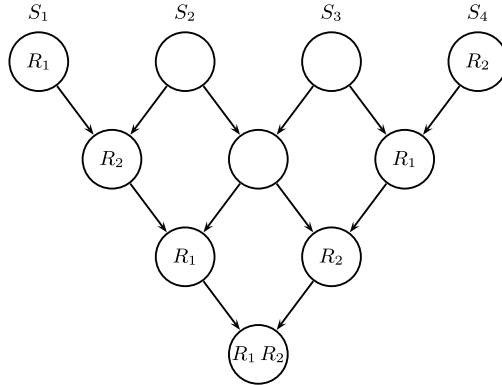


Fig. 7.3 Minimal configuration with  $h = 4$  sources and  $N = 2$  receivers.

Configurations with  $N$  receivers can be reduced to  $\mathcal{O}(N^2)$  configurations with two receivers as follows. Given a pair of receivers, simply remove the edges/coding points corresponding to the remaining  $N - 2$  receivers. The resulting configuration will have at most  $\mathcal{O}(h^3)$  coding points. Thus the total number of coding points cannot exceed  $\mathcal{O}(h^3 N^2)$ .

---

**Lemma 7.7.** There exist instances  $\{G, S, \mathcal{R}\}$  with  $h$  sources and  $N$  receivers such that  $|\mathcal{C}| \geq \Omega(h^2 N)$ .

---

We prove this lemma constructively. Consider the minimal subtree graph in Figure 7.3 with  $h = 4$  sources and two receivers. It is easy to see that this is a minimal configuration. The construction directly extends to the case of  $h$  sources with  $h(h - 1)/2$  coding points. We can extend this construction to configurations with  $N$  receivers by using  $N/2$  non-overlapping receiver pairs.

### 7.2.3 Complexity

The problem of minimizing the number of coding points is NP-hard for the majority of cases. It is polynomial time in the case where the number of sources and receivers is constant, and the underlying graph is acyclic. The following theorem deals with the case of integral network coding over cyclic graphs (graphs that are allowed to have cycles).

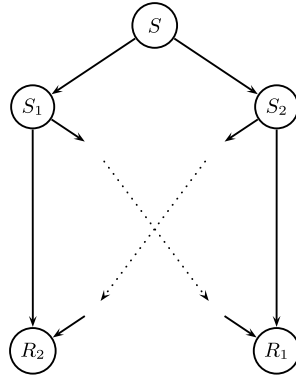


Fig. 7.4 Reduction between the minimum-number-of-coding-points and link-disjoint-paths problems.

---

**Theorem 7.8.** Consider integral network coding over cyclic graphs. Let  $\epsilon > 0$  be a constant. Finding the minimum number of coding points within any multiplicative factor or within an additive factor of  $|V|^{1-\epsilon}$  is NP-hard, where  $V$  is the number of vertices in the graph.

---

This can be proved by reduction to the link-disjoint path (LDP) problem in directed graphs. Here we give the main proof idea. Given a directed graph and nodes  $S_1$ ,  $S_2$ ,  $R_1$ , and  $R_2$ , the LDP problem asks to find two edge disjoint paths  $(S_1, R_1)$  and  $(S_2, R_2)$ . Connect  $S_1$  and  $S_2$  to a virtual common source  $S$ , and also directly connect  $S_2$  to  $R_1$  and  $S_1$  to  $R_2$  through virtual edges. Consider now the problem of  $h = 2$  sources multicasting from  $S$  to  $R_1$  and  $R_2$  using the minimum number of coding points. If this minimum number equals zero, we have identified a solution for the LDP problem. If it is greater than zero, the LDP problem has no solution. Since the LDP is known to be NP-complete, our original problem is at least equally hard.<sup>1</sup>

### 7.3 Coding with Limited Resources

In the previous sections we looked into network coding costs in terms of (i) the number of nodes required to perform coding (which are more

<sup>1</sup>We underline that this reduction applies for the case of integral routing.

expensive and may cause delay) and (ii) the required code alphabet size which determines the complexity of the electronic modules performing finite field arithmetics. We now examine whether we can quantify the trade-offs between alphabet size, number of coding points, throughput, and min-cut. Up to now we only have preliminary results toward these directions that apply for special configurations such as the following examples.

