UNIVERSITY OF CALIFORNIA, SAN DIEGO

Network Computing: Limits and Achievability

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Communication Theory and Systems)

by

Nikhil Karamchandani

Committee in charge:

Professor Massimo Franceschetti, Chair
Professor Ken Zeger, Co-Chair
Professor Young-Han Kim
Professor Alon Orlitsky
Professor Alexander Vardy

2011

The dissertation of Nikhil Karamchandani is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Co-Chair

Chair

University of California, San Diego

2011

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude towards Professor Massimo Franceschetti for his guidance and mentorship throughout the duration of my graduate studies. He has always treated me as a colleague and given me complete freedom to find and explore areas of research that interest me. He has been most instrumental in teaching me how to conduct scientific research and present complicated ideas in an accessible manner. For all this and more, I will be forever grateful.

I have been fortunate to have Professor Ken Zeger as a mentor and collaborator. His zeal for technical correctness and simple exposition are extraordinary and have greatly inspired me to pursue these virtues in all my future research. I gratefully acknowledge the support of my undergraduate advisor Prof. D. Manjunath, initial graduate advisor Prof. Rene Cruz, and Ph.D. defense committee members, Professor Young-Han Kim, Professor Alon Orlitsky, and Professor Alexander Vardy, who have all been very kind in devoting time to discuss research ideas whenever I have approached them. Finally, I thank Prof. Christina Fragouli for hosting me during my summer internship, providing a very stimulating research environment, and her guidance during that period and thereafter.

It has been my good fortune to have known many wonderful colleagues during my stay here. In particular, I would like to acknowledge my labmates Rathinakumar Appuswamy, Ehsan Ardestanizadeh, Lorenzo Coviello, Paolo Minero, and colleagues Jayadev Acharya, Abhijeet Bhorkar, Hirakendu Das, Arvind Iyengar, Lorenzo Keller, Mohammad Naghshvar, Matthew Pugh for their warm friendship and patient ear in discussing various research problems. I would also like to thank the ECE department staff, especially M'Lissa Michelson, John Minan, and Bernadette Villaluz, for all their help with administrative affairs.

Graduate life would not have been as pleasant without the companionship and support of my friends, especially Ankur Anchlia, Gaurav Dhiman, Nitin Gupta, Samarth Jain, Mayank Kabra, Uday Khankhoje, Himanshu Khatri, Neha Lodha, Vikram Mavalankar, Gaurav Misra, Abhijeet Paul, Nikhil Rasiwasia, Vivek Kumar Singh, Ankit Srivastava, Aneesh Subramaniam, and Neeraj Tripathi.

I owe the greatest debt to my family, especially my parents Mrs. Monita Karamchandani and Mr. Prakash Karamchandani, and my brother Ankit Karamchandani, for their unconditional love and support even during my long absence. Finally, a most special thanks to my wife, Megha Gupta, for always being there and making the worst days seem a lot better.

Chapter 2, in part, has been submitted for publication of the material. The dissertation author was the primary investigator and author of this paper. Chapter 3, in part, has been submitted for publication of the material. The dissertation author was a primary investigator and author of this paper. Chapter 4, in part, is a reprint of the material as it appears in R. Appuswamy, M. Franceschetti, N. Karamchandani and K. Zeger, “Network Coding for Computing: Cut-set bounds”, *IEEE Transactions on Information Theory*, vol. 57, no. 2, February 2011. The dissertation author was a primary investigator and author of this paper. Chapter 5 , in part, has been submitted for publication of the material. The dissertation author was a primary investigator and author of this paper.

VITA

2005	B.Tech. in Electrical Engineering, Indian Institute of Technology, Mumbai.
2007	M.S. in Electrical Engineering (Communication Theory and Systems), University of California, San Diego.
2011	Ph.D. in Electrical Engineering (Communication Theory and Systems), University of California, San Diego.

PUBLICATIONS

R. Appuswamy, M. Franceschetti, N. Karamchandani and K. Zeger, "*Linear Codes, Target Function Classes, and Network Computing Capacity*", submitted to the IEEE Transactions on Information Theory, May 2011.

N. Karamchandani, R. Appuswamy, and M. Franceschetti, "*Time and energy complexity of function computation over networks*", submitted to the IEEE Transactions on Information Theory, revised May 2011.

L. Keller, N. Karamchandani, C. Fragouli, and M. Franceschetti, "*Combinatorial designs for function computation over linear channels*", submitted to the Elsevier Physical Communication, Apr. 2011.

R. Appuswamy, M. Franceschetti, N. Karamchandani and K. Zeger, "*Network Coding for Computing: Cut-set bounds*", IEEE Transactions on Information Theory, Feb. 2011.

N. Karamchandani and M. Franceschetti, "*Scaling laws for delay sensitive traffic in Rayleigh fading networks*", Proceedings of the Royal Society, May 2008.

ABSTRACT OF THE DISSERTATION

Network Computing: Limits and Achievability

by

Nikhil Karamchandani

Doctor of Philosophy in Electrical Engineering (Communication Theory and Systems)

University of California, San Diego, 2011

Professor Massimo Franceschetti, Chair
Professor Ken Zeger, Co-Chair

Advancements in hardware technology have ushered in a digital revolution, with networks of thousands of small devices, each capable of sensing, computing, and communicating data, fast becoming a near reality. These networks are envisioned to be used for monitoring and controlling our transportation systems, power grids, and engineering structures. They are typically required to sample a field of interest, do ‘in-network’ computations, and then communicate a relevant summary of the data to a designated sink node(s), most often a function of the raw sensor measurements. In this thesis, we study such problems of network computing under various communication models. We derive theoretical limits on the performance of computation protocols as well as design efficient schemes which can match these limits. First, we begin with the one-shot computation problem where each node in a network is assigned an input bit and the objective is to compute a function f of the input messages at a designated receiver node. We study the energy and latency costs of function computation under both wired and wireless communication models. Next, we consider the case where the network operation is fixed, and its end result is to convey a fixed linear transformation of the source transmissions to the receiver. We design communication protocols that can compute functions without modifying the network operation. This model is motivated by practical considerations since constantly adapting the node operations according to changing demands is not always feasible in real networks. Thereafter, we move on to the case of repeated computation where source nodes in a network generate blocks of independent messages and a single receiver node computes a target function f for each instance of the source messages. The objective is to maximize the average number of times f can be computed per network usage, i.e., the *computing capacity*. We provide a generalized *min-cut* upper bound on the computing capacity and study its tightness for different classes of target functions and network topologies. Finally, we study the use of linear codes for network computing and quantify the benefits of non-linear coding vs linear coding vs routing for computing different classes of target functions.

Chapter 1

Introduction

Advancements in hardware technology have ushered in a digital revolution, with networks of thousands of small devices, each capable of sensing, computing, and communicating data, fast becoming a near reality. These networks are envisioned to be used for monitoring and controlling our transportation systems, power grids, and engineering structures. They are typically required to sample a field of interest, do ‘in-network’ computations, and then communicate a relevant summary of the data to a designated node(s), most often a function of the raw sensor measurements. For example, in environmental monitoring a relevant function can be the average temperature in a region. Another example is an intrusion detection network, where a node switches its message from 0 to 1 if it detects an intrusion and the function to be computed is the maximum of all the node messages. The engineering problem in these scenarios is to design schemes for computation which are efficient with respect to relevant metrics such as energy consumption and latency.

This new class of *computing networks* represents a paradigm shift from the way traditional *communication networks* operate. While the goal in the latter is usually to connect (multiple) source-destination pairs so that each destination can recover the messages from its intended source(s), the former aim to merge the information from the different sources to deliver useful summaries of the data to the destinations. Though there is a huge body of literature on communication networks and they have been studied extensively by both theorists and practitioners, computing networks are not as well understood. As argued above, such networks are going to be pervasive in the future and hence deserve close attention from the scientific community.

In this thesis, we study such problems of network computing under various communication models. We derive theoretical limits on the performance of computation protocols as well as design efficient schemes which can match these limits. The analysis uses tools from *communication complexity*, *information theory*, and *network coding*.

The thesis is organized as follows. In Chapter 2, we consider the following one-shot network computation problem: n nodes are placed on a $\sqrt{n} \times \sqrt{n}$ grid, each node is connected to every other node within distance $r(n)$ of itself, and it is given an arbitrary input bit. Nodes communicate with each other and a designated receiver node computes a target function f of the input bits, where f is either the *identity* or a *symmetric* function. We first consider a model where links are interference and noise-free, suitable for modeling wired networks. We then consider a model suitable for wireless networks. Due to interference, only nodes which do not share neighbors are allowed to transmit simultaneously; and when a node transmits a bit all of its neighbors receive an independent noisy copy of the bit. We present lower bounds on the minimum number of transmissions and the minimum number of time slots required to compute f. We also describe efficient schemes that match both of these lower bounds up to a constant factor and are thus jointly (near) optimal with respect to the number of transmissions and the number of time slots required for computation. Finally, we extend results on symmetric functions to more general network topologies, and obtain a corollary that answers an open question posed by El Gamal in 1987 regarding computation of the *parity* function over ring and tree networks.

In Chapter 3, we consider the case where the network operation is fixed, and its end result is to convey a fixed linear transformation of the source transmissions to the receiver. We design communication protocols that can compute functions without modifying the network operation, by appropriately selecting the codebook that the sources employ to map their input messages to the symbols they transmit over the network. We consider both the cases, when the linear transformation is known at the receiver and the sources and when it is apriori unknown to all. The model studied here is motivated by practical considerations: implementing networking protocols is hard and it is desirable to reuse the same network protocol to compute different target functions.

Chapter 4 considers the case of repeated computation where source nodes in a directed acyclic network generate blocks of independent messages and a single receiver node computes a target function f for each instance of the source messages. The objective is to maximize the average number of times f can be computed per network usage, i.e., the *computing capacity*. The *network coding* problem for a single-receiver network is a special case of the network computing problem in which all of the source messages must be reproduced at the receiver. For network coding with a single receiver, routing is known to achieve the capacity by achieving the network *min-cut* upper bound. We extend the definition of min-cut to the network computing problem and show that the min-cut is still an upper bound on the maximum achievable rate and is tight for computing (using coding) any target function in multi-edge tree networks and for computing linear target functions in any network. We also study the bound's tightness for different classes of target functions such as *divisible* and *symmetric* functions.

Finally, in Chapter 5 we study the use of linear codes for network computing in single-receiver networks with various classes of target functions of the source messages. Such classes include *reducible*, *injective*, *semi-injective*, and *linear* target functions over finite fields. Computing capacity bounds and achievability are given with respect to these target function classes for network codes that use routing, linear coding, or nonlinear coding.

Chapter 2

One-shot computation: Time and Energy Complexity

We consider the following network computation problem: n nodes are placed on a $\sqrt{n} \times \sqrt{n}$ grid, each node is connected to every other node within distance $r(n)$ of itself, and it is given an arbitrary input bit. Nodes communicate with each other and a designated sink node computes a function f of the input bits, where f is either the *identity* or a *symmetric* fuction. We first consider a model where links are interference and noise-free, suitable for modeling wired networks. Then, we consider a model suitable for wireless networks. Due to interference, only nodes which do not share neighbors are allowed to transmit simultaneously; and when a node transmits a bit all of its neighbors receive an independent noisy copy of the bit. We present lower bounds on the minimum number of transmissions and the minimum number of time slots required to compute f. We also describe efficient schemes that match both of these lower bounds up to a constant factor and are thus jointly (near) optimal with respect to the number of transmissions and the number of time slots required for computation. Finally, we extend results on symmetric functions to more general network topologies, and obtain a corollary that answers an open question posed by El Gamal in 1987 regarding computation of the *parity* function over ring and tree networks.

2.1 Introduction

Network computation has been studied extensively in the literature, under a wide variety of models. In wired networks with point-to-point noiseless communication links, computation has been traditionally studied in the context of communication complexity [1]. Wireless networks, on the other hand, have three distinguishing features: the inherent *broadcast* medium, *interference*, and *noise*. Due to the broadcast nature of the medium, when a node transmits a message, all of its neighbors receive it. Due to noise, the received message is a noisy copy of the transmitted one. Due to interference, simultaneous transmissions can lead to message collisions.

A simple *protocol model* introduced in [2] allows only nodes which do not share neighbors to transmit simultaneously to avoid interference. The works in [3–5] study computation restricted to the protocol model of operation and assuming noiseless transmissions. A noisy broadcast communication model over independent binary symmetric channels was proposed in [6] in which when a node transmits a bit, all of its neighbors receive an independent noisy copy of the bit. Using this model, the works in [7–9] consider computation in a *complete network* where each node is connected to every other node and only one node is allowed to transmit at any given time. An alternative to the complete network is the *random geometric network* in which n nodes are randomly deployed in continuous space inside a $\sqrt{n} \times \sqrt{n}$ square and each node can communicate with all other nodes in a range $r(n)$. Computation in such networks under the protocol model of operation and with noisy broadcast communication has been studied in [10–13]. In these works the connection radius $r(n)$ is assumed to be of order $\Theta\left(\sqrt{\log n}\right)$[1], which is the threshold required to obtain a connected random geometric network, see [14, Chapter 3].

We consider the class of *grid geometric networks* in which every node in a $\sqrt{n} \times \sqrt{n}$ grid is connected to every other node within distance r from it[2], see Fig-

[1] Throughout the thesis we use the following subset of the Bachman-Landau notation for positive functions of the natural numbers: $f(n) = O(g(n))$ as $n \to \infty$ if $\exists k > 0, n_0 : \forall n > n_0 \; f(n) \leq kg(n)$; $f(n) = \Omega(g(n))$ as $n \to \infty$ if $g(n) = O(f(n))$; $f(n) = \Theta(g(n))$ as $n \to \infty$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. The intuition is that f is asymptotically bounded up to constant factors from above, below, or both, by g.

[2] The connection radius r can be a function of n, but we suppress this dependence in the notation for ease of exposition.

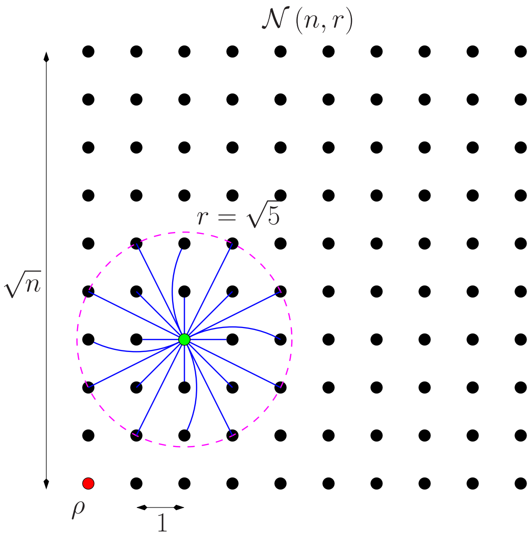


Figure 2.1: Network $\mathcal{N}(n,r)$: each node is connected to all nodes within distance r. The (red) node ρ is the sink that has to compute a function f of the input.

ure 2.1. This construction has many useful features. By varying the connection radius we can study a broad variety of networks with contrasting structural properties, ranging from the sparsely connected *g*rid network for $r = 1$ to the densely connected complete network when $r \geq \sqrt{2n}$. This provides intuition about how network properties like the average node degree impact the cost of computation and leads to natural extension of our schemes to more general network topologies. When $r \geq \sqrt{2n}$, all nodes are connected to each other and the network reduces to the complete one. Above the critical connectivity radius for the random geometric network $r = \Theta\left(\sqrt{\log n}\right)$, the grid geometric network has structural properties similar to its random geometric counterpart and all the results in this paper also hold in that scenario. Thus, our study includes the two network structures studied in previous works as special cases. At the end of the paper, we also present some extensions of our results to arbitrary network topologies.

We consider both noiseless wired communication over binary channels and noisy wireless communication over binary symmetric channels using the protocol model. We focus on computing two specific classes of functions with binary inputs, and measure the latency by the number of time slots it takes to compute the function and the energy cost by the total number of transmissions made in the network. The *identity* function (i.e. recover all source bits) is of interest because it can be used to compute any other function and thus gives a baseline to compare with when considering other functions. The class of *symmetric* functions includes all functions f such that for any input $\mathbf{x} \in \{0,1\}^n$ and permutation π on $\{1, 2, \ldots, n\}$,

$$f\left(x_1, x_2, \ldots, x_n\right) = f\left(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}\right).$$

In other words, the value of the function only depends on the arithmetic sum of the input bits, i.e., $\sum_{i=1}^{n} x_i$. Many functions which are useful in the context of sensor networks are symmetric, for example the *average*, *maximum*, *majority*, and *parity*.

2.1.1 Statement of results

Under the communication models described above, and for any connection radius $r \in [1, \sqrt{2n}]$, we prove lower bounds on the latency and on the number of transmissions required for computing the identity function. We then describe a scheme which matches these bounds up to a constant factor. Next, we consider the class of symmetric functions. For a particular symmetric target function (parity function), we provide lower bounds on the latency and the number of transmissions for computing the function. We then present a scheme which can compute any symmetric function while matching the above bounds up to a constant factor. These results are summarized in Tables 2.1 and 2.2. They illustrate the effect of the average node degree $\Theta(r^2)$ on the cost of computation under both communication models. By comparing the results for the identity function and symmetric functions, we can also quantify the gains in performance that can be achieved by using in-network aggregation for computation, rather than collecting all the data and perform the computation at the sink node. Finally, we extend our schemes to computing symmetric functions in more general network topologies and obtain a lower bound on the number of transmissions for arbitrary connected networks. A corollary of this result answers an open question originally posed by El Gamal in [6] regarding the computation of the parity function over ring and tree networks.

We point out that most of previous work ignored the issue of latency and is only concerned with minimizing the number of transmissions required for computation. Our schemes are latency-optimal, in addition to being efficient in terms of the number of

Table 2.1: Results for noiseless grid geometric networks.

Function	No. of time slots	No. of transmissions
Identity	$\Theta\left(n/r^2\right)$	$\Theta\left(n^{3/2}/r\right)$
Symmetric	$\Theta\left(\sqrt{n}/r\right)$	$\Theta\left(n\right)$

Table 2.2: Results for noisy grid geometric networks.

Function	No. of time slots	No. of transmissions
Identity	$\max\{\Theta(n), \Theta\left(r^2 \log\log n\right)\}$	$\max\{\Theta\left(n^{3/2}/r\right), \Theta\left(n \log\log n\right)\}$
Symmetric	$\max\{\Theta\left(\sqrt{n}/r\right), \Theta\left(r^2 \log\log n\right)\}$	$\max\{\Theta\left(n \log n/r^2\right), \Theta\left(n \log\log n\right)\}$

transmissions required. The works in [5, 11] consider the question of latency, but only for the case of $r = \Theta\left(\sqrt{\log n}\right)$.

The rest of the chapter is organized as follows. We formally describe the problem and mention some preliminary results in Section 2.2. Grid geometric networks with noiseless links are considered in Section 2.3 and their noisy counterparts are studied in Section 2.4. Extensions to general network topologies are presented in Section 2.5. In Section 2.6 we draw conclusions and mention some open problems.

2.2 Problem Formulation

A network $\mathcal{N}$ of n nodes is represented by an undirected graph. Nodes in the network represent communication devices and edges represent communication links. For each node i, let $N(i)$ denote its set of neighbors. Each node i is assigned an input bit $x_i \in \{0, 1\}$. Let $\mathbf{x}$ denote the vector whose i^{th} component is x_i. We refer to $\mathbf{x}$ as the input to the network. The nodes communicate with each other so that a designated sink node v^* can compute a *target function* f of the input bits,

$$f : \{0, 1\}^n \to \mathcal{B}$$

where $\mathcal{B}$ denotes the co-domain of f. Time is divided into slots of unit duration. The communication models are as follows.

- *Noiseless point-to-point model*: If a node i transmits a bit on an edge (i, j) in a time slot, then node j receives the bit without any error in the same slot. All the edges in the network can be used simultaneously, i.e., there is no interference.

- *Noisy broadcast model*: If a node i transmits a bit b in time slot t, then each neighboring node in $N(i)$ receives an independent noisy copy of b in the same slot. More precisely, neighbor $j \in N(i)$ receives $b \oplus \eta_{i,j,t}$ where $\oplus$ denotes modulo-2 sum. $\eta_{i,j,t}$ is a bernoulli random variable that takes value 1 with probability ϵ and 0 with probability $1 - \epsilon$. The noise bits $\eta_{i,j,t}$ are independent over i, j and t. A network in the noisy broadcast model with link error probability $1 - \epsilon$ is called an ϵ-noise network. We restrict to the protocol model of operation, namely two nodes i and j can transmit in the same time slot only if they do not have any common neighbors, i.e., $N(i) \cap N(j) = \phi$. Thus, any node can receive at most one bit in a time slot. In the protocol model originally introduced in [2] communication is reliable. In our case, even if bits do not collide at the receiver because of the protocol model of operation, there is still a probability of error ϵ which models the inherent noise in the wireless communication medium.

A scheme for computing a target function f specifies the order in which nodes in the network transmit and the procedure for each node to decide what to transmit in its turn. A scheme is defined by the total number of time slots T of its execution, and for each slot $t \in \{1, 2, \ldots, T\}$, by a collection of S_t simultaneously transmitting nodes $\left\{v_1^t, v_2^t, \ldots v_{S_t}^t\right\}$ and corresponding encoding functions $\left\{\phi_1^t, \phi_2^t, \ldots, \phi_{S_t}^t\right\}$. In any time slot $t \in \{1, 2, \ldots, T\}$, node v_j^t computes the function $\phi_j^t : \{0,1\} \times \{0,1\}^{\varphi_j^t} \to \{0,1\}$ of its input bit and the φ_j^t bits it received before time t and then transmits this value. In the noiseless point-to-point case, nodes in the list S_t are repeated for each distinct edge on which they transmit in a given slot. After the T rounds of communication, the sink node ρ computes an estimate $\hat{f}$ of the value of the function f. The duration T of a scheme and the total number of transmissions $\sum_{i=1}^{T} S_t$ are constants for all inputs $\mathbf{x} \in \{0,1\}^n$.

Our scheme definition has a number of desirable properties. First, schemes are *oblivious* in the sense that in any time slot, the node which transmits is decided ahead of time and does not depend on a particular execution of the scheme. Without this property, the noise in the network may lead to multiple nodes transmitting at the same time, thereby causing collisions and violating the protocol model. Second, the definition rules out communication by *silence*: when it is a node's turn to transmit, it must send something.

We call a scheme a δ-error scheme for computing f if for any input $\mathbf{x} \in \{0,1\}^n$, $\Pr\left(\hat{f}(\mathbf{x}) \neq f(\mathbf{x})\right) \leq \delta$. For both the noiseless and noisy broadcast communication models, our objective is to characterize the minimum number of time slots T and the minimum number of transmissions required by any δ-error scheme for computing a target function f in a network $\mathcal{N}$. We first focus on grid geometric networks of connection radius r, denoted by $\mathcal{N}(n,r)$, and then extend our results to more general network topologies.

2.2.1 Preliminaries

We mention some known useful results.

Remark 2.2.1. For any connection radius $r < 1$, every node in the grid geometric network $\mathcal{N}(n,r)$ is isolated and hence computation is infeasible. On the other hand, for any $r \geq \sqrt{2n}$, the network $\mathcal{N}(n,r)$ is fully connected. Thus the interesting regime is when the connection radius $r \in [1, \sqrt{2n}]$.

Remark 2.2.2. For any connection radius $r \in [1, \sqrt{2n}]$, every node in the grid geometric network $\mathcal{N}(n,r)$ has $\Theta(r^2)$ neighbors.

Theorem 2.2.3. *(Gallager's Coding Theorem) [10, Page 3, Theorem 2], [15]: For any $\gamma > 0$ and any integer $m \geq 1$, there exists a code for sending an m-bit message over a binary symmetric channel using $O(m)$ transmissions such that the message is received correctly with probability at least $1 - e^{-\gamma m}$.*

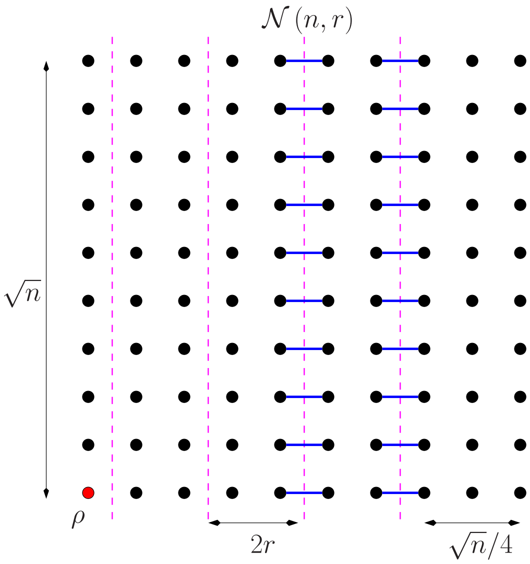


Figure 2.2: Each dashed (magenta) line represents a cut of network $\mathcal{N}(n,r)$ which separates at least $n/4$ nodes from the sink ρ. Since the cuts are separated by a distance of at least $2r$, the edges in any two cuts, denoted by the solid (blue) lines, are disjoint.

2.3 Noiseless Grid Geometric Networks

We begin by considering computation of the identity function. We have the following straightforward lower bound.

Theorem 2.3.1. *Let f be the identity function, let $\delta \in [0, 1/2)$, and let $r \in [1, \sqrt{2n}]$. Any δ-error scheme for computing f over $\mathcal{N}(n,r)$ requires at least $\Omega(n/r^2)$ time slots and $\Omega\left(n^{3/2}/r\right)$ transmissions.*

Proof. To compute the identity function the sink node ρ should receive at least $(n-1)$ bits. Since ρ has $O(r^2)$ neighbors and can receive at most one bit on each edge in a time slot, it will require at least $\Omega(n/r^2)$ time slots to compute the identity function.

Let a cut be any set of edges separating at least one node from the sink ρ. It is easy to verify that there exists a collection of $\Omega(\sqrt{n}/r)$ disjoint cuts such that each cut separates $\Omega(n)$ nodes from the sink ρ, see Figure 2.2 for an example. Thus to ensure that ρ can compute the identity function, there should be at least $\Omega(n)$ transmissions across each cut. The lower bound on the total number of transmissions then follows. ■

We now present a simple scheme for computing the identity function which is order-optimal in both the latency and the number of transmissions.

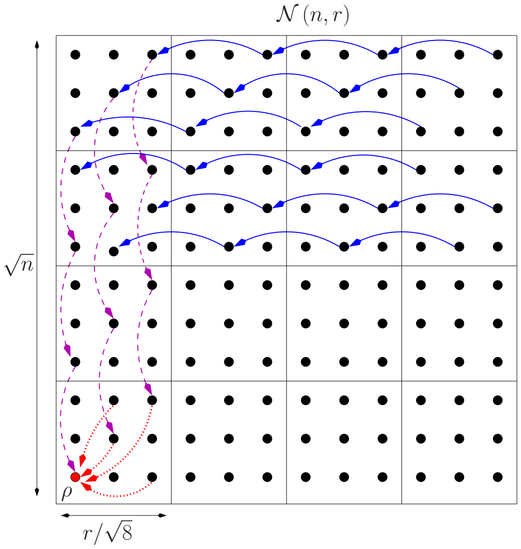


Figure 2.3: The scheme for computing the identity function works in three phases: the solid (blue) lines depict the first horizontal aggregation phase, the dashed (magenta) lines denote the second vertical aggregation phase, and the dotted (red) lines represent the final phase of downloading data to the sink.

Theorem 2.3.2. *Let f be the identity function and let $r \in [1, \sqrt{2n}]$. There exists a zero-error scheme for computing f over $\mathcal{N}(n,r)$ which requires at most $O(n/r^2)$ time slots and $O\left(n^{3/2}/r\right)$ transmissions.*

Proof. Let $c = r/\sqrt{8}$. Consider a partition of the network $\mathcal{N}(n,r)$ into cells of size $c \times c$, see Figure 2.3. Note that each node is connected to all nodes in its own cell as well as in any neighboring cell. The scheme works in three phases, see Figure 2.3. In the first phase, bits are horizontally aggregated towards the left-most column of cells along parallel linear chains. In the second phase, the bits in the left-most cells are vertically aggregated towards the nodes in the cell containing the sink node ρ. In the final phase, all the bits are collected at the sink node.

The first phase has bits aggregating along $O(\sqrt{n}r)$ parallel linear chains each of length $O(\sqrt{n}/r)$. By pipelining the transmissions, this phase requires $O(\sqrt{n}/r)$ time slots and a total of $O(\sqrt{n}r \times n/r^2)$ transmissions in the network. Since each node in the left-most column of cells has $O(\sqrt{n}/r)$ bits and there are $O(r^2)$ parallel chains each of length $O(\sqrt{n}/r)$, the second phase uses $O(r^2 \times \sqrt{n}/r \times n/r^2)$ transmissions and $O(\sqrt{n}/r \times \sqrt{n}/r)$ time slots. In the final phase, each of the $O(r^2)$ nodes in the cell with ρ has $O(n/r^2)$ bits and hence it requires $O(n)$ transmissions and $O(n/r^2)$ slots to

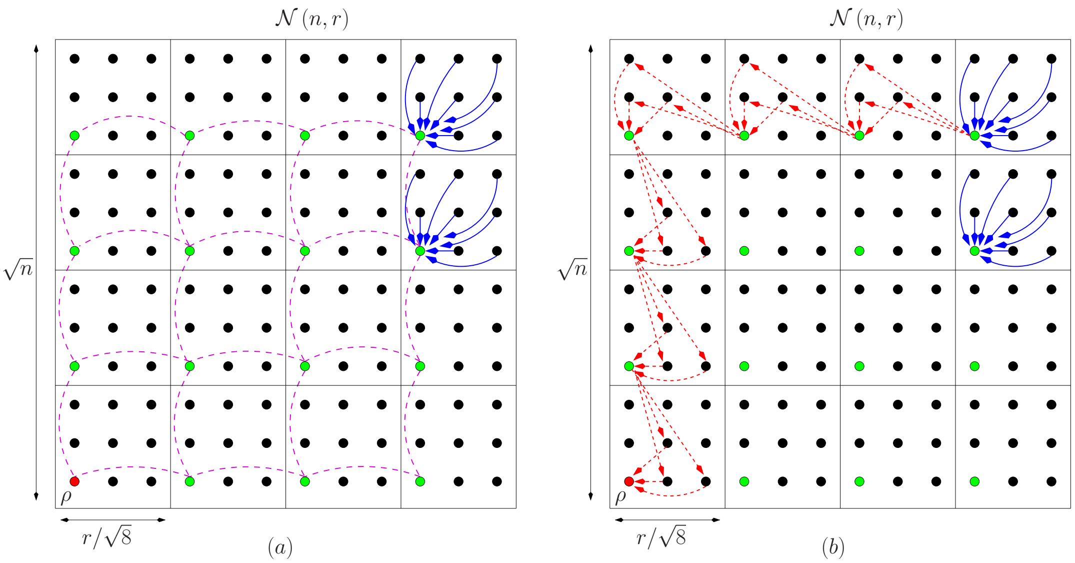


Figure 2.4: Figures (a) and (b) represent the cases $r \leq \sqrt{8\log n}$ and $r > \sqrt{8\log n}$ respectively. The scheme for computing any symmetric function works in two phases: the solid (blue) lines indicate the first phase which is the same in both cases. The second phase differs in the two cases. It is represented by the dashed (magenta) lines in Fig. (a) and the dashed (red) lines in Fig. (b).

finish. Adding the costs, the scheme can compute the identity function with $O\left(n^{3/2}/r\right)$ transmissions and $O\left(n/r^2\right)$ time slots. ■

Now we consider the computation of symmetric functions. We have the following straightforward lower bound:

Theorem 2.3.3. *Let $\delta \in [0,1/2)$ and let $r \in [1,\sqrt{2n}]$. There exists a symmetric target function f such that any δ-error scheme for computing f over $\mathcal{N}(n,r)$ requires at least $\Omega\left(\sqrt{n}/r\right)$ time slots and $(n-1)$ transmissions.*

Proof. Let f be the parity function. To compute this function, each non-sink node in the network should transmit at least once. Hence, at least $(n-1)$ transmissions are required. Since the bit of the farthest node requires at least $\Omega\left(\sqrt{n}/r\right)$ time slots to reach ρ, we have the desired lower bound on the latency of any scheme. ■

Next, we present a matching upper bound.

Theorem 2.3.4. *Let f be any symmetric function and let $r \in [1, \sqrt{2n}]$. There exists a zero-error scheme for computing f over $\mathcal{N}(n,r)$ which requires at most $O(\sqrt{n}/r)$ time slots and $O(n)$ transmissions.*

Proof. We present a scheme which can compute the arithmetic sum of the input bits over $\mathcal{N}(n,r)$ in at most $O(\sqrt{n}/r)$ time slots and $O(n)$ transmissions. This suffices to prove the result since f is symmetric and thus its value only depends on the arithmetic sum of the input bits.

Again, consider a partition of the noiseless network $\mathcal{N}(n,r)$ into cells of size $c \times c$ with $c = r/\sqrt{8}$. For each cell, pick one node arbitrarily and call it the "cell-center". For the cell containing ρ, choose ρ to be the cell center. The scheme works in two phases, see Figure 2.4.

First phase: All the nodes in a cell transmit their input bits to the cell-center. This phase requires only one time-slot and n transmissions and at the end of the phase each cell-center knows the arithmetic sum of the input bits in its cell, which is an element of $\{0, 1, \ldots, \Theta(r^2)\}$.

Second phase: In this phase, the bits at the cell-centers are aggregated so that ρ can compute the arithmetic sum of all the input bits in the network. There are two cases, depending on the connection radius r.

- $r \le \sqrt{8 \log n}$: Since each cell-center is connected to the other cell-centers in its neighboring cells, this phase can be mapped to computing the arithmetic sum over the noiseless network $\mathcal{N}(\Theta(n/r^2), 1)$ where each node observes a message in $\{0, 1, \ldots, \Theta(r^2)\}$. See Figure 2.4$(a)$ for an illustration. In Appendix 2.1 we present a scheme to complete this phase using $O(n/r^2)$ transmissions and $O(\sqrt{n}/r)$ time slots.
- $r > \sqrt{8 \log n}$: The messages at cell-centers are aggregated towards ρ along a tree, see Figure 2.4(b). The value at each cell-center can be viewed as a $\lceil \log n \rceil$-length binary vector. To transmit its vector to the parent (cell-center) node in the tree, every leaf node (in parallel) transmits each bit of the vector to a distinct node in the parent cell. In the next time slot, each of these intermediate nodes relays its received bit to the corresponding cell-center. The parent cell-center can then reconstruct the message and aggregate it with its own value to form another $\lceil \log n \rceil$-length binary vector. Note that it requires two time slots and $O(\log n)$ transmissions by a cell-center to traverse one level

of depth in the aggregation tree. This step is performed repeatedly (in succession) till the sink node ρ receives the sum of all the input bits in the network. Since the depth of the aggregation tree is $O\left(\sqrt{n}/r\right)$, the phase requires $O\left(\sqrt{n}/r\right)$ time slots. There are $O\left(\log n\right)$ transmissions in each cell of the network. Hence the phase requires a total of $O\left(n/r^2 \times \log n\right) = O\left(n\right)$ transmissions.

Adding the costs of the two phases, we conclude that it is possible to compute any symmetric function using $O(n)$ transmissions and $O\left(\sqrt{n}/r\right)$ time slots. ■

2.4 Noisy Grid Geometric Networks

We start by considering the computation of the identity function. We have the following lower bound.

Theorem 2.4.1. *Let f be the identity function. Let $\delta \in (0, 1/2)$, let $\epsilon \in (0, 1/2)$, and let $r \in [1, \sqrt{2n}]$. Any δ-error scheme for computing f over an ϵ-noise grid geometric network $\mathcal{N}\left(n, r\right)$ requires at least $\max\{n-1, \Omega\left(r^2 \log\log n\right)\}$ time slots and $\max\{\Omega\left(n^{3/2}/r\right), \Omega\left(n \log\log n\right)\}$ transmissions.*

Proof. The lower bound of $\Omega\left(n^{3/2}/r\right)$ transmissions follows from the same argument as in the proof of Theorem 2.3.1. The other lower bound of $\Omega\left(n \log\log n\right)$ transmissions follows from [8, Corollary 2].

We now turn to the number of time slots required. For computing the identity function, the sink node ρ should receive at least $(n-1)$ bits. However, the sink can receive at most one bit in any slot and hence any scheme for computing the identity function requires at least $(n-1)$ time slots. For the remaining lower bound, consider a partition of the network $\mathcal{N}\left(n, r\right)$ into cells of size $c \times c$ with $c = r/\sqrt{8}$. Since the total number of transmissions in the network is at least $\Omega\left(n \log\log n\right)$ and there are $O\left(n/r^2\right)$ cells, there is at least one cell where the number of transmissions is at least $\Omega\left(r^2 \log\log n\right)$. Since all nodes in a cell are connected to each other, at most one of them can transmit in a slot. Thus any scheme for computing the identity function requires at least $\Omega\left(r^2 \log\log n\right)$ time slots. ■

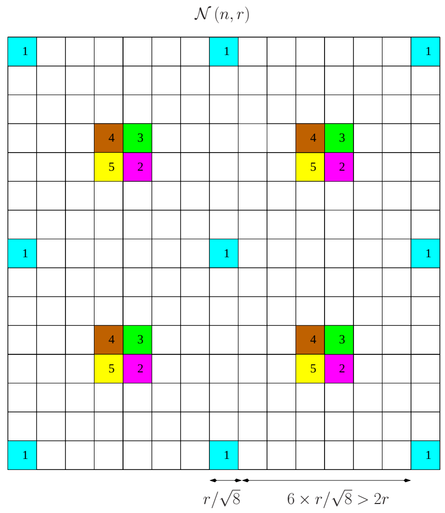


Figure 2.5: Cells with the same number (and color) can be active in the same time slot and different numbers (colors) activate one after the other. Each cell is active once in 49 slots.

Next, we present an efficient scheme for computing the identity function in noisy broadcast networks, which matches the above bounds.

Theorem 2.4.2. *Let f be the identity function. Let $\delta \in (0, 1/2)$, let $\epsilon \in (0, 1/2)$, and let $r \in [1, \sqrt{2n}]$. There exists a δ-error scheme for computing f over an ϵ-noise grid geometric network $\mathcal{N}(n,r)$ which requires at most* $\max\{O(n), O(r^2 \log\log n)\}$ *time slots and* $\max\{O\left(n^{3/2}/r\right), O(n \log\log n)\}$ *transmissions.*

Proof. Consider the usual partition of the network $\mathcal{N}(n,r)$ into cells of size $c \times c$ with $c = r/\sqrt{8}$. By the protocol model of operation any two nodes are allowed to transmit in the same time slot only if they do not have any common neighbors. Cells are scheduled according to the scheme shown in Figure 2.5 to ensure that all transmissions are successful. Thus, each cell is scheduled once every 7×7 time slots. Within a cell, at most one node can transmit in any given time slot and nodes take turns to transmit one after the other. For each cell, pick one node arbitrarily and call it the "cell-center". The scheme works in three phases, see Figure 2.6.

First phase: There are two different cases, depending on the connection radius r.

- $r \leq \sqrt{n}/\log n$: In this case, each node in its turn transmits its input bit to the

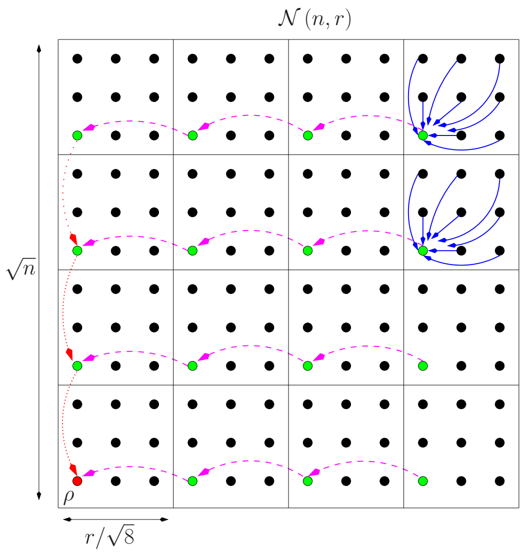


Figure 2.6: The scheme for computing the identity function in a noisy network involves three phases: the solid (blue) lines indicate the first in-cell aggregation phase, the dashed (magenta) lines represent the second horizontal aggregation phase, and the dotted (red) lines represent the final vertical aggregation phase.

corresponding cell-center using a codeword of length $O(\log n)$ such that the cell-center decodes the message correctly with probability at least $1-1/n^2$. The existence of such a code is guaranteed by Theorem 2.2.3. This phase requires at most $O(r^2 \log n)$ time slots and at most $O(n \log n)$ transmissions in the network. Since there are $O(n/r^2)$ cells in the network, the probability that the computation fails in at least one cell is bounded by $O(1/n)$.

- $r \geq \sqrt{n}/\log n$: In this case, each cell uses the more sophisticated scheme described in [8, Section 7] for recovering all the input messages from the cell at the cell-center. This scheme requires at most $O(r^2 \log\log n)$ time slots and a total of at most $O(n/r^2 \times r^2 \log\log n)$ transmissions in the network. At the end of the scheme, a cell-center has all the input messages from its cell with probability of error at most $O(\log n/n)$. Since there are at most $\log^2 n$ cells in the network for this case, the probability that the computation fails in at least one cell is bounded by $O(\log^3 n/n)$.