### 7.3.1 Min-Cut Alphabet-Size Trade-Off

We here consider networks with two sources where the processing complexity is a stronger constraint than the bandwidth. In particular, the system cannot support an alphabet size large enough to accommodate all users, but on the other hand, the min-cut toward each receiver is larger than the information rate that we would like to multicast.

When the min-cut toward each user is exactly equal to the number of sources, the bound in Theorem 7.3 gives the maximum alphabet size a network with  $N$  users may require. One would expect that, if the min-cut toward some or all of the receivers is greater than the number of sources, a smaller alphabet size would be sufficient. The intuition is that, if the min-cut to each receiver is  $m > h$ , and we only want to route  $h$  sources, we can “select” the  $h$  paths toward each receiver that lead to the smallest alphabet size requirements. Equivalently, if we have  $N$  receivers,  $h$  sources, and each receiver has mincut  $m \geq h$ , we have a choice of  $\binom{m}{h}^N$  possible configurations  $\{G, S, \mathcal{R}\}$  to select from.<sup>2</sup>

For the special case when the subtree graph is bipartite (which means that the parent of coding subtrees can only be source subtrees) and we have  $h = 2$  sources, we can show that this is indeed true by applying the following result. Consider a set of points  $X$  and a family  $\mathcal{F}$  of subsets of  $X$ . A coloring of the points in  $X$  is legal if no element of  $\mathcal{F}$  is monochromatic. If a family admits a legal coloring with  $k$  colors, then it is called  $k$ -colorable.

---

<sup>2</sup>Note however that several of these choices may turn out to correspond to the same minimal configuration.

---

**Theorem 7.9.** (Erdős 1963) Let  $\mathcal{F}$  be a family of sets each of size at least  $m$ . If  $|\mathcal{F}| < k^{m-1}$ , then  $\mathcal{F}$  is  $k$ -colorable.

---

In our case,  $X$  is the set of subtrees,  $m$  is the min-cut from the sources to each receiver, each element of  $\mathcal{F}$  corresponds to the set of  $m$  subtrees observed by a receiver, and the set of colors are the points on the projective line. Therefore,  $\mathcal{F}$  is a family of sets each of size  $m$ , and  $|\mathcal{F}| = N$ . Suppose that we can use an alphabet size  $k - 1$  (which gives  $k$  colors). Note that each receiver can observe both sources if  $\mathcal{F}$  is colorable, since that means that no set of subtrees observed by a receiver is monochromatic. From Theorem 7.9, this holds as long as

$$N < k^{m-1}.$$

The above inequality shows a trade-off between the min-cut  $m$  to each user and the alphabet size  $k - 1$  required to accommodate  $N$  receivers for the special class of configurations we examine. We expect a similar tradeoff in the case where the graph is not bipartite as well. However, Theorem 7.9 cannot be directly applied, because in this case there are additional constraints on coloring of the elements of  $X$  coming from the requirement that each child subtree has to be assigned a vector lying in the linear span of its parents' coding vectors.

### 7.3.2 Throughput Alphabet-Size Trade-Off

Consider again the case where the subtree graph is bipartite, we have  $h = 2$  sources, and the min-cut to each receiver is  $m$ . We are interested in the number of receivers which will not be able to decode both sources, when we are restricted to use an alphabet of size  $k - 1$ . We obtain a bound to this number by making use of the following result.

---

**Theorem 7.10.** For every family  $\mathcal{F}$  whose all members have size exactly  $m$ , there exists a  $k$ -coloring of its points that colors at most  $|\mathcal{F}|k^{1-m}$  of the sets of  $\mathcal{F}$  monochromatically.

---

Thus if we have  $|\mathcal{F}| = N$  receivers, the min-cut to each receiver is  $m$ , and we only employ routing, then at most  $Nk^{1-m}$  receivers will not

be able to decode both sources. In other words, if the alphabet size is not large enough to accommodate all users, but on the other hand, the min-cut toward each receiver is larger than the information rate that we would like to multicast, at least  $|F|(1 - k^{1-m})$  receivers will still be able to successfully decode both sources.

### 7.3.3 Design Complexity and Alphabet-Size Trade-off

Here we show that, if we use very low complexity decentralized network code design algorithms, we may also use an alphabet size much larger than the optimal code design might require.

We make this case for the specific class of  $\text{ZK}(p, N)$  networks that are described in Example 4.2. We will show that for the  $\text{ZK}(p, N)$  configurations there exist network codes over the binary alphabet. On the contrary, if we perform coding by random assignment of coding vectors over an alphabet  $\mathbb{F}_q$ , the probability  $P_N^d$  that all  $N$  receivers will be able to decode is bounded (Theorem 5.4) as

$$P_N^d \geq \left(1 - \frac{N}{q}\right)^{\binom{N}{p}} \approx e^{-N\binom{N}{p}/q}.$$

Thus, if we want this bound to be smaller than, say,  $e^{-1}$ , we need to choose  $q \geq N\binom{N}{p}$ .

For arbitrary values of  $p$  and  $N$ , network coding using a binary alphabet can be achieved as follows: We first remove the edges going out of  $S$  into those  $A$ -nodes whose labels contain  $N$ . There are  $\binom{N-1}{p-2}$  such edges. Since the number of edges going out of  $S$  into  $A$ -nodes is  $\binom{N}{p-1}$ , the number of remaining edges is  $\binom{N}{p-1} - \binom{N-1}{p-2} = \binom{N-1}{p-1}$ . We label these edges by the  $h = \binom{N-1}{p-1}$  different basis elements of  $\mathbb{F}_2^h$ . We further remove all  $A$ -nodes which have lost their connection with the source  $S$ , as well as their outgoing edges. The  $B$ -nodes merely sum their inputs over  $\mathbb{F}_2^h$ , and forward the result to the  $C$ -nodes.

Consider a  $C$ -node that the  $N$ th receiver is connected to. Its label, say  $\omega$ , is a  $p$ -element subset of  $\mathcal{I}$  containing  $N$ . Because of our edge removal, the only  $A$ -node that this  $C$ -node is connected to is the one with the label  $\omega \setminus \{N\}$ . Therefore, all  $C$ -nodes that the  $N$ th receiver is connected to have a single input, and all those inputs are different. Consequently, the  $N$ th receiver observes all the sources directly.

Each of the receivers  $1, 2, \dots, N - 1$  will have to solve a system of equations. Consider one of these receivers, say  $R_j$ . Some of the  $C$ -nodes that  $R_j$  is connected to have a single input: those are the nodes whose label contains  $N$ . There are  $\binom{N-2}{p-2}$  such nodes, and they all have different labels. For the rest of the proof, it is important to note that each of these labels contains  $j$ , and the  $\binom{N-2}{p-2}$  labels are all  $(p - 1)$ -element subsets of  $\mathcal{I}$  which contain  $j$  and do not contain  $N$ . Let us now consider the remaining  $\binom{N-1}{p-1} - \binom{N-2}{p-2} = \binom{N-2}{p-1}$   $C$ -nodes that  $R_j$  is connected to. Each of these nodes is connected to  $p$   $A$ -nodes. The labels of  $p - 1$  of these  $A$ -nodes contain  $j$ , and only one does not. That label is different for all  $C$ -nodes that the receiver  $R_j$  is connected to. Consequently,  $R_j$  gets  $\binom{N-2}{p-2}$  sources directly, and each source of the remaining  $\binom{N-2}{p-1}$  as a sum of that source and some  $p - 1$  of the sources received directly.

### 7.3.4 Throughput Number-of-Coding-Points Trade-off

We show this trade-off again for the specific class of  $\text{zK}(p, N)$  networks also used in the previous section. We examine a hybrid coding/routing scheme in which only a fraction of the nodes that are supposed to perform coding according to the scheme described in Section 7.3.3 are actually allowed only to forward one of their inputs. We derive an exact expression for the average throughput in this scenario, and show that it increases linearly with the number of coding points.