Thus at the end of the first phase, all cell-centers in the network have the input bits of the nodes in their cells with probability at least $1 - O(\log^3 n/n)$.

Second phase: In this phase, the messages collected at the cell-centers are aggregated horizontally towards the left-most cells, see Figure 2.6. Note that there are

$\sqrt{n}/r$ horizontal chains and each cell-center has $O\left(r^2\right)$ input messages. In each such chain, the rightmost cell-center maps its set of messages into a codeword of length $O\left(\sqrt{n}r\right)$ and transmits it to the next cell-center in the horizontal chain. The receiving cell-center decodes the incoming codeword, appends its own input messages, re-encodes it into a codeword of length $O\left(\sqrt{n}r\right)$, and then transmits it to the next cell-center, and so on. This phase requires at most $O\left(\sqrt{n}r \times \sqrt{n}/r\right)$ time slots and a total of at most $O\left(\sqrt{n}r \times n/r^2\right)$ transmissions in the network. From Theorem 2.2.3, this step can be executed without error with probability at least $1 - O\left(1/n\right)$.

Third phase: In the final phase, the messages at the cell-centers of the left-most column are aggregated vertically towards the sink node ρ, see Figure 2.6. Each cell-center maps its set of input messages into a codeword of length $O\left(\sqrt{n}r\right)$ and transmits it to the next cell-center in the chain. The receiving cell-center decodes the incoming message, re-encodes it, and then transmits it to the next node, and so on. By pipelining the transmissions, this phase requires at most $O\left(\sqrt{n}r \times \sqrt{n}/r\right)$ time slots and at most $O\left(\sqrt{n}r \times n/r^2\right)$ transmissions in the network. This phase can also be executed without error with probability at least $1 - O\left(1/n\right)$.

It now follows that at the end of the three phases, the sink node ρ can compute the identity function with probability of error at most $O(\log^3 n/n)$. Thus for n large enough, we have a δ-error scheme for computing any symmetric function in the network $\mathcal{N}\left(n,r\right)$. Adding the costs of the phases, the scheme requires at most $\max\{O\left(n\right), O\left(r^2 \log\log n\right)\}$ time slots and $\max\{O\left(n^{3/2}/r\right), O\left(n \log\log n\right)\}$ transmissions. ■

We now discuss the computation of symmetric functions in noisy broadcast networks. We begin with a lower bound on the latency and the number of transmissions required.

Theorem 2.4.3. *Let $\delta \in (0, 1/2)$, let $\epsilon \in (0, 1/2)$, and let $r \in [1, n^{1/2-\beta}]$ for any $\beta > 0$. There exists a symmetric target function f such that any δ-error scheme for computing f over an ϵ-noise grid geometric network $\mathcal{N}\left(n,r\right)$ requires at least $\max\{\Omega\left(\sqrt{n}/r\right), \Omega\left(r^2 \log\log n\right)\}$ time slots and $\max\{\Omega\left(n \log n/r^2\right), \Omega\left(n \log\log n\right)\}$ transmissions.*

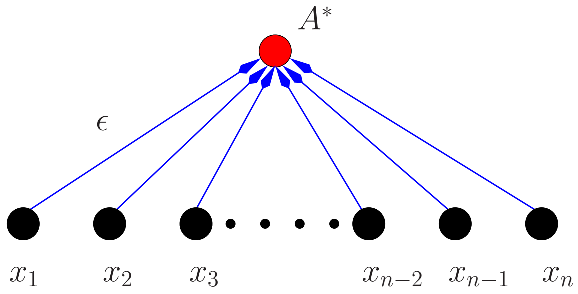


Figure 2.7: The n-noisy star network.

We briefly describe the idea of the proof before delving into details. Let f be the parity function. First, we notice that [12, Theorem 1.1, page 1057] immediately implies that any δ-error scheme for computing f over $\mathcal{N}(n, r)$ requires at least $\Omega(n \log \log n)$ transmissions. So, we only need to establish that any such scheme also requires $\Omega(n \log n / r^2)$ transmissions.

Suppose there exists a δ-error scheme $\mathcal{P}$ for computing the parity function in an ϵ-noise grid geometric network $\mathcal{N}(n, r)$ which requires S transmissions. In Lemma 2.4.5 we translate the given scheme $\mathcal{P}$ into a new scheme $\mathcal{P}_1$ operating on a "noisy star" network (see Figure 2.7) of noise parameter dependent on Sr^2/n, such that the probability of error for the new scheme $\mathcal{P}_1$ is also at most δ. In Lemma 2.4.4 we derive a lower bound on the probability of error of the scheme $\mathcal{P}_1$ in terms of the noise parameter of the noisy star network (which depends on Sr^2/n). Combining these results we obtain the desired lower bound on the number of transmissions S. We remark that while the proof of the lower bound in [12, Theorem 1.1, page 1057] operates a transformation to a problem over "noisy decision trees", here we need to transform the problem into one over a noisy star network. Hence, the two different transformations lead to different lower bounds on the number of transmissions required for computation.

A *n-noisy star network* consists of n input nodes and one auxiliary node A^*. Each of the n input nodes is connected directly to A^* via a noisy link, see Figure 2.7. We have the following result for any scheme which computes the parity function in an n-noisy star network:

Lemma 2.4.4. *Consider an n-noisy star network of noise parameter ϵ and let the input $\mathbf{x}$ be distributed uniformly over $\{0,1\}^n$. For any scheme $\mathcal{P}_1$ which computes the parity* *function (on n bits) in the network and in which each input node transmits its input bit only once, the probability of error is at least $\left(1-(1-2\epsilon)^n\right)/2$.*

Proof. See Appendix 2.2.1. ■

We have the following lemma relating the original network $\mathcal{N}(n,r)$ and a noisy star network.

Lemma 2.4.5. *Let $\alpha \in (0,1)$. If there is a δ-error scheme $\mathcal{P}$ for computing the parity function (on n input bits) in $\mathcal{N}(n,r)$ with S transmissions, then there is a δ-error scheme $\mathcal{P}_1$ for computing the parity function (on αn input bits) in an αn-noisy star network with noise parameter $\epsilon^{O(Sr^2/n)}$, with each input node transmitting its input bit only once.*

Proof. See in Appendix 2.2.2. ■

We are now ready to complete the proof of Theorem 2.4.3.

Proof (of Theorem 2.4.3). Let $\alpha \in (0,1)$. If there is a δ-error scheme for computing the parity function in $\mathcal{N}(n,r)$ which requires S transmissions, then by combining the Lemmas 2.4.5 and 2.4.4, the following inequalities must hold:

$$\begin{aligned} \delta &\geq \frac{1-\left(1-2\epsilon^{O\left(Sr^2/n\right)}\right)^{\alpha n}}{2}, \\ \implies \left(1-2\epsilon^{O\left(Sr^2/n\right)}\right)^{\alpha n} &\geq 1-2\delta \\ \implies \left(2^{-2\epsilon^{O\left(Sr^2/n\right)}}\right)^{\alpha n} &\overset{(a)}{\geq} 1-2\delta \\ \implies S &\geq \Omega\left(\frac{n\left(\log n-\log\log\left(1/(1-2\delta)\right)\right)}{r^2\log(1/\epsilon)}\right) \end{aligned} \tag{2.1}$$

where (a) follows since $2^{-x} \geq 1-x$ for every $x > 0$. Thus we have that any δ-error scheme for computing the parity function in an ϵ-noise network $\mathcal{N}(n,r)$ requires at least $\Omega\left(n\log n/r^2\right)$ transmissions.

We now consider the lower bound on the number of time slots. Since the message of the farthest node requires at least $\Omega\left(\sqrt{n}/r\right)$ time slots to reach ρ, we have the corresponding lower bound on the duration of any δ-error scheme. The lower bound of $\Omega\left(r^2\log\log n\right)$ time slots follows from the same argument as in the proof of Theorem 2.4.1. ■

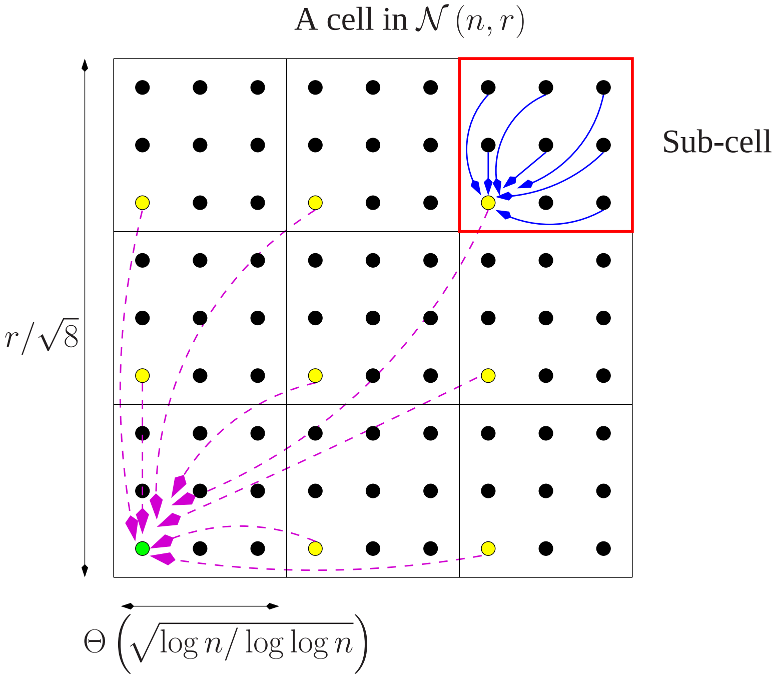

A cell in $\mathcal{N}(n,r)$
Sub-cell
$r/\sqrt{8}$
$\Theta\left(\sqrt{\log n/\log\log n}\right)$

Figure 2.8: Each cell in the network $\mathcal{N}(n,r)$ is divided into sub-cells of side $\Theta\left(\sqrt{\log n/\log\log n}\right)$. Each sub-cell has a "head", denoted by a yellow node. The sum of input messages from each sub-cell is obtained at its head node, depicted by the solid (blue) lines. These partial sums are then aggregated at the cell-center. The latter step is represented by the dashed (magenta) lines.

We now present an efficient scheme for computing any symmetric function in a noisy broadcast network which matches the above lower bounds.

Theorem 2.4.6. *Let f be any symmetric function. Let $\delta \in (0,1/2)$, let $\epsilon \in (0,1/2)$, and let $r \in [1,\sqrt{2n}]$. There exists a δ-error scheme for computing f over an ϵ-noise grid geometric network $\mathcal{N}(n,r)$ which requires at most* $\max\{O(\sqrt{n}/r), O(r^2\log\log n)\}$ *time slots and* $\max\{O(n\log n/r^2), O(n\log\log n)\}$ *transmissions.*

Proof. We present a scheme which can compute the arithmetic sum of the input bits over $\mathcal{N}(n,r)$. Note that this suffices to prove the result since f is symmetric and thus its value only depends on the arithmetic sum of the input bits.

Consider the usual partition of the network $\mathcal{N}(n,r)$ into cells of size $c \times c$ with $c = r/\sqrt{8}$. For each cell, we pick one node arbitrarily and call it the "cell-center". As before, cells are scheduled to prevent interference between simultaneous transmissions according to Figure 2.5. The scheme works in three phases.

First phase: The objective of the first phase is to ensure that each cell-center computes the arithmetic sum of the input messages from the corresponding cell. Depending on the connection radius r, this is achieved using two different strategies.

- $r \leq \sqrt{\log n / \log\log n}$: In Appendix 2.3, we describe a scheme which can compute the partial sums at all cell-centers with probability at least $1 - O(1/n)$ and requires $O\left(n/r^2 \times \log n\right)$ total transmissions and $O\left(\log n\right)$ time slots.

- $r > \sqrt{\log n / \log\log n}$: In this case, we first divide each cell into smaller sub-cells with $\Theta\left(\log n / \log\log n\right)$ nodes each, see Figure 2.8. Each sub-cell has an arbitrarily chosen "head" node. In each sub-cell, we use the *Intra-cell scheme* from [10, Section III] to compute the sum of the input bits from the sub-cell at the corresponding head node. This requires $O\left(\log\log n\right)$ transmissions from each node in the sub-cell. Since there are $O\left(r^2\right)$ nodes in each cell and only one node in a cell can transmit in a time slot, this step requires $O\left(r^2 \log\log n\right)$ time slots and a total of $O\left(n \log\log n\right)$ transmissions in the network. The probability that the computation fails in at least one sub-cell is bounded by $O(1/n)$.

Next, each head node encodes the sum of the input bits from its sub-cell into a codeword of length $O\left(\log n\right)$ and transmits it to the corresponding cell-center. This step requires a total of $O\left(n \log\log n\right)$ transmissions in the network and $O\left(r^2 \log\log n\right)$ time slots and can be performed also with probability of error at most $O(1/n)$.

The received values are aggregated so that at the end of the first phase, all cell-centers know the sum of their input bits in their cell with probability at least $1 - O(1/n)$. The phase requires $O\left(n \log\log n\right)$ transmissions in the network and $O\left(r^2 \log\log n\right)$ time slots to complete.

Second phase: In this phase, the partial sums stored at the cell-centers are aggregated along a tree (see for example, Figure 2.6) so that the sink node ρ can compute the sum of all the input bits in the network. We have the following two cases, depending on the connection radius r.

- $r \geq \left(\sqrt{n} \log n\right)^{1/3}$: For this regime, our aggregation scheme is similar to the *Inter-cell scheme* in [10, Section III]. Each cell-center encodes its message into a codeword of length $\Theta\left(\log n\right)$. Each leaf node in the aggregation tree sends its codeword to the parent node which decodes the message, sums it with its own message and then re-encodes it into a codeword of length $\Theta\left(\log n\right)$. The process continues till the sink node ρ receives the sum of all the input bits in the network. From Theorem 2.2.3, this phase carries a probability of error at most $O(1/n)$. It requires $O\left(n \log n/r^2\right)$ transmissions in the network and $O\left(\sqrt{n}/r \times \log n\right)$ time slots.

- $r \leq \left(\sqrt{n} \log n\right)^{1/3}$: In this regime, the above simple aggregation scheme does not match the lower bound for the latency in Theorem 2.4.3. A more sophisticated aggregation scheme is presented in [11, Section V], which uses ideas from [16] to efficiently simulate a scheme for noiseless networks in noisy networks. The phase carries a probability of error at most $O(1/n)$. It requires $O\left(n \log n/r^2\right)$ transmissions in the network and $O\left(\sqrt{n}/r\right)$ time slots.

Combining the two phases, the above scheme can compute any symmetric function with probability of error at most $O(1/n)$. Thus for n large enough, we have a δ-error scheme for computing any symmetric function in the network $\mathcal{N}\left(n, r\right)$. It requires at most $\max\{O\left(\sqrt{n}/r\right), O\left(r^2 \log \log n\right)\}$ time slots and
$\max\{O\left(n \log n/r^2\right), O\left(n \log \log n\right)\}$ transmissions. ■

2.5 General Network Topologies

In the previous sections, we focused on grid geometric networks for their suitable regularity properties and for ease of exposition. The extension to random geometric networks in the continuum plane when $r = \Omega\left(\sqrt{\log n}\right)$ is immediate, and we focus here on extensions to more general topologies. First, we discuss extensions of our schemes for computing symmetric functions and then present a generalized lower bound on the number of transmissions required to compute symmetric functions in arbitrary connected networks.

2.5.1 Computing symmetric functions in noiseless networks

One of the key components for efficiently computing symmetric functions in noiseless networks in Theorem 2.3.4 was the hierarchical scheme proposed for computing the arithmetic sum function in the grid geometric network $\mathcal{N}(n, 1)$. The main idea behind the scheme was to consider successively coarser partitions of the network and at any given level aggregate the partial sum of the input messages in each individual cell of the partition using results from the finer partition in the previous level of the hierarchy. Using this idea we extend the hierarchical scheme to any connected noiseless network $\mathcal{N}$ and derive an upper bound on the number of transmissions required for the scheme. Let each node in the network start with an input bit and denote the set of nodes by $\mathcal{V}$. The scheme is defined by the following parameters:

- The number of levels h.

- For each level i, a partition $\Pi_i = \{P_i^1, P_i^2, \ldots, P_i^{s_i}\}$ of the set of nodes in the network $\mathcal{V}$ into s_i disjoint cells such that each $P_i^j = \cup_{k \in T_i^j} P_{i-1}^k$ where $T_i^j \subseteq \{1, 2, \ldots, s_{i-1}\}$, i.e., each cell is composed of one or more cells from the next lower level in the hierarchy. See Figure 2.9 for an illustration. Here, $\Pi_0 = \{\{i\} : i \in \mathcal{V}\}$ and $\Pi_h = \{\mathcal{V}\}$.

- For each cell P_i^j, a designated cell-center $c_i^j \in P_i^j$. Let c_h^1 be the designated sink node v^*.

- For each cell P_i^j, let S_i^j denote a Steiner tree with the minimum number of edges which connects the corresponding cell-center with all the cell-centers of its component cells P_{i-1}^k, i.e., the set of nodes $\cup_{k \in T_i^j} c_{i-1}^k \cup c_i^j$. Let l_i^j denote the number of edges in S_i^j.

Using the above definitions, the hierarchical scheme from Theorem 2.3.4 can now be easily extended to general network topologies. We start with the first level in the hierarchy and then proceed recursively. At any given level, we compute the partial sums of the input messages in each individual cell of the partition at the corresponding cell-centers by aggregating the results from the previous level along the minimum Steiner tree. It is easy to verify that after the final level in the scheme, the sink node v^* possesses the arithmetic sum of all the input messages in the network $\mathcal{N}$. The total number

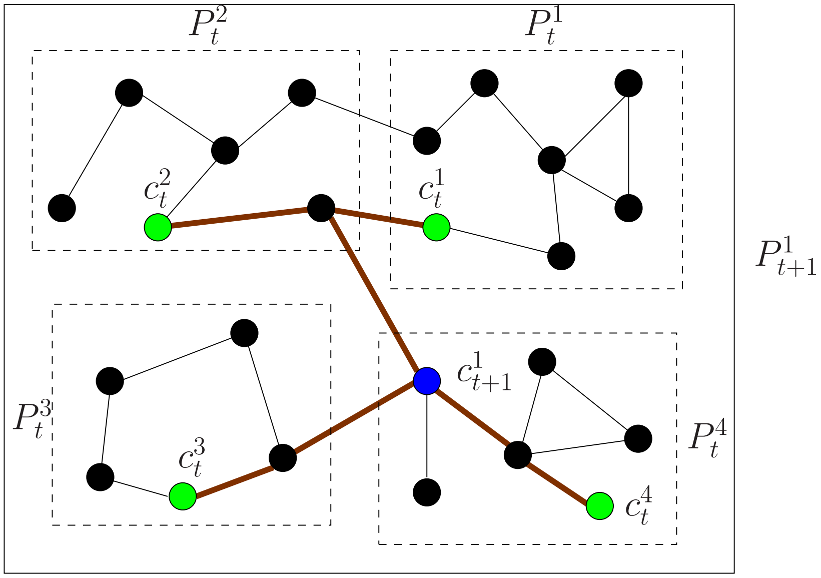


Figure 2.9: Cell P_{t+1}^1 is composed of $\{P_t^k\}_{k=1}^4$ smaller cells from the previous level in the hierarchy. Each of the cell-centers c_t^k (denoted by the green nodes) holds the sum of the input bits in the corresponding cell P_t^k. These partial sums are aggregated along the minimum Steiner tree S_{t+1}^1 (denoted by the brown bold lines) so that the cell-center c_{t+1}^1 (denoted by the blue node) can compute the sum of all the input bits in P_{t+1}^1.

of transmissions made by the scheme is at most

$$\sum_{t=0}^{h-1} \sum_{j=1}^{s_{t+1}} l_{t+1}^j \cdot \log\left(\left|P_{t+1}^j\right|\right) .$$

Thus, we have a scheme for computing the arithmetic sum function in any arbitrary connected network. In the proof of Theorem 2.3.4, the above bound is evaluated for the grid geometric network $\mathcal{N}(n, 1)$ with $h = \log\sqrt{n}$, $s_t = n/2^{2t}$, $l_t^j \leq 4 \cdot 2^{t-1}$, $\left|P_t^j\right| = 2^{2t}$, and is shown to be $O(n)$.

2.5.2 Computing symmetric functions in noisy networks

We generalize the scheme in Theorem 2.4.6 for computing symmetric functions in a noisy grid geometric network $\mathcal{N}(n, r)$ to a more general class of network topologies and derive a corresponding upper bound on the number of transmissions required. The original scheme consists of two phases: an intra-cell phase where the network is partitioned into smaller cells, each of which is a clique, and partial sums are computed in each individual cell; and an inter-cell phase where the partial sums in cells are aggregated to compute the arithmetic sum of all input messages at the sink node. We extend the above idea to more general topologies. First, for any $z \geq 1$, consider the following definition:

Clique-cover property $\mathcal{C}(z)$: a network $\mathcal{N}$ of n nodes is said to satisfy the clique-cover property $\mathcal{C}(z)$ if the set of nodes $\mathcal{V}$ is covered by at most $\lfloor n/z \rfloor$ cliques, each of size at most $\log n / \log \log n$.