---

**Lemma 7.11.** Let  $A_{\text{zK}}^k$  be a hybrid coding/routing scheme in which the number of coding points in  $\Gamma_{\text{zK}}(p, N)$  that are allowed to perform linear combination of their inputs (as opposed to simply forwarding one of them) is restricted to  $k$ . The average throughput under this scheme  $T(A_{\text{zK}}^k)$  is given by

$$T(A_{\text{zK}}^k) = \frac{h}{N} \left( p + \frac{N - p}{p} + k \frac{p - 1}{h} \right). \quad (7.6)$$


---

*Proof.* If only  $k$  out of the  $\binom{N-1}{p}$  coding points are allowed to code, we get that

$$T_k^{av} = \frac{1}{N} \left[ \binom{N-1}{p-1} + (N-1) \binom{N-2}{p-2} + \left( \binom{N-1}{p} - k \right) + kp \right]. \quad (7.7)$$

In the above equation, we have

- the first term because receiver  $N$  observes all sources,
- the second term because each of the remaining  $N-1$  receivers observes  $\binom{N-2}{p-2}$  sources directly at the source nodes,
- the third term because, at the  $\binom{N-1}{p} - k$  forwarding points, exactly one receiver gets rate 1, and
- the fourth term because, the  $k$  coding points where coding is allowed, all of its  $p$  receivers get rate 1 by binary addition of the inputs at each coding point (see the description of the coding scheme in Section 7.3.3).

Equation (7.6) follows from (7.7) by simple arithmetics.  $\square$

Substituting  $k = h \frac{N-p}{p}$  in (7.7), i.e., using network coding at all coding points, we get that  $T_k^{av} = h$  as expected. At the other extreme, substituting  $k = 0$ , i.e., using routing only, we get an exact characterization of  $T_i^{av}$  as

$$T_k^{av} = T_i^{av} = \frac{h}{N} \left( p + \frac{N-p}{p} \right).$$

Theorem 7.6 shows that the throughput benefits increase *linearly* with the number of coding points  $k$ , at a rate of  $(p-1)/(hN)$ . Thus, a significant number of coding points is required to achieve a constant fraction of the network coding throughput.

## Notes

The very first paper on network coding by Ahlswede *et al.* required codes over arbitrarily large sequences. Alphabets of size  $N \cdot h$  were shown to be sufficient for designing codes based on the algebraic

approach of Koetter and Médard [32]. The polynomial time code design algorithms of Jaggi *et al.* in [30] operate on alphabets of size  $N$ . That codes for all networks with two sources require no more than  $\mathcal{O}(\sqrt{N})$  size alphabets was proved in by Fragouli and Soljanin in [24] by observing a connection with network code design and vertex coloring. The connection with coloring was independently observed by Rasala-Lehman and Lehman in [35]. It is not known whether alphabet of size  $\mathcal{O}(N)$  is necessary for general networks; only examples of networks for which alphabets of size  $\mathcal{O}(\sqrt{N})$  are necessary were demonstrated in [24, 35]. Algorithms for fast finite field operations can be found in [46].

The encoding complexity in terms of the number of coding points was examined in [24] who showed that the maximum number of coding points a network with two sources and  $N$  receivers can have equals to  $N - 1$ . Bounds on the number of coding points for general networks were derived by Langberg *et al.* in [34], who also showed that not only determining the number of coding points but even determining whether coding is required or not for a network is for most cases NP-hard. Coding with limited resources was studied by Chekuri *et al.* in [15], Cannons and Zeger in [12], and Kim *et al.* in [31].

# Appendix

---

## Points in General Position

---

For networks with  $h$  sources and linear network coding over a field  $\mathbb{F}_q$ , the coding vectors lie in  $\mathbb{F}_q^h$ , the  $h$ -dimensional vector space over the field  $\mathbb{F}_q$ . Since in network coding we only need to ensure linear independence conditions, we are interested in many cases in sets of vectors in general position:

---

**Definition A.1.** The vectors in a set  $\mathcal{A}$  are said to be *in general position* in  $\mathbb{F}_q^h$  if any  $h$  vectors in  $\mathcal{A}$  are linearly independent.

---

The moment curve provides an explicit construction of  $q$  vectors in general position. It can be defined as the range of the  $f : \mathbb{F}_q \rightarrow \mathbb{F}_q^h$  function

$$f(\alpha) = [1 \ \alpha \ \alpha^2 \ \dots \ \alpha^h].$$

---

**Definition A.2.** The *moment curve* in  $\mathbb{F}_q^h$  is the set

$$\{[1 \ \alpha \ \alpha^2 \ \dots \ \alpha^h] \mid \alpha \text{ primitive element of } \mathbb{F}_q\}.$$

---

The vectors in the moment curve are in general position. Indeed, take any  $h$  of these  $q$  vectors, the corresponding determinant

$$\det \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{h-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{h-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_h & x_h^2 & \dots & x_h^{h-1} \end{bmatrix},$$

is a Vandermonde determinant, that equals

$$\prod_{1 \leq j < i \leq h} (x_i - x_j).$$

Thus, provided  $x_i \neq x_j$  (for all  $i \neq j$ ), it is nonzero.

---

**Example A.1.** For  $q + 1 \geq h$ , the following set of  $q + 1$  points are in general position in  $\mathbb{F}_q^h$ :

$$\begin{array}{cccccc} [1 & x_1 & x_1^2 & \dots & x_1^{h-1}] & \\ [1 & x_2 & x_2^2 & \dots & x_2^{h-1}] & \\ \vdots & \vdots & \vdots & \dots & \vdots & \\ [1 & x_q & x_q^2 & \dots & x_q^{h-1}] & \\ [0 & 0 & 0 & \dots & 1] & \end{array} \quad \text{where } x_i \in \mathbb{F}_q \text{ and } x_i \neq x_j \text{ for } i \neq j.$$


---

**Example A.2.** For networks with two sources, we often use the  $q + 1$  vectors in the set

$$[0 \ 1], \ [1 \ 0], \ \text{and} \ [1 \ \alpha^i], \ \text{for } 1 \leq i \leq q - 1, \quad (\text{A.1})$$

where  $\alpha$  is a primitive element of  $\mathbb{F}_q$ . Any two different vectors in this set form a basis for  $\mathbb{F}_q^2$ , and thus these vectors are in general position. In fact, we cannot expect to find more than  $q + 1$  vectors in general position in  $\mathbb{F}_q^2$ . Indeed, if  $g$  denotes the maximum possible number of such vectors,  $g$  needs to satisfy

$$g(q - 1) + 1 \leq q^2 \Rightarrow g \leq q + 1.$$

This follows from a simple counting argument. For each of the  $g$  vector, we cannot reuse any of its  $q - 1$  nonzero multiples. We also cannot use

the all-zero vector. Hence the left-hand side. The right-hand side counts all possible vectors in  $\mathbb{F}_q^2$ .

Thus, for networks with two sources, we can, without loss of generality, restrict our selection of coding vectors to the set (A.1).

**Example A.3.** The following set of  $h + 1$  points are in general position in  $\mathbb{F}_q^h$ :

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

The maximum size  $g(h, q)$  that a set of vectors in  $\mathbb{F}_q^h$  in general position can have, is not known in general. Some bounds include the following:

- $g(h, q) \geq q + 1$ ,
- $g(h, q) \geq n + 1$ ,
- $g(h + 1, q) \leq 1 + g(h, q)$ ,
- $g(h, q) \leq q + n - 1$ .

Some known results include the following:

- $g(h, q) = q + 1$  if  $h = 2$  or if  $h = 3$  and  $q$  is odd,
- $g(h, q) = q + 1$  if  $q = h + 1$  and  $q$  is odd,
- $g(h, q) = q + 2$  if  $h = 3$  and  $q$  is even,
- $g(3, q) = q + 2$  for even  $q$ ,
- $g(4, q) = g(5, q) = q + 1$ ,
- $g(6, q) = g(7, q) = q + 1$  if  $q$  is even,
- $g(6, q) = q + 1$  if  $q = 11$  or  $13$ .