For example, a grid geometric network $\mathcal{N}(n, r)$ with $r = O(\sqrt{\log n / \log \log n})$ satisfies $C(z)$ for $z = O(r^2)$. On the other hand, a tree network satisfies $C(z)$ only for $z \leq 2$. Note that any connected network satisfies property $C(1)$. By regarding each disjoint clique in the network as a cell, we can easily extend the analysis in Theorem 2.4.6 to get the following result, whose proof is omitted.

Theorem 2.5.1. *Let $\delta \in \left(0, \frac{1}{2}\right)$, $\epsilon \in \left(0, \frac{1}{2}\right)$ and $\mathcal{N}$ be any connected network of n nodes with $n \geq 2/\delta$. For $z \geq 1$, if $\mathcal{N}$ satisfies $\mathcal{C}(z)$, then there exists a δ-error scheme for computing any symmetric function over $\mathcal{N}$ which requires at most $O(n \log n / z)$ transmissions.*

2.5.3 A generalized lower bound for symmetric functions

The proof techniques that we use to obtain lower bounds are also applicable to more general network topologies. Recall that $N(i)$ denotes the set of neighbors for any node i. For any network, define the average degree as

$$d(n) = \frac{\sum_{i \in \mathcal{V}} |N(i)|}{n}.$$

A slight modification to the proof of Theorem 2.4.3 leads to the following result:

Theorem 2.5.2. *Let $\delta \in \left(0, \frac{1}{2}\right)$ and let $\epsilon \in \left(0, \frac{1}{2}\right)$. There exists a symmetric target function f such that any δ-error scheme for computing f over any connected network of n nodes with average degree $d(n)$, requires at least $\Omega\left(\frac{n \log n}{d(n)}\right)$ transmissions.*

Proof. Let f be the parity function. The only difficulty in adapting the proof of Theorem 2.4.3 arises from the node degree not being necessarily the same for all the nodes. We circumvent this problem as follows: in addition to decomposing the network into the set of source nodes σ and auxiliary nodes $\mathcal{A}$, such that $|\sigma| = \alpha n$ for $\alpha \in (0, 1)$, as in the proof of Lemma 2.4.5 (see Appendix 2.2.2); we also let every source node with degree more than $\frac{2d(n)}{\alpha}$ be an auxiliary node. There can be at most $\frac{\alpha n}{2}$ of such nodes in the network since the average degree is $d(n)$. Thus, we obtain an $\left(\frac{\alpha n}{2}, (1 - \frac{\alpha}{2})n\right)$ decomposition of the network such that each source node has degree at most $\frac{2d(n)}{\alpha}$. The rest of the proof then follows in the same way. ■

As an application of the above result, we have the following lower bound for ring or tree networks.

Corollary 2.5.3. *Let f be the parity function, let $\delta \in \left(0, \frac{1}{2}\right)$, and let $\epsilon \in \left(0, \frac{1}{2}\right)$. Any δ-error scheme for computing f over any ring or tree network of n nodes requires at least $\Omega\left(n \log n\right)$ transmissions.*

The above result answers an open question, posed originally by El Gamal [6].

2.6 Conclusion

We conclude with some observations and directions for future work.

2.6.1 Target functions

We considered all symmetric functions as a single class and presented a worst-case characterization (up to a constant) of the number of transmissions and time slots required for computing this class of functions. A natural question to ask is whether it is possible to obtain better performance if one restricts to a particular sub-class of symmetric functions. For example, two sub-classes of symmetric functions are considered in [3]: *type-sensitive* and *type-threshold*. Since the parity function is a type-sensitive function, the characterization for noiseless networks in Theorems 2.3.3 and 2.3.4, as well as noisy broadcast networks in Theorems 2.4.3 and 2.4.6 also holds for the restricted sub-class of type-sensitive functions. A similar general characterization is not possible for type-threshold functions since the trivial function ($f(\mathbf{x}) = 0$ for all $\mathbf{x}$) is also in this class and it requires no transmissions and time slots to compute. The following result, whose proof follows similar lines as the results in previous sections and is omitted, characterizes the number of transmissions and the number of time slots required for computing the maximum function, which is an example type-threshold function. This can be compared with the corresponding results for the whole class of symmetric functions in Theorems 2.4.3 and 2.4.6.

Theorem 2.6.1. *Let f be the maximum function. Let $\delta \in (0, 1/2)$, $\epsilon \in (0, 1/2)$, and $r \in [1, \sqrt{2n}]$. Any δ-error scheme for computing f over an ϵ-noise network $\mathcal{N}(n, r)$ requires at least $\max\{\Omega(\sqrt{n}/r), \Omega(r^2)\}$ time slots and $\max\{\Omega(n \log n/r^2), \Omega(n)\}$ transmissions. Further, there exists a δ-error scheme for computing f which requires at most $\max\{O(\sqrt{n}/r), O(r^2)\}$ time slots and $\max\{O(n \log n/r^2), O(n)\}$ transmissions.*

2.6.2 On the role of ϵ and δ

Throughout the paper, the channel error parameter ϵ and the threshold δ on the probability of error are taken to be given constants. It is also interesting to study how the cost of computation depends on these parameters. The careful reader might have noticed that our proposed schemes work also when only an upper bound on the channel error parameter ϵ is considered, and always achieve a probability of error δ that is either zero or tends to zero as $n \to \infty$. It is also clear that the cost of computation should decrease with smaller values of ϵ and increase with smaller values of δ. Indeed, from (2.1) in the proof of Theorem 2.4.3 we see that the lower bound on the number of transmissions required for computing the parity function depends on ϵ as $1/(-\log \epsilon)$. On the other hand, from the proof of Theorem 2.4.6 the upper bound on the number of transmissions required to compute any symmetric function depends on ϵ as $1/\left(-\log(\epsilon(1-\epsilon))\right)$. The two expressions are close for small values of ϵ.

2.6.3 Network models

We assumed that each node in the network has a single bit value. Our results can be immediately adapted to obtain upper bounds on the latency and number of transmissions required for the more general scenario where each node i observes a block of input messages $x_i^1, x_i^2, \ldots, x_i^k$ with each $x_i^j \in \{0, 1, \ldots, q\}$, $q \geq 2$. However, finding matching lower bounds seems to be more challenging.

Appendix

2.1 Computing the arithmetic sum over $\mathcal{N}(n, 1)$

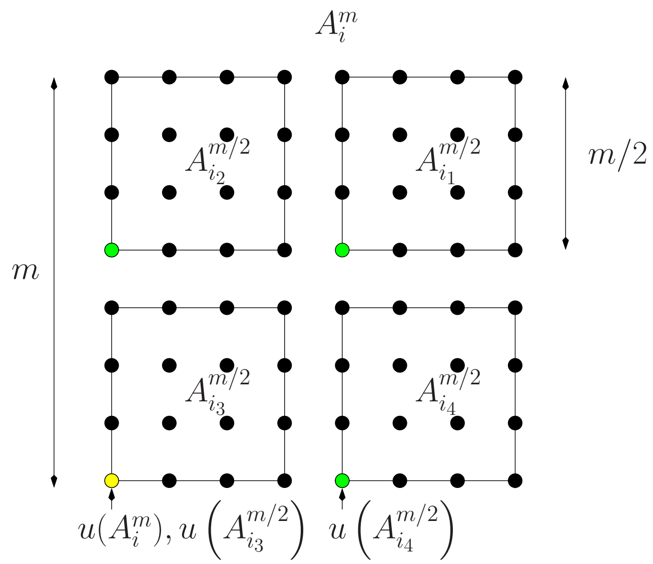


Figure 2.10: A_i^m is a square cell of size $m \times m$. This figure illustrates some notation with regards to A_i^m.

Consider a noiseless network $\mathcal{N}(n, 1)$ where each node i has an input message $x_i \in \{0, 1, \ldots, q-1\}$. We present a scheme which can compute the arithmetic sum of the input messages over the network in $O(\sqrt{n} + \log q \cdot \log n)$ time slots and using $O(n + \log q)$ transmissions. We briefly present the main idea of the scheme before

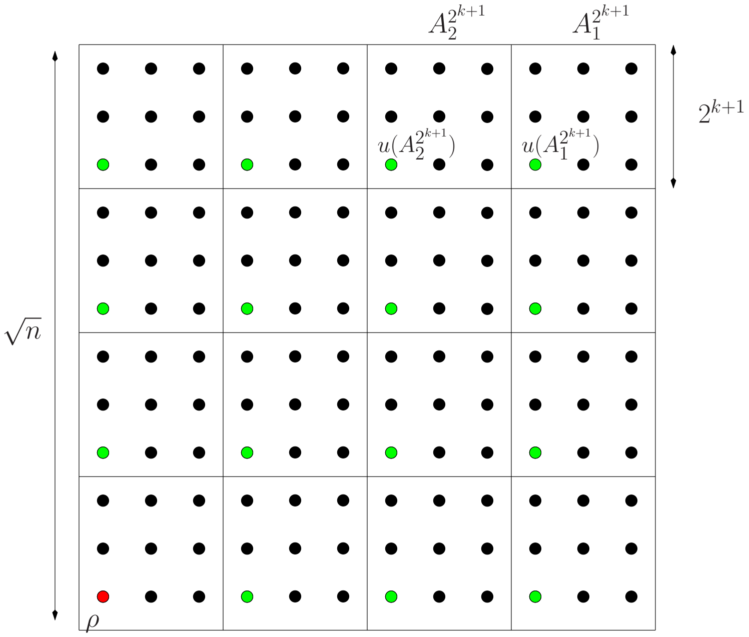


Figure 2.11: This figure illustrates the partition of the network $\mathcal{N}(n,1)$ into smaller cells $\left\{A_i^{2^{k+1}}\right\}_{i=1}^{\frac{n}{2^{2(k+1)}}}$, each of size $2^{k+1} \times 2^{k+1}$.

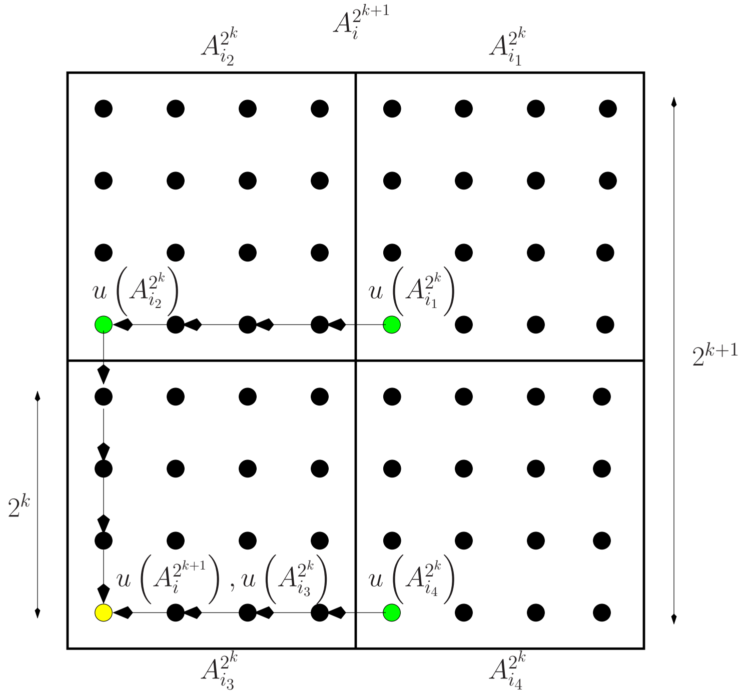
delving into details. Our scheme divides the network into small cells and computes the sum of the input messages in each individual cell at designated cell-centers. We then proceed recursively and in each iteration we double the size of the cells into which the network is partitioned and compute the partial sums by aggregating the computed values from the previous round. This process finally yields the arithmetic sum of all the input messages in the network.

Before we describe the scheme, we define some notation. Consider an $m \times m$ square cell in the network, see Figure 2.10. Denote this cell by A_i^m and the node in the lower-left corner of A_i^m by $u\left(A_i^m\right)$. For any m which is a power of 2, $m \geq 2$, A_i^m can be divided into 4 smaller cells, each of size $m/2 \times m/2$, see Figure 2.10. Denote these cells by $\left\{A_{i_j}^{m/2}\right\}_{j=1}^{4}$.

Without loss of generality, let n be a power of 4. The scheme has the following steps :

1. Let $k = 0$.

2. Consider the partition of the network into cells $\left\{A_i^{2^{k+1}}\right\}_{i=1}^{\frac{n}{2^{2(k+1)}}}$ each of size $2^{k+1} \times$



$A_{i}^{2^{k+1}}$
$A_{i_2}^{2^k}$
$A_{i_1}^{2^k}$
$u\left(A_{i_2}^{2^k}\right)$
$u\left(A_{i_1}^{2^k}\right)$
2^{k+1}
2^k
$u\left(A_{i}^{2^{k+1}}\right), u\left(A_{i_3}^{2^k}\right)$
$u\left(A_{i_4}^{2^k}\right)$
$A_{i_3}^{2^k}$
$A_{i_4}^{2^k}$

Figure 2.12: Step 2 of the scheme for computing the sum of input messages. The network is divided into smaller cells, each of size $2^{k+1} \times 2^{k+1}$. For any such cell $A_{i}^{2^{k+1}}$, $j \in \{1,2,3,4\}$, each corner node $u\left(A_{i_j}^{2^k}\right)$ has the sum of the input messages corresponding to the nodes in the cell $A_{i_j}^{2^k}$. Then the sum of the input messages corresponding to the cell $A_{i}^{2^{k+1}}$ is aggregated at $u\left(A_{i}^{2^{k+1}}\right)$, along the tree shown in the figure.

2^{k+1}, see Figure 2.11. Note that each cell $A_{i}^{2^{k+1}}$ consists of exactly four cells $\left\{A_{i_1}^{2^k}, \ldots, A_{i_4}^{2^k}\right\}$, see Figure 2.12. Each corner node $u\left(A_{i_j}^{2^k}\right), j = 1,2,3,4$ possesses the sum of the input messages corresponding to the nodes in the cell $A_{i_j}^{2^k}$.

The partial sums stored at $u\left(A_{i_j}^{2^k}\right), j = 1,2,3,4$ are aggregated at the node $u\left(A_{i}^{2^{k+1}}\right)$, along the tree shown in Figure 2.12. Each node in the tree makes at most $\log\left(2^{2(k+1)}q\right)$ transmissions.

At the end of this step, each corner node $u\left(A_{i}^{2^{k+1}}\right)$ has the sum of the input messages corresponding to the nodes in the cell $A_{i}^{2^{k+1}}$. By pipelining the transmissions along the tree, this step takes at most

$$2\left(2^k + \log\left(2^{2(k+1)}q\right)\right) \text{ time slots.}$$

The total number of transmissions in the network for this step is at most

$$\frac{n}{2^{2(k+1)}} \cdot 4 \cdot 2^k \cdot \log\left(2^{2(k+1)} q\right) = \frac{4n}{2^{k+1}} \left(k + 1 + \log q\right).$$

3. Let $k \leftarrow k+1$. If $2^{k+1} \leq \sqrt{n}$, return to step 2, else terminate.

Note that at the end of the process, the node ρ can compute the sum of the input messages for any input $\mathbf{x} \in \{0, 1, \ldots, q-1\}^n$. The total number of steps in the scheme is $\log\sqrt{n}$. The number of time slots that the scheme takes is at most

$$\begin{aligned} & \sum_{k=0}^{\log\sqrt{n}-1} 2\left(2^k + \log\left(2^{2(k+1)} q\right)\right) \\ \leq \quad & O\left(\log q \cdot \log n + \sqrt{n}\right). \end{aligned}$$

The total number of transmissions made by the scheme is at most

$$\begin{aligned} & \sum_{k=0}^{\log\sqrt{n}-1} \frac{4n(k + 1 + \log q)}{2^{k+1}} \\ \leq \quad & O\left(n + \log q\right). \end{aligned}$$

2.2 Completion of the proof of Theorem 2.4.3

2.2.1 Proof of Lemma 2.4.4

For every $i \in \{1, 2, \ldots, n\}$, let y_i be the noisy copy of x_i that the auxiliary node A^* receives. Denote the received vector by $\mathbf{y}$. The objective of A^* is to compute the parity of the input bits $x_1, x_2, \ldots, x_n$. Thus, the target function f is defined as

$$f\left(\mathbf{x}\right) = x_1 \oplus x_2 \oplus \ldots \oplus x_n.$$

Since the input $\mathbf{x}$ is uniformly distributed, we have $\mathsf{Pr}\left(f(\mathbf{x}) = 0\right) = \mathsf{Pr}\left(f(\mathbf{x}) = 1\right) = 1/2$. In the following, we first show that Maximum Likelihood estimation is equivalent to using the parity of the received bits $y_1, y_2, \ldots, y_n$ i.e. $f(\mathbf{y})$ as an estimate for $f(\mathbf{x})$, and then compute the corresponding probability of error. From the definition of Maximum

Likelihood estimation, we have

$$\hat{f} = \begin{cases} 1 & \text{if } \mathsf{Pr}\left(\mathbf{y}|f(\mathbf{x})=1\right) > \mathsf{Pr}\left(\mathbf{y}|f(\mathbf{x})=0\right) \\ 0 & \text{otherwise .} \end{cases}$$

Next,

$$\begin{aligned} \mathsf{Pr}\left(\mathbf{y}|f(\mathbf{x})=1\right) &= \sum_{\substack{\mathbf{x}\in\{0,1\}^n \\ \text{s.t } f(\mathbf{x})=1}} \mathsf{Pr}(\mathbf{x}|f(\mathbf{x}=1)) \cdot \mathsf{Pr}\left(\mathbf{y}|\mathbf{x}\right) \\ &\overset{(a)}{=} \kappa \sum_{\substack{\mathbf{x}\in\{0,1\}^n \\ \text{s.t } f(\mathbf{x})=1}} \mathsf{Pr}\left(y_1|x_1\right) \mathsf{Pr}\left(y_2|x_2\right) \ldots \mathsf{Pr}\left(y_n|x_n\right) \end{aligned}$$

where $\kappa = 2^{-(n-1)}$ and (a) follows since $\mathbf{x}$ is uniformly distributed over $\{0,1\}^n$ and from the independence of the channels between the sources and the auxiliary node. Similarly,

$$\mathsf{Pr}\left(\mathbf{y}|f(\mathbf{x})=0\right) = \kappa \sum_{\substack{\mathbf{x}\in\{0,1\}^n \\ \text{s.t } f(\mathbf{x})=0}} \mathsf{Pr}\left(y_1|x_1\right) \mathsf{Pr}\left(y_2|x_2\right) \ldots \mathsf{Pr}\left(y_n|x_n\right).$$

Putting things together, we have

$$\begin{aligned} \mathsf{Pr}\left(\mathbf{y}|f(\mathbf{x})=0\right) - \mathsf{Pr}\left(\mathbf{y}|f(\mathbf{x})=1\right) &= \kappa \left(\sum_{\substack{\mathbf{x}\in\{0,1\}^n \\ \text{s.t } f(\mathbf{x})=0}} \prod_{i=1}^{n} \mathsf{Pr}\left(y_i|x_i\right) - \sum_{\substack{\mathbf{x}\in\{0,1\}^n \\ \text{s.t } f(\mathbf{x})=1}} \prod_{i=1}^{n} \mathsf{Pr}\left(y_i|x_i\right) \right) \\ &\overset{(a)}{=} \kappa \prod_{i=1}^{n} \left(\mathsf{Pr}(y_i|x_i=0) - \mathsf{Pr}(y_i|x_i=1) \right) \\ &= \kappa \left(-1\right)^{n_1(\mathbf{y})} \left(1-2\epsilon\right)^n, \end{aligned} \tag{2.2}$$

where $n_1(\mathbf{y})$ is the number of components in $\mathbf{y}$ with value 1. The above equality (a) can be verfied by noting that the product in (a) produces a sum of 2^n monomials and that the sign of each monomial is positive if the number of terms of the monomial conditioned on $x_i = 1$ is even, and negative otherwise. From (2.2), we now have

$$\mathsf{Pr}\left(\mathbf{y}|f(\mathbf{x})=0\right) - \mathsf{Pr}\left(\mathbf{y}|f(\mathbf{x})=1\right) \begin{cases} > 0 & \text{if } f(\mathbf{y}) = 0 \\ < 0 & \text{if } f(\mathbf{y}) = 1. \end{cases}$$

Thus, we have shown that Maximum Likelihood estimation is equivalent to using $f(\mathbf{y})$ as an estimate for $f(\mathbf{x})$. The corresponding probability of error is given by

$$\begin{aligned} \mathsf{Pr}_{ML}\left(\text{Error}\right) &= \mathsf{Pr}\left(f(\mathbf{y}) \neq f(\mathbf{x})\right) \\ &= \sum_{j=1}^{\lceil \frac{n}{2} \rceil} \mathsf{Pr}\left(\sum_{i=1}^{n} x_i \oplus y_i = 2j-1\right) \\ &= \sum_{j=1}^{\lceil \frac{n}{2} \rceil} \binom{n}{2j-1} \epsilon^{2j-1} \left(1-\epsilon\right)^{n-2j+1} \\ &= \frac{1-(1-2\epsilon)^n}{2}. \end{aligned}$$

Hence, for any scheme $\mathcal{P}_1$ which computes the parity function f in an n-noisy star network, the probability of error is at least $\left(1-(1-2\epsilon)^n\right)/2$.