In general, for  $h \geq q$ , we know that  $g(h, q) = h + 1$ , whereas, for  $h \leq q$ , it holds that  $g(h, q) \geq h + 1$ , and it is widely believed that

$$g(h, q) = \begin{cases} q + 2 & \text{if } q \text{ is even and either } h = 3 \text{ or } h = q - 1, \\ q + 1 & \text{otherwise.} \end{cases}$$

## A.1 Projective Spaces and Arcs

Vectors in general position correspond to points in arcs in projective spaces. The projective space  $\mathbb{P}\mathbb{G}(h-1, q)$  is defined as follows:

---

**Definition A.3.** The *projective*  $(h-1)$ -space over  $\mathbb{F}_q$  is the set of  $h$ -tuples of elements of  $\mathbb{F}_q$ , not all zero, under the equivalence relation given by

$$[a_1 \cdots a_h] \sim [\lambda a_1 \cdots \lambda a_h], \quad \lambda \neq 0, \quad \lambda \in \mathbb{F}_q.$$


---

---

**Definition A.4.** In  $\mathbb{P}\mathbb{G}(h-1, q)$ , a  $k$ -arc is a set of  $k$  points any  $h$  of which form a basis for  $\mathbb{F}_q^h$ .

---

In a projective plane, i.e.,  $\mathbb{P}\mathbb{G}(2, q)$ , a  $k$ -arc is a set of  $k$  points no three of which are collinear (hence the name arc).

As a final note, arcs have been used by coding theorists in the context of MDS codes, i.e., codes that achieve the Singleton bound. Consider a matrix  $G$  with columns a set of vectors in general position in  $\mathbb{F}_q^h$ . Matrix  $G$  is a generator matrix for an MDS code of dimension  $h$  over  $\mathbb{F}_q$ .

### Notes

Although the problem of identifying  $g(h, q)$  looks combinatorial in nature, most of the harder results on the maximum size of arcs have been obtained by using algebraic geometry which is also a natural tool to use for understanding the structure (i.e., geometry) of arcs. The moment curve was discovered by Carathéodory in 1907 and then independently by Gale in 1956. A good survey on the size of arcs in projective spaces can be found in [3].

## Acknowledgments

---

This work was in part supported by the Swiss National Science Foundation under award No. PP002-110483, NSF under award No. CCR-0325673, and DIMACS under the auspices of Special focus on Computational Information Theory and Coding.

## Notations and Acronyms

---

$\mathbb{F}_q$ : finite field with  $q$  elements

$h$ : number of sources

$N$ : number of receivers

$\{G, S, \mathcal{R}\}$ : a multicast instance comprising of a directed graph  $G = (V, E)$ , a source vertex  $S \in V$ , and a set  $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$  of  $N$  receivers.

$\sigma_i$ : source  $S_i$  emits symbols  $\sigma_i$  over a finite field  $\mathbb{F}_q$

$(S_i, R_j)$ : path from source  $S_i$  to receiver  $R_j$

$S_i^e$ : source node in the line graph, edge through which source  $S_i$  flows

$c^\ell(e)$ : local coding vector associated with edge  $e$  of size  $1 \times |\text{In}(e)|$  and elements over  $\mathbb{F}_q$

$c(e)$ : global coding vector of size  $1 \times h$  and elements over  $\mathbb{F}_q$

$c_{xy}$ : capacity associated with the edge  $(x, y)$  that connects vertex  $x$  to vertex  $y$

$\gamma$ : a line graph

$\Gamma$ : an information flow decomposition graph

$\mathcal{D}$ : delay operator

$\text{In}(e)$ : set of incoming edges for edge  $e$

$\text{Out}(e)$ : set of outgoing edges for edge  $e$   
 $\text{In}(v)$ : set of incoming edges for vertex  $v$   
 $\text{Out}(v)$ : set of outgoing edges for vertex  $v$   
LP: Linear Program  
IP: Integer Program  
LIF: Linear Information Flow algorithm  
 $T_c$ : throughput achieved with network coding  
 $T_f$ : symmetric throughput achieved with rational routing  
 $T_i$ : symmetric throughput achieved with integral routing  
 $T_f^{av}$ : symmetric throughput achieved with rational routing  
 $T_i^{av}$ : symmetric throughput achieved with integral routing