2.2.2 Proof of Lemma 2.4.5

We borrow some notation from [8] and [12]. Consider the nodes in a network and mark a subset σ of them as input nodes and the rest $\mathcal{A}$ as auxiliary nodes. Such a decomposition of the network is called an $(|\sigma|, |\mathcal{A}|)$-decomposition. An input value to this network is an element of $\{0,1\}^{|\sigma|}$. Consider a scheme $\mathcal{P}$ on such a network which computes a function f of the input. The scheme is said to be m-bounded with respect to an $(|\sigma|, |\mathcal{A}|)$-decomposition if each node in σ makes at most m transmissions. Recall from Section 2.2 that for any scheme in our model, the number of transmissions that any node makes is fixed a priori and does not depend on a particular execution of the scheme. Following [8] and [12] we define the *semi-noisy network*, in which whenever it is the turn of an input node to transmit, it sends its input bit whose independent ϵ-noisy copies are received by its neighbors, while the transmission made by auxiliary nodes are not subject to any noise.

The proof now proceeds by combining three lemmas. Suppose there exists a δ-error scheme $\mathcal{P}$ for computing the parity function in an ϵ-noise network $\mathcal{N}(n, r)$ which requires S transmissions. We first show in Lemma 2.2.1 that this implies the existence of a suitable decomposition of the network and a δ-error, $O(S/n)$-bounded scheme $\mathcal{P}_d$ for computing the parity function in this decomposed network. Lemma 2.2.2 translates the scheme $\mathcal{P}_d$ into a scheme $\mathcal{P}_s$ for computing in a semi-noisy network and Lemma 2.2.3 translates $\mathcal{P}_s$ into a scheme $\mathcal{P}_1$ for computing in a noisy star network, while ensuring that the probability of error does not increase at any intermediate step. The proof is completed using the fact that the probability of error for original scheme $\mathcal{P}$ is at most δ.

Let $\alpha \in (0, 1)$. We have the following lemma.

Lemma 2.2.1. *If there is a δ-error scheme $\mathcal{P}$ for computing the parity function (on n input bits) in $\mathcal{N}(n, r)$ with S transmissions, then there is an $(\alpha n, (1-\alpha)n)$-decomposition of $\mathcal{N}(n, r)$ and a δ-error, $O(S/n)$-bounded scheme $\mathcal{P}_d$ for computing the parity function (on αn bits) in this decomposed network.*

Proof. If all nodes in the network make $O(S/n)$ transmissions, then the lemma follows trivially. Otherwise, we decompose the network into the set of input nodes σ and auxiliary nodes $\mathcal{A}$ as follows. Consider the set of nodes which make more than $\frac{S}{n(1-\alpha)}$ transmissions each during the execution of the scheme $\mathcal{P}$. Since $\mathcal{P}$ requires S transmissions, there can be at most $(1-\alpha)n$ of such nodes. We let these nodes be auxiliary nodes and let their input be 0. Thus, we have an $(\alpha n, (1-\alpha)n)$-decomposition of the network $\mathcal{N}(n, r)$. The scheme now reduces to computing the parity (on αn bits) over this decomposed network. By construction, each input node makes at most $\frac{S}{n(1-\alpha)}$ transmissions and hence the scheme is $O(S/n)$-bounded. ■

The following lemma is stated without proof, as it follows immediately from [8, Section 6, page 1833], or [12, Lemma 5.1, page 1064].

Lemma 2.2.2. (FROM NOISY TO SEMI-NOISY) *For any function $f : \{0, 1\}^{\alpha n} \to \{0, 1\}$ and any δ-error, $O(S/n)$-bounded scheme $\mathcal{P}_d$ for computing f in an $(\alpha n, (1-\alpha)n)$-decomposition of $\mathcal{N}(n, r)$, there exists an $(\alpha n, n)$-decomposed semi-noisy network of $(1+\alpha)n$ nodes such that each input node has at most $O(r^2)$ neighbors and a δ-error, $O(S/n)$-bounded scheme $\mathcal{P}_s$ for computing f in the semi-noisy network.*

We now present the final lemma needed to complete the proof.

Lemma 2.2.3. *(FROM SEMI-NOISY TO NOISY STAR) For any function $f : \{0,1\}^{\alpha n} \to \{0,1\}$ and any δ-error, $O(S/n)$-bounded scheme $\mathcal{P}_s$ for computing f in an $(\alpha n, n)$-decomposed semi-noisy network where each input node has at most $O(r^2)$ neighbors, there exists a δ-error scheme $\mathcal{P}_1$ for computing f in an αn-noisy star network with noise parameter $\epsilon^{O(Sr^2/n)}$, with each input node transmitting its input bit only once.*

Proof. In a semi-noisy network, when it is the turn of an input node to transmit during the execution of $\mathcal{P}_s$, it transmits its input bit. Since the bits sent by the input nodes do not depend on bits that these nodes receive during the execution of the scheme, we can assume that the input nodes make their transmissions at the beginning of the scheme an appropriate number of times, and after that only the auxiliary nodes communicate without any noise. Further, since any input node in the $(\alpha n, n)$-decomposed network has at most $O(r^2)$ neighbors, at most $O(r^2)$ auxiliary nodes receive independent ϵ-noisy copies of each such input bit. Since $\mathcal{P}_s$ is an $O(S/n)$-bounded scheme, each input node makes at most $O(S/n)$ transmissions and hence the auxiliary nodes receive a total of at most $O(Sr^2/n)$ independent ϵ-noisy copies of each input bit.

Next, we use the scheme $\mathcal{P}_s$ to construct a scheme $\mathcal{P}_1$ for computing f in an αn-noisy star network of noise parameter $\epsilon^{O(Sr^2/n)}$ with each input node transmitting its input bit only once. Lemma 2.2.4 shows that upon receiving an $\epsilon^{O(Sr^2/n)}$-noisy copy for every input bit, the auxiliary node A^* in the noisy star network can generate $O(Sr^2/n)$ independent ϵ-noisy copies for each input bit. Then onwards, the auxiliary node A^* can simulate the scheme $\mathcal{P}_s$. This is true since for $\mathcal{P}_s$ only the auxiliary nodes operate after the initial transmissions by the input nodes, and their transmissions are not subject to any noise. ■

Lemma 2.2.4. *[8, Lemma 36, page 1834] Let $t \in \mathbb{N}$, $\epsilon \in (0, 1/2)$, and $\gamma = \epsilon^t$. There is a randomized algorithm that takes as input a single bit b and outputs a sequence of t bits such that if the input is a γ-noisy copy of 0 (respectively of 1), then the output is a sequence of independent ϵ-noisy copies of 0 (respectively of 1).*

2.3 Scheme for computing partial sums at cell-centers

We describe an adaptation of the scheme in [10, Section III], which requires at most $O\left(\frac{n\log n}{r^2}\right)$ transmissions and $O(\log n)$ time slots. The scheme in [10, Section III] is described for $r > \sqrt{\log n}$ and while the same ideas work for $r \leq \sqrt{\log n/\log\log n}$, the parameters need to be chosen carefully so that the scheme can compute efficiently in the new regime.

Recall that the network is partitioned into cells of size $c \times c$ where $c = r/\sqrt{8}$. Consider any cell A_1 in the network and denote its cell-center by v_1. The scheme has the following steps:

1. Every node in A_1 takes a turn to transmit its input bit x_i, $\frac{6}{\lambda}\left(\frac{\log n}{c^2}\right)$ times, where $\lambda = -\ln\left(4\epsilon(1-\epsilon)\right)$. Thus, every node in A_1 receives $\frac{8}{\lambda}\left(\frac{\log n}{c^2}\right)$ independent noisy copies of the entire input. This step requires $n \cdot \frac{8}{\lambda}\left(\frac{\log n}{c^2}\right)$ transmissions and $\frac{8\log n}{\lambda}$ time slots.

2. Each node in A_1 forms an estimate for the input bits of the other nodes in A_1 by taking the *majority* of the noisy bits that it received from each of them. It is easy to compute the probability that a node has a wrong estimate for any given input bit to be at most $c^2 \cdot e^{-\frac{4\log n}{c^2}}$, see for example Gallager's book [15, page 125]. Each node then computes the arithmetic sum of all the decoded bits and thus has an estimate of the sum of all the input bits in the cell A_1.

3. Each node in A_1 transmits its estimate to the cell-center v_1 using a codeword of length $\frac{k\log n}{c^2}$ such that k is a constant and the cell-center decodes the message with probability of error at most $e^{-\frac{4\log n}{c^2}}$. The existence of such a code is guaranteed by Theorem 2.2.3 and from the fact that the size of the estimate in bits $\log(c^2+1) \leq \log n/c^2$, since $c^2 = \frac{r^2}{8} \leq \frac{\log n}{8\log\log n}$. At the end of this step, v_1 has c^2 independent estimates for the sum of the input bits corresponding to the nodes in A_1. The total number of transmissions for this step is at most $n \cdot \frac{k\log n}{c^2}$ and it requires at most $k\log n$ time slots.

4. The cell-center v_1 takes the *mode* of these c^2 values to make the final estimate for the sum of the input bits in A_1.

We can now bound the probability of error for the scheme as follows.

$$\begin{aligned}\mathsf{Pr}(\text{Error}) &\leq \left(4\left(c^2 e^{-\frac{4\log n}{c^2}} + e^{-\frac{4\log n}{c^2}}\right)\left(1-(c^2 e^{-\frac{4\log n}{c^2}} + e^{-\frac{4\log n}{c^2}})\right)\right)^{c^2} \\ &\overset{(a)}{\leq} \left(4\cdot 9c^2 e^{-\frac{4\log n}{c^2}}\right)^{c^2} \\ &\overset{(b)}{\leq} \frac{1}{n^2}, \text{ for } n \text{ large enough}\end{aligned}$$

where (a) follows since $8c^2 = r^2 \geq 1$; and (b) follows since $c^2 \log c^2 \leq \log n$. Thus, every cell-center v_i can compute the sum of the input bits corresponding to the nodes in A_i with probability of error at most $\frac{1}{n^2}$. The total probability of error is then at most $\frac{n}{c^2} \cdot \frac{1}{n^2} \leq \frac{1}{n}$. The total number of transmissions in the network for this scheme is at most $\left(\frac{8}{\lambda}+k\right) \cdot \frac{n\log n}{c^2}$, i.e. $O\left(\frac{n\log n}{r^2}\right)$ and it takes at most $\left(\frac{8}{\lambda}+k\right) \cdot \log n$, i.e., $O\left(\log n\right)$ time slots.

Chapter 2, in part, has been submitted for publication of the material. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Function computation over linear channels

We consider multiple sources communicating over a network to a common sink. We assume that the network operation is fixed, and its end result is to convey a fixed unkown linear transformation of the source data to the sink. We design communication protocols that can perform computation without modifying the network operation, by appropriately selecting the codebook that the sources employ to map their measurements to the data they send over the network. The model studied in this chapter is motivated by practical considerations: since implementing networking protocols is hard, there is a strong incentive to reuse the same network protocol to compute different functions.

3.1 Introduction

In sensor networks, the need for energy efficiency has stimulated research efforts towards in-network aggregation and function computation, see for example [3, 17, 18]. Recent work [19, 20] has also pointed out the need to have *simple* coding schemes, since "systems are hard to develop and debug". They advocate a solution where most nodes in the network perform the same operations regardless of the function to be computed, and the onus of guaranteeing successful computation is on a few special nodes that are allowed to vary their operation.

Motivated by the above considerations, we consider the problem of computing functions in a network where multiple sources are connected to a single sink via relays. The sources may have several different possible codebooks, and can select which one to employ depending on the function to be computed. Given a certain target function, each source transmits a codeword corresponding to its observed message. The relay nodes, however, perform the same linear operations, for example randomized network coding (which is a practical and efficient way of transmitting data in a network [21]) irrespective of the target function, i.e., the vectors inserted by the sources are randomly combined and forwarded towards the sink, using linear coefficients that are unknown to both the sources and the sink. The sink then proceeds to evaluate the target function of the source messages.

Following [22–24], we use subspace coding for computing functions in our network model. Given a target function, we assume that each source uses a codebook consisting of subspaces. Each source message is mapped to a subspace in the codebook. When a source generates a message, it injects the basis vectors of the corresponding subspace into the network. The network operation is abstracted by assuming that the sink collects enough linear combinations of these vectors to learn the joint span of the injected subspaces. Given this information, the sink then attempts to compute the target function of the source messages. Our objective is to design codebooks which minimize the number of symbols each source needs to transmit, while guaranteeing successful function computation by the sink.

Thus, we envision a network architecture where intermediate network nodes always perform the same operations for information transfer, which leads to a simple implementation. At the same time, the sink has the flexibility to utilize the network to learn different functions of the source data by informing the source nodes to employ the corresponding codebooks. Here we focus on non-coherent communication where we have no knowledge about the network transformation; in [25] we look at the case where this transformation is fixed and known.

We note that a scheme which optimizes the intermediate node operations according to the function to be computed might need fewer transmissions. However, it would be more complex to implement, would require topology knowledge, and might be sensitive to the employed communication protocol. In contrast, our approach is transparent both to the topology and the employed communication protocol: the only requirement we impose is that we gather sufficient linear independent combinations. As a result, our protocol would be very well suited to dynamically changing topologies, and could be applied without change on top of very different communication protocols.

The chapter is organized as follows. Section 3.2 presents the problem formulation. In Section 3.3, we present various lower bounds on the number of symbols each source needs to transmit to evaluate an arbitrary function. In Section 3.4, we discuss various example target functions. In particular, we provide lower bounds as well as near-optimal coding schemes for the *identity*, T*-threshold*, *maximum* and K*-largest values* functions. Finally, in Section 3.3, we present a constructive scheme to evaluate arbitrary functions.

3.2 Problem Formulation and Notation

We consider a set of s sources $\sigma_1, \sigma_2, \ldots, \sigma_s$ connected to a sink ρ via a network $\mathcal{N}$. Each source σ_i is either inactive or observes a message $x_i \in \mathcal{A}$, where $\mathcal{A}$ is a finite alphabet. For ease of notation, when a source σ_i is inactive we will set $x_i = \phi$. The sink needs to compute a *target function* f of the source messages, where f is of the form

$$f \ : \ (\mathcal{A} \cup \{\phi\})^s \longrightarrow \mathcal{B}.$$

Some example target functions are defined below.

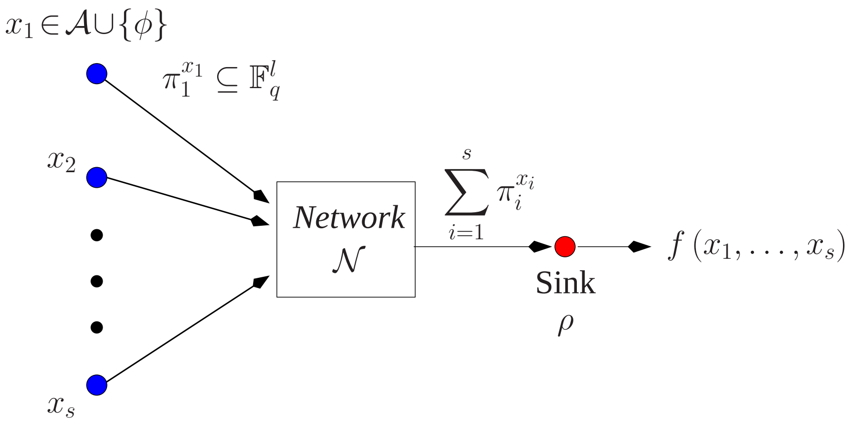

$x_1 \in \mathcal{A} \cup \{\phi\}$
$\pi_1^{x_1} \subseteq \mathbb{F}_q^l$
x_2
x_s
Network
$\mathcal{N}$
$\sum_{i=1}^{s} \pi_i^{x_i}$
Sink
ρ
$f(x_1, \ldots, x_s)$

Figure 3.1: Description of the network operation

Definition 3.2.1.

- The *identity* target function has $\mathcal{B} = (\mathcal{A} \cup \{\phi\})^s$ and is defined by

$$f(x_1, \ldots, x_s) = (x_1, \ldots, x_s).$$

- For $m \geq 1$, the *arithmetic sum* target function has $\mathcal{A} = \{1, \ldots, m\}$, $\mathcal{B} = \{0, 1, \ldots, ms\}$, and is defined by

$$f(x_1, \ldots, x_s) = x_1 + x_2 + \cdots + x_s$$

where '$+$' denotes ordinary integer summation. For any $a \in \mathcal{A} \cup \{\phi\}$, we set $a + \phi = a$.

- Let $\mathcal{A}$ be an ordered set. The *maximum* target function has $\mathcal{B} = \mathcal{A}$ and is defined by

$$f(x_1, \ldots, x_s) = \max\{x_1, \ldots, x_s\}.$$

For any $a \in \mathcal{A} \cup \{\phi\}$, we set $\max\{a, \phi\} = a$.

- The *parity* target function has $\mathcal{A} = \mathcal{B} = \{0, 1\}$, and is defined by

$$f(x_1, \ldots, x_s) = x_1 \oplus x_2 \oplus \ldots \oplus x_s$$

where $\oplus$ denotes mod-2 addition. Again, for any $a \in \mathcal{A} \cup \{\phi\}$ we set $a \oplus \phi = a$.

- The *majority* target function has $\mathcal{A} = \mathcal{B} = \{0, 1\}$, and is defined by

$$f\left(x_1, \ldots, x_s\right) = \begin{cases} 1 & \text{if } \left|\{i : x_i = 1\}\right| > \left|\{i : x_i = 0\}\right| \\ 0 & \text{otherwise.} \end{cases}$$

We consider operation using subspace coding. We denote a subspace by π and the union of two subspaces π_1, π_2 is defined as $\pi_1 + \pi_2 = \{\mathbf{x} + \mathbf{y} : \mathbf{x} \in \pi_1, \mathbf{y} \in \pi_2\}$. The network operates as follows.

- At each source, every alphabet symbol is mapped to a subspace, which serves as the corresponding codeword. Thus, each source σ_i has an associated codebook $\mathcal{C}_i = \left\{\pi_i^j\right\}_{j \in \mathcal{A}}$ where π_i^j is a d-dimensional subspace[1] of an l-dimensional vector space $\mathbb{F}_q^l$ where $d, l \geq 1$ are design parameters. When the source σ_i is active and observes a message $x_i \in \mathcal{A}$, it injects into the network $\mathcal{N}$ a set of d vectors from $\mathbb{F}_q^l$ which span the subspace $\pi_i^{x_i}$. When the source is σ_i inactive, it does not make any transmissions and hence we set $\pi_i^{\phi} = \emptyset$.
- The sink ρ receives from the network $\mathcal{N}$ a set of vectors from $\mathbb{F}_q^l$ which span the union of the input subspaces[2] i.e., ρ observes $\sum_{i=1}^{s} \pi_i^{x_i}$.
- The sink uses the received information to compute the value of $f\left(x_1, x_2, \ldots, x_s\right)$.

A (d, l) *feasible code for computing* f is a collection of codebooks $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_s\}$ such that each π_i^j in the codebooks is a d-dimensional subspace of $\mathbb{F}_q^l$ and the sink can compute the value of $f\left(x_1, x_2, \ldots, x_s\right)$ for any choice of input messages $x_1, x_2, \ldots, x_s$ where each $x_i \in \mathcal{A} \cup \{\phi\}$.

For a (d, l) feasible code for computing f, each source transmits at most $d \cdot l$ symbols from $\mathbb{F}_q$, and we thus consider the associated cost[3] to be $d \cdot l$. Our code design seeks to achieve

$$\mathcal{E}_{\text{min}}(f) = \inf\left\{d \cdot l : \exists \text{ a } (d, l) \text{ feasible code for computing } f\right\}.$$

[1] Although restricting our code design to subspaces of equal dimension may not always be optimal, it significantly simplies the design, and is a standard approach in the literature [22, 26].

[2] In practice, networks operate in rounds. The duration of a round can be chosen large enough to ensure that the sink receives enough linear independent combinations to span the union of the input subspaces.

[3] In this work, the field size q is considered to be fixed and hence not included in the cost.

We begin by showing that for the purpose of minimizing the cost $d \cdot l$, it suffices to consider codes which use one-dimensional subspaces.

Theorem 3.2.2. *Given any (d, l) feasible code for computing a target function f, there also exists a $(1, d \cdot l)$ feasible code for computing f.*

Proof. Partition $(\mathcal{A} \cup \{\phi\})^s$ into $P_1, P_2, \ldots, P_{|\mathcal{B}|}$ so that each P_i consists of all the input vectors which result in the same function value. Then a necessary and sufficient condition for successful computation is that the sink should receive different union subspaces for any two input vectors $\mathbf{x}$ and $\mathbf{y}$ if they belong to distinct partitions.

Let $\{\pi_i^j \subseteq \mathbb{F}_q^l : i \in \{1, \ldots, s\}, j \in \mathcal{A}\}$ denote the collection of d-dimensional subspaces associated with the given (d, l) feasible code for computing f. The above necessary condition implies that these subspaces satisfy a collection of inequalities, each of the form

$$\sum_{i=1}^{s} \pi_i^{a_i} \neq \sum_{i=1}^{s} \pi_i^{b_i} \tag{3.1}$$

where each $a_i, b_i \in \mathcal{A} \cup \{\phi\}$. Now corresponding to each d-dimensional subspace π_i^j in the above code, construct a one-dimensional subspace $\widehat{\pi}_i^j \subseteq \mathbb{F}_q^{d \cdot l}$ by concatenating the d basis vectors of π_i^j into a single vector. It can be verified that the collection of one-dimensional subspaces $\{\widehat{\pi}_i^j : i \in \{1, \ldots, s\}, j \in \mathcal{A}\}$ constructed this way also satisfy all the inequalities that the original code satisfied, i.e., if (3.1) holds for some $\{a_i\}, \{b_i\}$, then

$$\sum_{i=1}^{s} \widehat{\pi}_i^{a_i} \neq \sum_{i=1}^{s} \widehat{\pi}_i^{b_i}. \tag{3.2}$$

Since (3.1) holds, there exists at least one basis vector, say $v_1 \in \pi_1^{a_1}$, which is not in $\sum_{i=1}^{s} \pi_i^{b_i}$. This immediately implies that $\widehat{\pi}_1^{a_1}$ cannot be an element of $\sum_{i=1}^{s} \widehat{\pi}_i^{b_i}$, which proves (3.2).

Thus the collection of one-dimensional subspaces $\{\widehat{\pi}_i^j : i \in \{1, \ldots, s\}, j \in \mathcal{A}\}$ ensures that for any two input vectors in distinct partitions, the sink receives different union subspaces. Since this is sufficient for function computation, we have shown that we can construct a $(1, d \cdot l)$ feasible code from any (d, l) feasible code. ■

In the sequel, we will only consider codes which use one-dimensional subspaces. We will denote the dimension of any subspace π by $\dim(\pi)$. Also, for any vector $\mathbf{x}$, the

j-th component will be denoted by $(\mathbf{x})_j$. Consider a set of indices $I = \left(i_1, i_2, \ldots, i_{|I|}\right) \subseteq \{1, \ldots, s\}$. For any $\mathbf{a} = \left(a_1, a_2, \ldots, a_{|I|}\right) \in (\mathcal{A} \cup \{\phi\})^{|I|}$ and any vector $\mathbf{x} \in (\mathcal{A} \cup \{\phi\})^s$, let $\mathbf{x}(I, \mathbf{a}) = (x_1, x_2, \ldots, x_s)$ denote a vector which is obtained from $\mathbf{x}$ by substituting the components corresponding to the index set I with values from the vector $\mathbf{a}$ and retaining all the other components. That is, for each $j \in \{1, \ldots, |I|\}$, $(\mathbf{x}(I, \mathbf{a}))_{i_j} = (\mathbf{a})_j$ and for each $k \notin I$, $(\mathbf{x}(I, \mathbf{a}))_k = (\mathbf{x})_k$. We conclude this section with a lemma that is often used in the subsequent sections.

Lemma 3.2.3. *If there exist one-dimensional subspaces* $\pi_1, \pi_2, \ldots, \pi_K \subseteq \mathbb{F}_q^l$ *and a subspace* $\pi^* \subseteq \mathbb{F}_q^l$ *such that*

$$\pi_i \nsubseteq \pi^* + \sum_{j<i} \pi_j \quad \forall\, i \in \{1, \ldots, K\} \tag{3.3}$$

then $l \geq K$.

Proof. (3.3) implies that the basis vectors for the K one-dimensional subspaces are linearly independent. The result then follows. ■

3.3 Lower bounds

The number of d-dimensional subspaces of $\mathbb{F}_q^l$ for any $d \leq l, l \geq 1$ is given by the gaussian binomial coefficient

$$\begin{bmatrix} l \\ d \end{bmatrix}_q = \frac{(q^l - 1)(q^{l-1} - 1) \cdots (q^{l-d+1} - 1)}{(q^d - 1)(q^{d-1} - 1) \cdots (q - 1)}. \tag{3.4}$$

We have the following upper bound on the number of d-dimensional subspaces..

Lemma 3.3.1. *The number of* d*-dimensional subspaces of* $\mathbb{F}_q^l$ *is at most* $4q^{d(l-d)}$ *[22, Lemma 4].*

Recall that $\mathbf{x}(I, \mathbf{a})$ denotes a vector which is obtained from $\mathbf{x}$ by substituting the components corresponding to the index set I with values from the vector $\mathbf{a}$ and retaining all the other components. Consider a target function with the following property.

Function property $\mathbf{P}$: There exists $k \in \{1, \ldots, s\}$ and $\mathbf{x} \in (\mathcal{A} \cup \{\phi\})^s$ such that for any $a, b \in \mathcal{A}$,

$$f(\mathbf{x}(\{k\}, a)) \neq f(\mathbf{x}(\{k\}, b)).$$

Examples : The identity function and arithmetic sum function satisfy property $\mathbf{P}$.

We have the following simple lower bound for functions which satisfy property $\mathbf{P}$.

Lemma 3.3.2. *For any target function f which satisfies property $\mathbf{P}$,*

$$\mathcal{E}_{\mathit{min}}(f) \geq \log_q \left(|\mathcal{A}| \, (q-1) \right) + 1.$$

Proof. From the definition of property $\mathbf{P}$, there exists k such that source σ_k must assign a distinct one-dimensional subspace to each $a \in \mathcal{A}$. From (3.4), we have

$$\frac{q^l - 1}{q - 1} \geq |\mathcal{A}|$$
$$\Longrightarrow l \geq \log_q \left(|\mathcal{A}| \, (q-1) \right) + 1.$$

■

Consider a target function with the following property.

Function property $\mathbf{Q}(k)$: There exists $\mathbf{x} \in (\mathcal{A} \cup \{\phi\})^s$ and a collection of k distinct indices $i_1, i_2, \ldots, i_k$ such that for every $j \in \{1, 2, \ldots, k\}$, we have $(\mathbf{x})_j \neq \phi$ and

$$f\left(\mathbf{x}(\{i_1, \ldots, i_{j-1}\}, \{\phi, \ldots, \phi\})\right) \neq f\left(\mathbf{x}(\{i_1, \ldots, i_{j-1}, i_j\}, \{\phi, \ldots, \phi\})\right). \tag{3.5}$$

Example 3.3.3.

- The identity function satisfies property $\mathbf{Q}(s)$ by choosing each $(\mathbf{x})_j$ equal to any element of the alphabet $\mathcal{A}$.
- The arithmetic sum function satisfies property $\mathbf{Q}(s)$ by choosing each $(\mathbf{x})_j$ equal to some non-zero element of the alphabet $\mathcal{A}$.
- The parity function satisfies property $\mathbf{Q}(s)$ by choosing each $(\mathbf{x})_j$ equal to 1.

- The majority function satisfies property $\mathbf{Q}(s)$ by choosing $(\mathbf{x})_j$ equal to 1 if j is even and 0 otherwise when s is even and vice-versa when s is odd.

Lemma 3.3.4. *For any target function f which satisfies property $\mathbf{Q}(k)$,*

$$\mathcal{E}_{\textit{min}}(f) \geq k.$$

Proof. Let

$$\Pi^c = \sum_{t \notin \{i_1, i_2, \ldots, i_k\}} \pi_t^{(\mathbf{x})_t}.$$

From (3.5), any feasible code for computing f should satisfy for every $j \in \{1, 2, \ldots, k\}$

$$\begin{aligned} \pi_{i_j}^{(\mathbf{x})_{i_j}} + \sum_{m=j+1}^{k} \pi_{i_m}^{(\mathbf{x})_{i_m}} + \Pi^c &\neq \sum_{m=j+1}^{k} \pi_{i_m}^{(\mathbf{x})_{i_m}} + \Pi^c \\ \Longrightarrow \pi_{i_j}^{(\mathbf{x})_{i_j}} \nsubseteq \sum_{m=j+1}^{k} \pi_{i_m}^{(\mathbf{x})_{i_m}} + \Pi^c. \end{aligned}$$

Then using Lemma 3.2.3 for the collection of k one-dimensional subspaces $\pi_{i_1}^{(\mathbf{x})_{i_1}}, \ldots, \pi_{i_k}^{(\mathbf{x})_{i_k}}$ and the subspace Π^c, the result follows. ■

We borrow the following definition from [18].

Definition 3.3.5. For any target function $f : (\mathcal{A} \cup \{\phi\})^s \longrightarrow \mathcal{B}$, any index set $I \subseteq \{1, 2, \ldots, s\}$, and any $\mathbf{a}, \mathbf{b} \in (\mathcal{A} \cup \{\phi\})^{|I|}$, we write $\mathbf{a} \equiv \mathbf{b}$ if for every $\mathbf{x} \in (\mathcal{A} \cup \{\phi\})^s$, we have $f(\mathbf{x}(I, \mathbf{a})) = f(\mathbf{x}(I, \mathbf{b}))$.

It can be verified that $\equiv$ is an equivalence relation[4] for every f and I.

Definition 3.3.6. For every f and I, let $R_{I,f}$ denote the total number of equivalence classes induced by $\equiv$ and let

$$\Phi_{I,f} : (\mathcal{A} \cup \{\phi\})^{|I|} \longrightarrow \{1, 2, \ldots, R_{I,f}\}$$

[4] Witsenhausen [27] represented this equivalence relation in terms of the independent sets of a characteristic graph and his representation has been used in various problems related to function computation [28, 29]. Although $\equiv$ is defined with respect to a particular index set I and a function f, we do not make this dependence explicit – the values of I and f will be clear from the context.

be any function such that $\Phi_{I,f}(\mathbf{a}) = \Phi_{I,f}(\mathbf{b})$ iff $\mathbf{a} \equiv \mathbf{b}$.

That is, $\Phi_{I,f}$ assigns a unique index to each equivalence class, and

$$R_{I,f} = \left|\left\{\Phi_{I,f}(\mathbf{a}) : \mathbf{a} \in (\mathcal{A} \cup \{\phi\})^{|I|}\right\}\right|.$$

The value of $R_{I,f}$ is independent of the choice of $\Phi_{I,f}$.

Example 3.3.7.

- Let f be the identity target function. Then for every $\mathbf{a}, \mathbf{b} \in (\mathcal{A} \cup \{\phi\})^{|I|}$ we have $\mathbf{a} \equiv \mathbf{b}$ if and only if $\mathbf{a} = \mathbf{b}$. The number of distinct equivalence classes is

$$R_{I,f} = (|\mathcal{A}| + 1)^{|I|}.$$

- Let $\mathcal{A} = \{1, \ldots, m\}$ for some $m \geq 1$ and let f be the arithmetic sum target function. For $\mathbf{a}, \mathbf{b} \in (\mathcal{A} \cup \{\phi\})^{|I|}$, we have $\mathbf{a} \equiv \mathbf{b}$ if and only if

$$\sum_{i=1}^{|I|} (\mathbf{a})_i = \sum_{i=1}^{|I|} (\mathbf{b})_i$$

and the number of such possible sums is

$$R_{I,f} = m\,|I| + 1.$$

Lemma 3.3.8. *For any target function f,*

$$\mathcal{E}_{\mathit{min}}(f) \geq \max_I \max \left\{ \frac{\sqrt{\log_q\left(R_{I,f}\right)}}{3}, \frac{\log_q(R_{I,f})}{3\,|I|} \right\}.$$

Proof. Consider any $I = \{i_1, i_2, \ldots, i_{|I|}\}$. For any $\mathbf{a}, \mathbf{b} \in (\mathcal{A} \cup \{\phi\})^{|I|}$ such that $\mathbf{a} \not\equiv \mathbf{b}$, any feasible code should satisfy

$$\sum_{j \in \{1,\ldots,|I|\}} \pi_{i_j}^{(\mathbf{a})_j} \neq \sum_{j \in \{1,\ldots,|I|\}} \pi_{i_j}^{(\mathbf{b})_j}. \tag{3.6}$$

Note that for any I and $\mathbf{a} \in (\mathcal{A} \cup \{\phi\})^{|I|}$, $\dim\left(\sum_{j\in\{1,\dots,|I|\}} \pi_{i_j}^{(\mathbf{a})_j}\right) \leq |I|$ since it is composed of the union of at most $|I|$ one-dimensional subspaces. Then from Definition 4.1.4 and (3.6), there exist at least $R_{I,f}$ distinct subspaces, each with dimension at most $|I|$. From Lemma 3.3.1, we have

$$4 \cdot \sum_{j=1}^{\min\{l,|I|\}} q^{j(l-j)} \quad \geq \quad R_{I,f}. \tag{3.7}$$

This implies that

$$\begin{aligned} & 4 \cdot \sum_{j=1}^{l} q^{j(l-j)} \geq R_{I,f} \\ \Longrightarrow & 4l \cdot q^{\left(\frac{l}{2}\right)^2} \geq R_{I,f} \\ \Longrightarrow & \log_q(4l) + \left(\frac{l}{2}\right)^2 \geq \log_q(R_{I,f}). \end{aligned}$$

Since $\log_q(4l) \leq 2l^2$, we have

$$\begin{aligned} & 3l^2 \geq \log_q(R_{I,f}) \\ \Longrightarrow & l \geq \frac{\sqrt{\log_q(R_{I,f})}}{3}. \end{aligned}$$

From (3.7), we also have

$$\begin{aligned} & 4 \cdot \sum_{j=1}^{|I|} q^{j(l-j)} \geq R_{I,f} \\ \Longrightarrow & 4\,|I| \cdot q^{\hat{d}(l-\hat{d})} \geq R_{I,f} \;\; \text{with } \hat{d} = \operatorname*{argmax}_{j\in\{1,|I|\}} q^{j(l-j)} \\ \Longrightarrow & \log_q(4\,|I|) + \hat{d}(l-\hat{d}) + \geq \log_q(R_{I,f}). \end{aligned}$$

Since $\log_q(4\,|I|) \leq 2\,|I|$ and $\hat{d} \leq |I|$, we have

$$\begin{aligned} 2\,|I|\,l + |I|\,l &\geq \log_q(R_{I,f}) \\ \Longrightarrow l &\geq \frac{\log_q(R_{I,f})}{3\,|I|}. \end{aligned}$$

■

Example 3.3.9.

- For the arithmetic sum target function f, we get

$$\mathcal{E}_{\text{min}}(f) \geq \frac{\sqrt{\log_q(s\,|\mathcal{A}| + 1)}}{3}.$$

Comment : Note that when $|\mathcal{A}| \gg s$, the bounds in the above examples are better than the ones presented earlier in the section.

3.4 Bounds for specific functions

Any target function can be computed by first reconstructing all the source messages at the sink (i.e., computing the identity function $f(x_1, x_2, \ldots, x_s) = (x_1, x_2, \ldots, x_s)$ with each $x_i \in \mathcal{A} \cup \{\phi\}$) and then deriving the function value. Hence, the following lemma provides an upper bound on the cost for computing any function f.

Lemma 3.4.1. *There exists a $(1, l)$ feasible code for computing the identity function such that*

$$l = s + \lceil \log_q |\mathcal{A}| \rceil.$$

Proof. It is easy to see that this can be achieved simply by using coding vectors of length s, where each source σ_i when active uses the basis vector $\mathbf{e_i}$ as its coding vector and appends this to the information packet that consists of $\lceil \log_q |\mathcal{A}| \rceil$ symbols. ■

In the previous section, we provided lower bounds on $\mathcal{E}_{\text{min}}(f)$ for arbitrary functions. Functions for which the lower bound is of the same order as $s + \lceil \log_q |\mathcal{A}| \rceil$ are

- Let $\mathbf{H}$ be the $l \times s$ parity check matrix of a q-ary code with minimum distance $d_{min} = T + 1$.
- Source σ_i uses $C_i = \{\mathbf{h_i}\}$, where $\mathbf{h_i}$ is a column of $\mathbf{H}$.
- If the dimension of the subspace that the sink receives is less than T, it outputs 0. Otherwise, it outputs 1.

Figure 3.2: A $(1, l)$ code for the T-threshold function

hard to compute in the sense that it is almost optimal (up to a constant factor) to first recover all the source messages and then compute the function. For example, when $s \geq \log_q |\mathcal{A}|$, this is true for the arithmetic sum function, the parity function, and the majority function. Next, we discuss some example target functions.

3.4.1 T-threshold Function

Let $\mathcal{A} = \{1\}$. The T-threshold function is defined as[5]

$$f(x_1, x_2, \ldots, x_s) = \begin{cases} 1 & \text{if } x_1 + x_2 + \ldots + x_s \geq T \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 3.4.2. *There exists a $(1, l)$ feasible code for computing the T-threshold function with $T < (1 - 1/q)s$, such that*

$$l \leq sH_q\left(\frac{T}{s}\right)$$

where H_q is the q-ary entropy function defined as

$$H_q(x) = x \log_q\left(\frac{q-1}{x}\right) + (1 - x)\log_q\left(\frac{1}{1-x}\right) \ \forall\, x \in (0, 1).$$

Proof. Consider the scheme in Figure 3.2. The scheme uses a $l \times s$ parity check matrix of a binary code with minimum distance $d_{min} = T + 1$. From the Gilbert-Varshamov

[5] The function computes whether the number of active sources is at least T or not.

bound [30, 31], there exists such a matrix with

$$l \leq sH_q \left(\frac{T}{s}\right).$$

■

Comment : For a constant T, $O\left(sH_q\left(\frac{T}{s}\right)\right) = O\left(T \log_q s\right)$. Thus, while computing the identity function requires the cost to grow linearly with the number of sources s, the T-threshold function requires only logarithmic growth. We have the following matching lower bound.

Lemma 3.4.3. *For the T-threshold function f with $T < s/2$,*

$$\mathcal{E}_{min}(f) \geq \frac{s}{2} H_q \left(\frac{T}{2s}\right).$$

Proof. Consider two possible input vectors $(x_1, x_2, \ldots, x_s)$ and $(y_1, y_2, \ldots, y_s)$ such that

$$x_i = 1 \; \forall \; i \in \{1, 2, \ldots, T\} \text{ and } x_i = \phi \text{ otherwise}$$
$$y_i = 1 \; \forall \; i \in \{2, 3, \ldots, T\} \text{ and } y_i = \phi \text{ otherwise} .$$

Note that

$$1 = f(x_1, x_2, \ldots, x_s) \neq f(y_1, y_2, \ldots, y_s) = 0$$

and hence it is necessary for any feasible code for computing f that

$$\pi_1^1 + \sum_{i=2}^{T} \pi_i^1 \neq \sum_{i=2}^{T} \pi_i^1 \Longrightarrow \pi_1^1 \not\subseteq \sum_{i=2}^{T} \pi_i^1.$$

The same argument can be extended to get the following necessary condition. For any subset $(i_1, i_2, \ldots, i_T)$ of $\{1, 2, \ldots, s\}$,

$$\pi_{i_j}^1 \not\subseteq \sum_{k \neq j} \pi_{i_k}^1 \text{ for every } j \in \{1, 2, \ldots, T\}.$$

Denote a basis vector for π_i^1 by $\mathbf{v_i}$. From the necessary condition on the subspaces

$\pi_1^1, \pi_2^1, \ldots, \pi_s^1$, any collection of T vectors from $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_s}$ are linearly independent. The $l \times s$ matrix with the vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_s}$ as columns corresponds to the parity check matrix of a q-ary linear code of length s and minimum distance at least $T+1$. Using the bounds in [30, 31], for $T < s/2$ we have

$$l \quad \geq \quad sH_q\left(\frac{T}{2s}\right) - \frac{1}{2}\log_q\left(4T\left(1 - \frac{T}{2s}\right)\right).$$

The result then follows since

$$\frac{1}{2}\log_q\left(4T\left(1 - \frac{T}{2s}\right)\right) \leq \frac{s}{2}H_q\left(\frac{T}{2s}\right). \tag{3.8}$$

For $s \leq 11$, (3.8) can be verified numerically. Let $s \geq 12$. Then (3.8) holds if we show that for every $1 \leq T < s/2$,

$$s \cdot \frac{T}{2s}\ln\left(\frac{2s}{T}\right) \geq \ln(4T) \text{ or equivalently,}$$

$$T\ln\left(\frac{2s}{T}\right) - 2\ln(4T) \geq 0. \tag{3.9}$$

For $T = 1$, (3.9) holds since $s \geq 8$. Differentiating the left-hand side of (3.9) with respect to T, we get

$$\ln(2s) - \ln(T) - 1 - \frac{2}{T}$$

which is greater than zero since $s \geq 12$ and $T \leq s/2$. Thus, (3.9) is true for every $1 \leq T < s/2$ and thus (3.8) holds. ■

3.4.2 Maximum Function

Lemma 3.4.4. *There exists a $(1, l)$ feasible code for computing the maximum function such that*

$$l \leq \min\left\{|\mathcal{A}|, s + \lceil\log_q|\mathcal{A}|\rceil)\right\}.$$

Proof. Consider the following two schemes for computing the maximum function.