## References

---

- [1] A. Agarwal and M. Charikar, “On the advantage of network coding for improving network throughput,” *IEEE Information Theory Workshop*, San Antonio, Texas, 2004.
- [2] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network information flow,” *IEEE Transactions on Information Theory*, vol. 46, pp. 1204–1216, July 2000.
- [3] A. H. Ali, J. W. P. Hirschfeld, and H. Kaneta, “On the size of arcs in projective spaces,” *IEEE Transaction on Information Theory*, vol. 41, pp. 1649–1656, September 1995.
- [4] M. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, “Network Models,” in *Handbooks in Operations Research and Management Science*, North Holland, 1994.
- [5] J. Bang-Jensen, A. Frank, and B. Jackson, “Preserving and increasing local edge-connectivity in mixed graphs,” *SIAM Journal on Discrete Mathematics*, vol. 8, pp. 155–178, 1995.
- [6] Á. M. Barbero and Ø. Ytrehus, “Heuristic algorithms for small field multicast encoding,” *2006 IEEE International Symposium Information Theory (ISIT’06)*, Chengdu, China, October 2006.
- [7] A. M. Barbero and Ø. Ytrehus, “Cycle-logical treatment for cyclopathic networks,” *Joint special issue of the IEEE Transactions on Information Theory and the IEEE/ACM Transaction on Networking*, vol. 52, pp. 2795–2804, June 2006.
- [8] B. Bollobás, *Modern Graph Theory*. Springer-Verlag, 2002.
- [9] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*. Amsterdam: North-Holland, 1979.

- [10] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, March 2004.
- [11] J. Cannons, R. Dougherty, C. Freiling, and K. Zeger, "Network routing capacity," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 777–788, March 2006.
- [12] J. Cannons and K. Zeger, "Network coding capacity with a constrained number of coding nodes," *Allerton Conference on Communication, Control, and Computing Allerton Park*, Illinois, September 2006.
- [13] Y. Cassuto and J. Bruck, "Network coding for non-uniform demands," *2005 IEEE International Symposium Information Theory (ISIT 2005)*, Adelaide, Australia, September 2005.
- [14] C. Chekuri, C. Fragouli, and E. Soljanin, "On achievable information rates in single-source non-uniform demand networks," *IEEE International Symposium on Information Theory*, July 2006.
- [15] C. Chekuri, C. Fragouli, and E. Soljanin, "On average throughput benefits and alphabet size for network coding," *Joint Special Issue of the IEEE Transactions on Information Theory and the IEEE/ACM Transactions on Networking*, vol. 52, pp. 2410–2424, June 2006.
- [16] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," *Allerton Conference on Communication, Control, and Computing*, Monticello, IL, October 2003.
- [17] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley & Sons, New York, 1991.
- [18] R. Diestel, *Graph Theory*. Springer-Verlag, 2000.
- [19] R. Dougherty, C. Freiling, and K. Zeger, "Insufficiency of linear coding in network information flow," *IEEE Transactions on Information Theory*, vol. 51, no. 8, pp. 2745–2759, August 2005.
- [20] P. Elias, A. Feinstein, and C. E. Shannon, "Note on maximum flow through a network," *IRE Transaction on Information Theory*, vol. 2, pp. 117–119, 1956.
- [21] E. Erez and M. Feder, "Convolutional network codes," *IEEE International Symposium on Information Theory (ISIT)*, Chicago 2004.
- [22] L. R. Ford Jr. and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.
- [23] C. Fragouli and E. Soljanin, "A connection between network coding and convolutional codes," *IEEE International Conference on Communications (ICC)*, vol. 2, pp. 661–666, June 2004.
- [24] C. Fragouli and E. Soljanin, "Information flow decomposition for network coding," *IEEE Transactions on Information Theory*, vol. 52, pp. 829–848, March 2006.
- [25] N. Harvey, "Deterministic network coding by matrix completion," MS Thesis, 2005.
- [26] T. Ho, R. Kötter, M. Médard, M. Effros, J. Shi, and D. Karger, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, pp. 4413–4430, October 2006.