- *A $(1, |\mathcal{A}|)$ scheme*: Let $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v}_{|\mathcal{A}|}$ be linearly independent vectors of length $|\mathcal{A}|$ each. For every source σ_i, let $\mathcal{C}_i = \left(\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v}_{|\mathcal{A}|}\right)$. This scheme has $l = |\mathcal{A}|$.

• *A* $(1, s + \lceil \log_q |\mathcal{A}| \rceil)$ *scheme*: We can compute the identity function with $l = s + \lceil \log_q |\mathcal{A}| \rceil$ and hence can compute the maximum function also. This scheme is useful if $\mathcal{A} \geq s$. ■

Comment : Thus when $|\mathcal{A}| \ll s$, the first scheme is much more efficient than reconstructing all the source messages.

Lemma 3.4.5. *For the maximum target function f,*

$$\mathcal{E}_{\mathit{min}}(f) \geq \min\{|\mathcal{A}|, s\}.$$

Proof. Let $\mathcal{A} = \left(a_1, a_2, \ldots, a_{|\mathcal{A}|}\right)$ be an ordered set (in decreasing order) and let $M = \min\{|\mathcal{A}|, s\}$. Consider an input vector $\mathbf{x}$ such that

$$(\mathbf{x})_i = a_i \; \forall \; i \in \{1, \ldots, M\} \text{ and } (\mathbf{x})_i = \phi \text{ otherwise}$$

and a collection of M indices $i_1, i_2, \ldots, i_M$ such that each $i_j = j$. Then from (3.5), it can be easily verified that the maximum target function f satisfies property $\mathbf{Q}(M)$. The result then follows from Lemma 3.3.4. ■

3.4.3 K-largest Values Function

Let $\mathcal{A} = (a_1, a_2, \ldots, a_{|\mathcal{A}|})$ be an ordered set (in increasing order). For any given input vector $(x_1, x_2, \ldots, x_s)$, let $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_s)$ denote the vector which is a permutation of the input vector and satisfies $\hat{x}_i \geq \hat{x}_{i+1}$ for each i. Then the K-largest values function is given by

$$f\left(x_1, x_2, \ldots, x_s\right) = \left(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_K\right).$$

Lemma 3.4.6. *There exists a $(1, l)$ feasible code for computing the K-largest values function with $K < N/2$, such that*

$$l \leq |\mathcal{A}| \cdot sH_q\left(\frac{K}{2s}\right).$$

- Let $\mathbf{H}$ be the $(l/\left|\mathcal{A}\right|) \times s$ parity check matrix of a binary code with minimum distance $K+1$.
- If source σ_i takes value a_j from the alphabet $\mathcal{A}$, then it transmits a vector which is all zero except the $(j-1) \times (l/\left|\mathcal{A}\right|) + 1$ to $j \times (l/\left|\mathcal{A}\right|)$ elements, which take values from the i-th column of $\mathbf{H}$.
- Each vector in the union subspace Π that the sink receives is parsed into $|\mathcal{A}|$ sub-vectors of length $l/\left|\mathcal{A}\right|$.
- Let $\Pi_j \subseteq \mathbb{F}_q^{l/|\mathcal{A}|}$ denote the subspace spanned by collecting the j-th sub-vector of each vector in Π.
- Thus by calculating $\dim(\Pi_{|\mathcal{A}|})$, $\dim(\Pi_{|\mathcal{A}|-1})\ \ldots,$ the sink can compute the K largest values.

Figure 3.3: A $(1,l)$ code for K-largest values function

Proof. Consider the scheme in Figure 3.3.

Again from [30, 31], there exists a parity check matrix such that

$$\frac{l}{|\mathcal{A}|} \leq sH_q\left(\frac{K}{2s}\right).$$

■

Comment : Again, for constant $|\mathcal{A}|$ and K, the cost only grows logarithmically with the number of sources s.

Lemma 3.4.7. *For the K-largest values target function f with $K < s/2$,*

$$\mathcal{E}_{\mathit{min}}(f) \geq \frac{s}{2}H_q\left(\frac{K}{2s}\right).$$

Proof. If the receiver can correctly compute the K-largest values, then it can also deduce if the number of active sources is greater than K or not. Thus, it can also compute the T-threshold function with the threshold $T = K$. The result then follows from Lemma 3.4.3. ■

3.5 A general scheme for computation

We now present a general method to compute functions under our network model. We will illustrate the method for boolean functions of the form $f : \mathcal{A}^s \to \{0, 1\}$. For a general function, the output can be considered as a string of bits and the above scheme can be used separately to compute each bit of the output.

Since f has boolean output, it can be written as

$$f(x_1, x_2, \ldots, x_s) = \sum_{i=1}^{s} \prod_{j=1}^{s} B_{ij}$$

where s is some integer such that $1 \leq s \leq |\mathcal{A}|^s$; $\{B_{ij}\}$ are boolean variables such that the value of B_{ij} depends only on x_j; and the sum and product represent boolean OR and AND. By taking the complement, we have

$$\overline{f(x_1, x_2, \ldots, x_s)} = \prod_{i=1}^{s} \sum_{j=1}^{s} \overline{B_{ij}}.$$

Given any input x_j, source j creates a vector v_j of length s such that i-th component is $\overline{B_{ij}}$. Each source j then sends the corresponding vector v_j into the network and the sink collects linear combinations of these vectors. If the i-th component of any of the vectors in the union subspace at the sink is 1, then a boolean variable A_i is assigned the value 1. This implies that

$$A_i = \sum_{j=1}^{s} \overline{B_{ij}}$$

and hence,

$$f(x_1, x_2, \ldots, x_s) = \overline{\prod_{i=1}^{s} A_i}.$$

Thus, we have a $(1, s)$ scheme to compute any function f with binary output.

Comment : Since the cost associated with the above code is s, the above scheme is efficient when the number of input vectors for which the function value is 1 (or 0) is much smaller than the total number of possible input vectors.

We now present an example to illustrate the above method.

Example 3.5.1. Let $\mathcal{B} = \{1, 2, \ldots, K\}$ and let the source alphabet $\mathcal{A}$ be the power set of $\mathcal{B}$, i.e, $\mathcal{A} = 2^{\mathcal{B}}$. Then the set cover function is defined as

$$f(x_1, x_2, \ldots, x_s) = \begin{cases} 1 & \text{if } \mathcal{B} \not\subseteq \bigcup_{i=1}^{s} x_i \\ 0 & \text{otherwise.} \end{cases}$$

In words, each source observes a subset of $\mathcal{B}$ and the sink needs to compute if the union of the source messages covers $\mathcal{B}$. Define the boolean variable $\mathbb{1}_A$ as follows.

$$\mathbb{1}_A = \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{otherwise.} \end{cases}$$

Then the function f can be rewritten as

$$f(x_1, x_2, \ldots, x_s) = \sum_{i=1}^{K} \prod_{j=1}^{s} \mathbb{1}_{\{i \notin x_j\}}.$$

Then using the scheme described in this section, the set cover function can be computed using a $(1, K)$ code with $d \cdot l = \log_2 |\mathcal{A}| = K$. This scheme is in-fact optimal in terms of the smallest possible cost for any feasible code.

3.6 Conclusions

In this chapter we investigated function computation in a network where intermediate nodes perform randomized network coding, through appropriate choice of the subspace codebooks at the source nodes. Unlike traditional function computation, that requires intermediate nodes to be aware of the function to be computed, our designs are transparent to the intermediate node operations. Future work includes finding tighter bounds for general functions as well as designing more efficient schemes. Another direction of research would be to relax our assumption that the sink is able to observe the joint span of the injected subspaces and allow it to only learn some subspace of the union.

Chapter 3, in part, has been submitted for publication of the material. The dissertation author was a primary investigator and author of this paper.

Chapter 4

Repeated Computation: Network Coding for Computing

The following *network computing* problem is considered. Source nodes in a directed acyclic network generate independent messages and a single receiver node computes a target function f of the messages. The objective is to maximize the average number of times f can be computed per network usage, i.e., the "computing capacity". The *network coding* problem for a single-receiver network is a special case of the network computing problem in which all of the source messages must be reproduced at the receiver. For network coding with a single receiver, routing is known to achieve the capacity by achieving the network *min-cut* upper bound. We extend the definition of min-cut to the network computing problem and show that the min-cut is still an upper bound on the maximum achievable rate and is tight for computing (using coding) any target function in multi-edge tree networks and for computing linear target functions in any network. We also study the bound's tightness for different classes of target functions.

4.1 Introduction

We consider networks where source nodes generate independent messages and a single receiver node computes a target function f of these messages. The objective is to characterize the maximum rate of computation, that is the maximum number of times f can be computed per network usage.

Giridhar and Kumar [3] have recently stated:

> “In its most general form, computing a function in a network involves communicating possibly correlated messages, to a specific destination, at a desired fidelity with respect to a joint distortion criterion dependent on the given function of interest. This combines the complexity of source coding of correlated sources, with rate distortion, different possible network collaborative strategies for computing and communication, and the inapplicability of the separation theorem demarcating source and channel coding.”

The overwhelming complexity of network computing suggests that simplifications be examined in order to obtain some understanding of the field.

We present a natural model of network computing that is closely related to the network coding model of Ahlswede, Cai, Li, and Yeung [32, 33]. Network coding is a widely studied communication mechanism in the context of network information theory. In network coding, some nodes in the network are labeled as sources and some as receivers. Each receiver needs to reproduce a subset of the messages generated by the source nodes, and all nodes can act as relays and encode the information they receive on in-edges, together with the information they generate if they are sources, into codewords which are sent on their out-edges. In existing computer networks, the encoding operations are purely routing: at each node, the codeword sent over an out-edge consists of a symbol either received by the node, or generated by it if it is a source. It is known that allowing more complex encoding than routing can in general be advantageous in terms of communication rate [32, 34, 35]. Network coding with a single receiver is equivalent to a special case of our function computing problem, namely when the function to be computed is the identity, that is when the receiver wants to reproduce all the messages generated by the sources. In this chapter, we study network computation for target functions different than the identity.

Some other approaches to network computation have also appeared in the literature. In [28,29,36–39] network computing was considered as an extension of distributed source coding, allowing the sources to have a joint distribution and requiring that a function be computed with small error probability. For example, [36] considered a network where two correlated uniform binary sources are both connected to the receiver and determine the maximum rate of computing the parity of the messages generated by the two sources. A rate-distortion approach to the problem has been studied in [40–42]. However, the complexity of network computing has restricted prior work to the analysis of elementary networks. Networks with noisy links were studied in [6–8,10,12,43–46]. For example, [6] considered broadcast networks where any transmission by a node is received by each of its neighbors via an independent binary symmetric channel. Randomized gossip algorithms [47] have been proposed as practical schemes for information dissemination in large unreliable networks and were studied in the context of distributed computation in [47–52].

In the present chapter, our approach is somewhat (tangentially) related to the field of communication complexity [1,53] which studies the minimum number of messages that two nodes need to exchange in order to compute a function of their inputs with zero error. Other studies of computing in networks have been considered in [3,4], but these were restricted to the wireless communication protocol model of Gupta and Kumar [2].

In contrast, our approach is more closely associated with wired networks with independent noiseless links. Our work is closest in spirit to the recent work of [54–57] on computing the sum (over a finite field) of source messages in networks. We note that in independent work, Kowshik and Kumar [58] obtain the asymptotic maximum rate of computation in tree networks and present bounds for computation in networks where all nodes are sources.

Our main contributions are summarized in Section 4.1.3, after formally introducing the network model.

4.1.1 Network model and definitions

In this chapter, a *network* $\mathcal{N}$ consists of a finite, directed acyclic multigraph $G = (\mathcal{V}, \mathcal{E})$, a set of *source nodes* $S = \{\sigma_1, \ldots, \sigma_s\} \subseteq \mathcal{V}$, and a *receiver* $\rho \in \mathcal{V}$. Such a network is denoted by $\mathcal{N} = (G, S, \rho)$. We will assume that $\rho \notin S$ and that the graph[1] G contains a directed path from every node in $\mathcal{V}$ to the receiver ρ. For each node $u \in \mathcal{V}$, let $\mathcal{E}_i(u)$ and $\mathcal{E}_o(u)$ denote the set of in-edges and out-edges of u respectively. We will also assume (without loss of generality) that if a network node has no in-edges, then it is a source node.

An *alphabet* $\mathcal{A}$ is a finite set of size at least two. For any positive integer m, any vector $x \in \mathcal{A}^m$, and any $i \in \{1, 2, \ldots, m\}$, let x_i denote the i-th component of x. For any index set $I = \{i_1, i_2, \ldots, i_q\} \subseteq \{1, 2, \ldots, m\}$ with $i_1 < i_2 < \ldots < i_q$, let x_I denote the vector $(x_{i_1}, x_{i_2}, \ldots, x_{i_q}) \in \mathcal{A}^{|I|}$.

The *network computing* problem consists of a network $\mathcal{N}$ and a *target function* f of the form

$$f : \mathcal{A}^s \longrightarrow \mathcal{B}$$

(see Table 4.1 for some examples). We will also assume that any target function depends on all network sources (i.e. they cannot be constant functions of any one of their arguments). Let k and n be positive integers. Given a network $\mathcal{N}$ with source set S and alphabet $\mathcal{A}$, a *message generator* is any mapping

$$\alpha \ : \ S \longrightarrow \mathcal{A}^k.$$

For each i, $\alpha(\sigma_i)$ is called a *message vector* and its components $\alpha(\sigma_i)_1, \ldots, \alpha(\sigma_i)_k$ are called *messages*.[2]

Definition 4.1.1. A (k, n) *network code* for computing a target function f in a network $\mathcal{N}$ consists of the following:

[1] Throughout the chapter, we will use “graph” to mean a directed acyclic multigraph, and “network” to mean a single-receiver network. We may sometimes write $\mathcal{E}(G)$ to denote the edges of graph G.

[2] For simplicity, we assume that each source has exactly one message vector associated with it, but all of the results in this chapter can readily be extended to the more general case.

(i) For any node $v \in \mathcal{V} - \rho$ and any out-edge $e \in \mathcal{E}_o(v)$, an *encoding function*:

$$h^{(e)} : \begin{cases} \left(\prod_{\hat{e} \in \mathcal{E}_i(v)} \mathcal{A}^n \right) \times \mathcal{A}^k \longrightarrow \mathcal{A}^n & \text{if } v \text{ is a source node} \\ \prod_{\hat{e} \in \mathcal{E}_i(v)} \mathcal{A}^n \longrightarrow \mathcal{A}^n & \text{otherwise.} \end{cases}$$

(ii) A *decoding function*:

$$\psi : \prod_{j=1}^{|\mathcal{E}_i(\rho)|} \mathcal{A}^n \longrightarrow \mathcal{B}^k.$$

Given a (k, n) network code, every edge $e \in \mathcal{E}$ *carries a vector* z_e of at most n alphabet symbols[3], which is obtained by evaluating the encoding function $h^{(e)}$ on the set of vectors carried by the in-edges to the node and the node's message vector if it is a source. The objective of the receiver is to compute the target function f of the source messages, for any arbitrary message generator α. More precisely, the receiver constructs a vector of k alphabet symbols such that for each $i \in \{1, 2, \ldots, k\}$, the i-th component of the receiver's computed vector equals the value of the desired target function f applied to the i-th components of the source message vectors, for any choice of message generator α. Let $e_1, e_2, \ldots, e_{|\mathcal{E}_i(\rho)|}$ denote the in-edges of the receiver.

Definition 4.1.2. A (k, n) network code is called *a solution for computing f in $\mathcal{N}$* (or simply *a (k, n) solution*) if the decoding function ψ is such that for each $j \in \{1, 2, \ldots, k\}$ and for every message generator α, we have

$$\psi\left(z_{e_1}, \cdots, z_{e_{|\mathcal{E}_i(\rho)|}}\right)_j = f\left(\alpha(\sigma_1)_j, \cdots, \alpha(\sigma_s)_j\right). \tag{4.1}$$

If there exists a (k, n) solution, we say the rational number k/n is an *achievable computing rate*.

In the network coding literature, one definition of the *coding capacity* of a network is the supremum of all achievable coding rates [59, 60]. We adopt an analogous definition for computing capacity.

[3] By default, we will assume that edges carry exactly n symbols.

Table 4.1: Examples of target functions.

Target function f	Alphabet $\mathcal{A}$	$f\,(x_1, \ldots, x_s)$	Comments
identity	arbitrary	$(x_1, \ldots, x_s)$	
arithmetic sum	$\{0, \ldots, q-1\}$	$x_1 + \cdots + x_s$	'+' is ordinary integer addition
mod r sum	$\{0, \ldots, q-1\}$	$x_1 \oplus \ldots \oplus x_s$	$\oplus$ is $\mathrm{mod}\ r$ addition
histogram	$\{0, \ldots, q-1\}$	$(c_0, \ldots, c_{q-1})$	$c_i = \lvert\{j : x_j = i\}\rvert \, \forall \, i \in \mathcal{A}$
linear	finite field	$a_1x_1 + \ldots + a_sx_s$	arithmetic performed in the field
maximum	ordered set	$\max\{x_1, \ldots, x_s\}$	

Definition 4.1.3. The *computing capacity* of a network $\mathcal{N}$ with respect to target function f is

$$\mathcal{C}_{\text{cod}}(\mathcal{N}, f) = \sup\left\{ \frac{k}{n} \ : \ \exists\,(k,n) \text{ network code for computing } f \text{ in } \mathcal{N} \right\}.$$

Thus, the computing capacity is the supremum of all achievable computing rates for a given network $\mathcal{N}$ and a target function f. Some example target functions are defined in Table 4.1.

Definition 4.1.4. For any target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$, any index set $I \subseteq \{1, 2, \ldots, s\}$, and any $a, b \in \mathcal{A}^{|I|}$, we write $a \equiv b$ if for every $x, y \in \mathcal{A}^s$, we have $f(x) = f(y)$ whenever $x_I = a$, $y_I = b$, and $x_j = y_j$ for all $j \notin I$.

It can be verified that $\equiv$ is an equivalence relation[4] for every f and I.

Definition 4.1.5. For every f and I, let $R_{I,f}$ denote the total number of equivalence classes induced by $\equiv$ and let

$$\Phi_{I,f} : \mathcal{A}^{|I|} \longrightarrow \{1, 2, \ldots, R_{I,f}\}$$

[4] Witsenhausen [27] represented this equivalence relation in terms of the independent sets of a characteristic graph and his representation has been used in various problems related to function computation [28, 29, 37]. Although $\equiv$ is defined with respect to a particular index set I and a function f, we do not make this dependence explicit – the values of I and f will be clear from the context.

be any function such that $\Phi_{I,f}(a) = \Phi_{I,f}(b)$ iff $a \equiv b$.

That is, $\Phi_{I,f}$ assigns a unique index to each equivalence class, and

$$R_{I,f} = \left| \left\{ \Phi_{I,f}(a) : a \in \mathcal{A}^{|I|} \right\} \right| .$$

The value of $R_{I,f}$ is independent of the choice of $\Phi_{I,f}$. We call $R_{I,f}$ the *footprint size* of f with respect to I.

Remark 4.1.6. Let $I^c = \{1, 2, \ldots, s\} - I$. The footprint size $R_{I,f}$ has the following interpretation. Suppose a network has two nodes, X and Y, and both are sources. A single directed edge connects X to Y. Let X generate $x \in \mathcal{A}^{|I|}$ and Y generate $y \in \mathcal{A}^{|I^c|}$. X communicates a function $g(x)$ of its input, to Y so that Y can compute $f(a)$ where $a \in \mathcal{A}^s$, $a_I = x$, and $a_{I^c} = y$. Then for any $x, \hat{x} \in \mathcal{A}^{|I|}$ such that $x \not\equiv \hat{x}$, we need $g(x) \neq g(\hat{x})$. Thus $\left| g\left(\mathcal{A}^{|I|}\right) \right| \geq R_{I,f}$, which implies a lower bound on a certain amount of "information" that X needs to send to Y to ensure that it can compute the function f. Note that $g = \Phi_{I,f}$ achieves the lower bound. We will use this intuition to establish a cut-based upper bound on the computing capacity $\mathcal{C}_{\text{cod}}(\mathcal{N}, f)$ of any network $\mathcal{N}$ with respect to any target function f, and to devise a capacity-achieving scheme for computing any target function in multi-edge tree networks.