- [27] T. Ho, B. Leong, R. Kötter, and M. Médard, “Distributed Asynchronous Algorithms for Multicast Network Coding,” *1st Workshop on Network Coding, WiOpt 2005*.
- [28] S. Jaggi, Y. Cassuto, and M. Effros, “Low complexity encoding for network codes,” in *Proceedings of 2006 IEEE International Symposium Information Theory (ISIT’06)*, Seattle, USA, July 2006.
- [29] S. Jaggi, P. A. Chou, and K. Jain, “Low complexity algebraic network multicast codes,” presented at *ISIT 2003*, Yokohama, Japan.
- [30] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, “Polynomial time algorithms for multicast network code construction,” *IEEE Transaction on Information Theory*, vol. 51, no. 6, no. 6, pp. 1973–1982, 2005.
- [31] M. Kim, C. W. Ahn, M. Médard, and M. Effros, “On minimizing network coding resources: An evolutionary approach,” *Network Coding Workshop*, 2006.
- [32] R. Kötter and M. Médard, “Beyond routing: An algebraic approach to network coding,” *IEEE/ACM Transaction on Networking*, vol. 11, pp. 782–796, October 2003.
- [33] G. Kramer and S. Savari, “Cut sets and information flow in networks of two-way channels,” in *Proceedings of 2004 IEEE International Symposium Information Theory (ISIT 2004)*, Chicago, USA, June 27–July 2 2004.
- [34] M. Langberg, A. Sprintson, and J. Bruck, “The encoding complexity of network coding,” *Joint special issue of the IEEE Transactions on Information Theory and the IEEE/ACM Transaction on Networking*, vol. 52, pp. 2386–2397, 2006.
- [35] A. R. Lehman and E. Lehman, “Complexity classification of network information flow problems,” *SODA*, 2004.
- [36] S.-Y. R. Li, R. W. Yeung, and N. Cai, “Linear network coding,” *IEEE Transactions on Information Theory*, vol. 49, pp. 371–381, February 2003.
- [37] Z. Li, B. Li, and L. C. Lau, “On achieving optimal multicast throughput in undirected networks,” in *Joint Special Issue on Networking and Information Theory, IEEE Transactions on Information Theory (IT) and IEEE/ACM Transactions on Networking (TON)*, vol. 52, June 2006.
- [38] D. S. Lun, N. Ratnakar, M. Médard, R. Kötter, D. R. Karger, T. Ho, E. Ahmed, and F. Zhao, “Minimum-cost multicast over coded packet networks,” *IEEE Transactions on Information Theory*, vol. 52, pp. 2608–2623, June 2006.
- [39] R. J. McEliece, “The algebraic theory of convolutional codes,” in *Handbook of Coding Theory*, (V. Pless and W. C. Huffman, eds.), North Holland, October 1998.
- [40] K. Menger, “Zur allgemeinen Kurventheorie,” *Fundamenta Mathematicae*, vol. 10, pp. 95–115, 1927.
- [41] A. Rasala-Lehman and E. Lehman, “Complexity classification of network information flow problems,” *SODA*, pp. 142–150, 2004.
- [42] S. Riis, “Linear versus nonlinear boolean functions in network flow,” in *Proceedings of 38th Annual Conference on Information Sciences and Systems (CISS’04)*, Princeton, NJ, March 2004.
- [43] P. Sanders, S. Egner, and L. Tolhuizen, “Polynomial time algorithms for network information flow,” in *Proceedings of 15th ACM Symposium on Parallel Algorithms and Architectures*, 2003.

- [44] A. Schrijver, *Theory of Linear and Integer Programming*. John Wiley & Sons, June 1998.
- [45] J. T. Schwartz, “Fast probabilistic algorithms for verification of polynomial identities,” *Journal of the ACM*, vol. 27, pp. 701–717, 1980.
- [46] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*. Cambridge Univ. Press, Second Edition, September 2003.
- [47] Y. Wu, P. A. Chou, and K. Jain, “A comparison of network coding and tree packing,” *ISIT 2004*, 2004.
- [48] R. W. Yeung, “Multilevel diversity coding with distortion,” *IEEE Transaction on Information Theory*, vol. 41, pp. 412–422, 1995.
- [49] R. W. Yeung, S.-Y. R. Li, N. Cai, and Z. Zhang, “Network coding theory: A tutorial,” *Foundation and Trends in Communications and Information Theory*, vol. 2, pp. 241–381, 2006.
- [50] L. Zosin and S. Khuller, “On directed Steiner trees,” in *Proceedings of the 13th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pp. 59–63, 2002.