Definition 4.1.7. A set of edges $C \subseteq \mathcal{E}$ in network $\mathcal{N}$ is said to *separate* sources $\sigma_{m_1}, \ldots, \sigma_{m_d}$ from the receiver ρ, if for each $i \in \{1, 2, \ldots, d\}$, every directed path from σ_{m_i} to ρ contains at least one edge in C. The set C is said to be a *cut* in $\mathcal{N}$ if it separates at least one source from the receiver. For any network $\mathcal{N}$, define $\Lambda(\mathcal{N})$ to be the collection of all cuts in $\mathcal{N}$. For any cut $C \in \Lambda(\mathcal{N})$ and any target function f, define

$$\begin{aligned} I_C &= \{i : C \text{ separates } \sigma_i \text{ from the receiver}\} \\ R_{C,f} &= R_{I_C,f}. \end{aligned} \tag{4.2}$$

Since target functions depend on all sources, we have $R_{C,f} \geq 2$ for any cut C and any target function f. The footprint sizes $R_{C,f}$ for some example target functions are computed below.

A *multi-edge tree* is a graph such that for every node $v \in \mathcal{V}$, there exists a node u such that all the out-edges of v are in-edges to u, i.e., $\mathcal{E}_o(v) \subseteq \mathcal{E}_i(u)$ (e.g. see Figure 4.1).

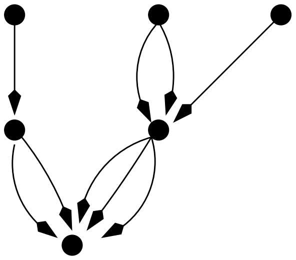


Figure 4.1: An example of a multi-edge tree.

4.1.2 Classes of target functions

We study the following four classes of target functions: (1) divisible, (2) symmetric, (3) λ-exponential, (4) λ-bounded.

Definition 4.1.8. A target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$ is *divisible* if for every index set $I \subseteq \{1, \ldots, s\}$, there exists a finite set $\mathcal{B}_I$ and a function $f^I : \mathcal{A}^{|I|} \longrightarrow \mathcal{B}_I$ such that the following hold:

(1) $f^{\{1,\ldots,s\}} = f$

(2) $\left|f^I\left(\mathcal{A}^{|I|}\right)\right| \leq |f\left(\mathcal{A}^s\right)|$

(3) For every partition $\{I_1, \ldots, I_\gamma\}$ of I, there exists a function
$g : \mathcal{B}_{I_1} \times \cdots \times \mathcal{B}_{I_\gamma} \longrightarrow \mathcal{B}_I$ such that for every $x \in \mathcal{A}^{|I|}$, we have
$f^I(x) = g\left(f^{I_1}(x_{I_1}), \ldots, f^{I_\gamma}\left(x_{I_\gamma}\right)\right)$.

Examples of divisible target functions include the identity, maximum, $\bmod r$ sum, and arithmetic sum.

Divisible functions have been studied previously[5] by Giridhar and Kumar [3] and Subramanian, Gupta, and Shakkottai [4]. Divisible target functions can be computed in networks in a divide-and-conquer fashion as follows. For any arbitrary partition

[5] The definitions in [3, 4] are similar to ours but slightly more restrictive.

$\{I_1, \ldots, I_\gamma\}$ of the source indices $\{1, \ldots, s\}$, the receiver ρ can evaluate the target function f by combining evaluations of $f^{I_1}, \ldots, f^{I_\gamma}$. Furthermore, for every $i = 1, \ldots, \gamma$, the target function f^{I_i} can be evaluated similarly by partitioning I_i and this process can be repeated until the function value is obtained.

Definition 4.1.9. A target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$ is *symmetric* if for any permutation π of $\{1, 2, \ldots, s\}$ and any vector $x \in \mathcal{A}^s$,

$$f(x_1, x_2, \ldots, x_s) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(s)}).$$

That is, the value of a symmetric target function is invariant with respect to the order of its arguments and hence, it suffices to evaluate the histogram target function for computing any symmetric target function. Examples of symmetric functions include the arithmetic sum, maximum, and $\mathrm{mod}\ r$ sum. Symmetric functions have been studied in the context of computing in networks by Giridhar and Kumar [3], Subramanian, Gupta, and Shakkottai [4], Ying, Srikant, and Dullerud [10], and [43].

Definition 4.1.10. Let $\lambda \in (0, 1]$. A target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$ is said to be λ-*exponential* if its footprint size satisfies

$$R_{I,f} \geq |\mathcal{A}|^{\lambda|I|} \text{ for every } I \subseteq \{1, 2, \ldots, s\}.$$

Let $\lambda \in (0, \infty)$. A target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$ is said to be λ-*bounded* if its footprint size satisfies

$$R_{I,f} \leq |\mathcal{A}|^{\lambda} \text{ for every } I \subseteq \{1, 2, \ldots, s\}.$$

Example 4.1.11. The following facts are easy to verify:

- The identity function is 1-exponential.
- Let $\mathcal{A}$ be an ordered set. The maximum (or minimum) function is 1-bounded.
- Let $\mathcal{A} = \{0, 1, \ldots, q-1\}$ where $q \geq 2$. The $\mathrm{mod}\ r$ sum target function with $q \geq r \geq 2$ is $\log_q r$-bounded.

Remark 4.1.12. Giridhar and Kumar [3] defined two classes of functions: *type-threshold* and *type-sensitive* functions. Both are sub-classes of symmetric functions. In addition, type-threshold functions are also divisible and c-bounded, for some constant c that is independent of the network size. However, [3] uses a model of interference for simultaneous transmissions and their results do not directly compare with ours.

Following the notation in Leighton and Rao [61], the *min-cut* of any network $\mathcal{N}$ with unit-capacity edges is

$$\text{min-cut}(\mathcal{N}) = \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{|I_C|}. \tag{4.3}$$

A more general version of the network min-cut plays a fundamental role in the field of multi-commodity flow [61, 62]. The min-cut provides an upper bound on the maximum flow for any multi-commodity flow problem. The min-cut is also referred to as "sparsity" by some authors, such as Harvey, Kleinberg, and Lehman [34] and Vazirani [62]. We next generalize the definition in (4.3) to the network computing problem.

Definition 4.1.13. If $\mathcal{N}$ is a network and f is a target function, then define

$$\text{min-cut}(\mathcal{N}, f) = \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{|\mathcal{A}|} R_{C,f}}. \tag{4.4}$$

Example 4.1.14.

- If f is the identity target function, then

$$\text{min-cut}(\mathcal{N}, f) = \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{|I_C|}.$$

  Thus for the identity function, the definition of min-cut in (4.3) and (4.4) coincide.

- Let $\mathcal{A} = \{0, 1, \ldots, q-1\}$. If f is the arithmetic sum target function, then

$$\text{min-cut}(\mathcal{N}, f) = \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_q\left((q-1)\,|I_C| + 1\right)}. \tag{4.5}$$

- Let $\mathcal{A}$ be an ordered set. If f is the maximum target function, then

$$\text{min-cut}(\mathcal{N}, f) = \min_{C \in \Lambda(\mathcal{N})} |C| \, .$$

4.1.3 Contributions

The main results of this chapter are as follows. In Section 4.2, we show (Theorem 4.2.1) that for any network $\mathcal{N}$ and any target function f, the quantity min-cut$(\mathcal{N}, f)$ is an upper bound on the computing capacity $\mathcal{C}_{\text{cod}}(\mathcal{N}, f)$. In Section 4.3, we note that the computing capacity for any network with respect to the identity target function is equal to the min-cut upper bound (Theorem 4.3.1). We show that the min-cut bound on computing capacity can also be achieved for all networks with linear target functions over finite fields (Theorem 5.5.6) and for all multi-edge tree networks with any target function (Theorem 4.3.3). For any network and any target function, a lower bound on the computing capacity is given in terms of the Steiner tree packing number (Theorem 4.3.5). Another lower bound is given for networks with symmetric target functions (Theorem 4.3.7). In Section 4.4, the tightness of the above-mentioned bounds is analyzed for divisible (Theorem 4.4.2), symmetric (Theorem 4.4.3), λ-exponential (Theorem 4.4.4), and λ-bounded (Theorem 4.4.5) target functions. For λ-exponential target functions, the computing capacity is at least λ times the min-cut. If every non-receiver node in a network is a source, then for λ-bounded target functions the computing capacity is at least a constant times the min-cut divided by λ. It is also shown, with an example target function, that there are networks for which the computing capacity is less than an arbitrarily small fraction of the min-cut bound (Theorem 4.4.7). In Section 4.5, we discuss an example network and target function in detail to illustrate the above bounds. In Section 4.6, conclusions are given and various lemmas are proven in the Appendix.

4.2 Min-cut upper bound on computing capacity

The following shows that the maximum rate of computing a target function f in a network $\mathcal{N}$ is at most min-cut$(\mathcal{N}, f)$.

Theorem 4.2.1. *If $\mathcal{N}$ is a network with target function f, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) \leq \text{min-cut}(\mathcal{N}, f).$$

Proof. Let the network alphabet be $\mathcal{A}$ and consider any (k, n) solution for computing f in $\mathcal{N}$. Let C be a cut and for each $i \in \{1, 2, \ldots, k\}$, let $a^{(i)}, b^{(i)} \in \mathcal{A}^{|I_C|}$. Suppose $j \in \{1, 2, \ldots, k\}$ is such that $a^{(j)} \not\equiv b^{(j)}$, where $\equiv$ is the equivalence relation from Definition 4.1.4. Then there exist $x, y \in \mathcal{A}^s$ satsifying: $f(x) \neq f(y)$, $x_{I_C} = a^{(j)}$, $y_{I_C} = b^{(j)}$, and $x_i = y_i$ for every $i \notin I_C$.

The receiver ρ can compute the target function f only if, for every such pair $\left\{a^{(1)}, \ldots, a^{(k)}\right\}$ and $\left\{b^{(1)}, \ldots, b^{(k)}\right\}$ corresponding to the message vectors generated by the sources in I_C, the edges in cut C carry distinct vectors. Since the total number of equivalence classes for the relation $\equiv$ equals the footprint size $R_{C,f}$, the edges in cut C should carry at least $\left(R_{C,f}\right)^k$ distinct vectors. Thus, we have

$$\mathcal{A}^{n|C|} \geq \left(R_{C,f}\right)^k$$

and hence for any cut C,

$$\frac{k}{n} \leq \frac{|C|}{\log_{|\mathcal{A}|} R_{C,f}}.$$

Since the cut C is arbitrary, the result follows from Definition 5.2.6 and (4.4). ■

The min-cut upper bound has the following intuition. Given any cut $C \in \Lambda(\mathcal{N})$, at least $\log_{|\mathcal{A}|} R_{C,f}$ units of information need to be sent across the cut to successfully compute a target function f. In subsequent sections, we study the tightness of this bound for different classes of functions and networks.

4.3 Lower bounds on the computing capacity

The following result shows that the computing capacity of any network $\mathcal{N}$ with respect to the identity target function equals the coding capacity for ordinary network coding.

Theorem 4.3.1. *If $\mathcal{N}$ is a network with the identity target function f, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) = \text{min-cut}(\mathcal{N}, f) = \text{min-cut}(\mathcal{N}).$$

Proof. Rasala Lehman and Lehman [63, p.6, Theorem 4.2] showed that for any single-receiver network, the conventional coding capacity (when the receiver demands the messages generated by all the sources) always equals the min-cut$(\mathcal{N})$. Since the target function f is the identity, the computing capacity is the coding capacity and min-cut$(\mathcal{N}, f)$ = min-cut$(\mathcal{N})$, so the result follows. ■

Theorem 4.3.2. *If $\mathcal{N}$ is a network with a finite field alphabet and with a linear target function f, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) = \text{min-cut}(\mathcal{N}, f).$$

Proof. The proof of this result is relegated to Section 5.5. It also follows from [56, Theorem 2]. ■

Theorems 4.3.1 and 5.5.6 demonstrate the achievability of the min-cut bound for arbitrary networks with particular target functions. In contrast, the following result demonstrates the achievability of the min-cut bound for arbitrary target functions and a particular class of networks. The following theorem concerns multi-edge tree networks, which were defined in Section 4.1.1.

Theorem 4.3.3. *If $\mathcal{N}$ is a multi-edge tree network with target function f, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) = \text{min-cut}(\mathcal{N}, f).$$

Proof. Let $\mathcal{A}$ be the network alphabet. From Theorem 4.2.1, it suffices to show that $\mathcal{C}_{\text{cod}}(\mathcal{N}, f) \geq$ min-cut$(\mathcal{N}, f)$. Since $\mathcal{E}_o(v)$ is a cut for node $v \in \mathcal{V} - \rho$, and using (4.2), we have

$$\text{min-cut}(\mathcal{N}, f) \leq \min_{v \,\in\, \mathcal{V} - \rho} \frac{|\mathcal{E}_o(v)|}{\log_{|\mathcal{A}|} R_{\mathcal{E}_o(v), f}}. \tag{4.6}$$

Consider any positive integers k, n such that

$$\frac{k}{n} \leq \min_{v\, \in\, \mathcal{V}-\rho} \frac{|\mathcal{E}_o(v)|}{\log_{|\mathcal{A}|} R_{I_{\mathcal{E}_o(v)},f}}\ . \tag{4.7}$$

Then we have

$$|\mathcal{A}|^{|\mathcal{E}_o(v)|n} \geq R^k_{I_{\mathcal{E}_o(v)},f} \text{ for every node } v \in \mathcal{V} - \rho. \tag{4.8}$$

We outline a (k, n) solution for computing f in the multi-edge tree network $\mathcal{N}$. Each source $\sigma_i \in S$ generates a message vector $\alpha(\sigma_i) \in \mathcal{A}^k$. Denote the vector of i-th components of the source messages by

$$x^{(i)} = \left(\alpha(\sigma_1)_i\,, \cdots, \alpha(\sigma_s)_i\right).$$

Every node $v \in \mathcal{V} - \{\rho\}$ sends out a unique index (as guaranteed by (4.8)) over $A^{|\mathcal{E}_o(v)|n}$ corresponding to the set of equivalence classes

$$\Phi_{I_{\mathcal{E}_o(v)},f}(x^{(l)}_{I_{\mathcal{E}_o(v)}}) \text{ for } l \in \{1, \cdots, k\}. \tag{4.9}$$

If v has no in-edges, then by assumption, it is a source node, say σ_j. The set of equivalence classes in (4.9) is a function of its own messages $\alpha(\sigma_j)_l$ for $l \in \{1, \ldots, k\}$. On the other hand if v has in-edges, then let $u_1, u_2, \cdots, u_j$ be the nodes with out-edges to v. For each $i \in \{1, 2, \cdots, j\}$, using the uniqueness of the index received from u_i, node v recovers the equivalence classes

$$\Phi_{I_{\mathcal{E}_o(u_i)},f}(x^{(l)}_{I_{\mathcal{E}_o(u_i)}}) \text{ for } l \in \{1, \cdots, k\}. \tag{4.10}$$

Furthermore, the equivalence classes in (4.9) can be identified by v from the equivalance classes in (4.10) (and $\alpha(v)$ if v is a source node) using the fact that for a multi-edge tree network $\mathcal{N}$, we have a disjoint union

$$I_{\mathcal{E}_o(v)} = \bigcup_{i=1}^{j} I_{\mathcal{E}_o(u_i)}.$$

If each node v follows the above steps, then the receiver ρ can identify the equivalence classes $\Phi_{I_{\mathcal{E}_i(\rho)},f}\left(x^{(i)}\right)$ for $i \in \{1,\ldots,k\}$. The receiver can evaluate $f(x^{(l)})$ for each l from these equivalence classes. The above solution achieves a computing rate of k/n. From (4.7), it follows that

$$\mathcal{C}_{\text{cod}}(\mathcal{N},f) \geq \min_{v\,\in\,\mathcal{V}-\rho} \frac{|\mathcal{E}_o(v)|}{\log_{|\mathcal{A}|} R_{I_{\mathcal{E}_o(v)},f}}. \tag{4.11}$$

■

We next establish a general lower bound on the computing capacity for arbitrary target functions (Theorem 4.3.5) and then another lower bound specifically for symmetric target functions (Theorem 4.3.7).

For any network $\mathcal{N} = (G, S, \rho)$ with $G = (\mathcal{V}, \mathcal{E})$, define a *Steiner tree*[6] of $\mathcal{N}$ to be a minimal (with respect to nodes and edges) subgraph of G containing S and ρ such that every source in S has a directed path to the receiver ρ. Note that every non-receiver node in a Steiner tree has exactly one out-edge. Let $\mathcal{T}(\mathcal{N})$ denote the collection of all Steiner trees in $\mathcal{N}$. For each edge $e \in \mathcal{E}(G)$, let $J_e = \{i\,:\,t_i \in \mathcal{T}(\mathcal{N})$ and $e \in \mathcal{E}(t_i)\}$. The *fractional Steiner tree packing number* $\Pi(\mathcal{N})$ is defined as the linear program

$$\Pi(\mathcal{N}) = \max \sum_{t_i \in \mathcal{T}(\mathcal{N})} u_i \text{ subject to } \begin{cases} u_i \geq 0 \quad \forall\, t_i \in \mathcal{T}(\mathcal{N}), \\ \displaystyle\sum_{i \in J_e} u_i \leq 1 \quad \forall\, e \in \mathcal{E}(G). \end{cases} \tag{4.12}$$

Note that $\Pi(\mathcal{N}) \geq 1$ for any network $\mathcal{N}$, and the maximum value of the sum in (4.12) is attained at one or more vertices of the closed polytope corresponding to the linear constraints. Since all coefficients in the constraints are rational, the maximum value in (4.12) can be attained with rational u_i's. The following theorem provides a lower bound[7] on the computing capacity for any network $\mathcal{N}$ with respect to a target function f and uses the quantity $\Pi(\mathcal{N})$. In the context of computing functions, u_i in the above

[6] Steiner trees are well known in the literature for undirected graphs. For directed graphs a “Steiner tree problem” has been studied and our definition is consistent with such work (e.g., see [64]).

[7] In order to compute the lower bound, the fractional Steiner tree packing number $\Pi(\mathcal{N})$ can be evaluated using linear programming. Also note that if we construct the *reverse multicast network* by letting each source in the original network $\mathcal{N}$ become a receiver, letting the receiver in the $\mathcal{N}$ become the only source, and reversing the direction of each edge, then it can be verified that the routing capacity for the reverse multicast network is equal to $\Pi(\mathcal{N})$.

linear program indicates the fraction of the time the edges in tree t_i are used to compute the desired function. The fact that every edge in the network has unit capacity implies $\sum_{i \in J_e} u_i \leq 1$.

Lemma 4.3.4. *For any Steiner tree G' of a network $\mathcal{N}$, let $\mathcal{N}' = (G', S, \rho)$. Let C' be a cut in $\mathcal{N}'$. Then there exists a cut C in $\mathcal{N}$ such that $I_C = I_{C'}$.*

(Note that $I_{C'}$ is the set indices of sources separated in $\mathcal{N}'$ by C'. The set $I_{C'}$ may differ from the indices of sources separated in $\mathcal{N}$ by C'.)

Proof. Define the cut

$$C = \bigcup_{i' \in I_{C'}} \mathcal{E}_o(\sigma_{i'}). \tag{4.13}$$

C is the collection of out-edges in $\mathcal{N}$ of a set of sources disconnected by the cut C' in $\mathcal{N}'$. If $i \in I_{C'}$, then, by (4.13), C disconnects σ_i from ρ in $\mathcal{N}$, and thus $I_{C'} \subseteq I_C$.

Let σ_i be a source. such that $i \in I_C$ and Let P be a path from σ_i to ρ in $\mathcal{N}$. From (4.13), it follows that there exists $i' \in I_{C'}$ such that P contains at least one edge in $\mathcal{E}_o(\sigma_{i'})$. If P also lies in $\mathcal{N}'$ and does not contain any edge in C', then $\sigma_{i'}$ has a path to ρ in $\mathcal{N}'$ that does not contain any edge in C', thus contradicting the fact that $\sigma_{i'} \in I_{C'}$. Therefore, either P does not lie in $\mathcal{N}'$ or P contains an edge in C'. Thus $\sigma_i \in I_{C'}$, i.e., $I_C \subseteq I_{C'}$. ■

Theorem 4.3.5. *If $\mathcal{N}$ is a network with alphabet $\mathcal{A}$ and target function f, then*

$$\mathcal{C}_{\mathit{cod}}(\mathcal{N}, f) \geq \Pi(\mathcal{N}) \cdot \min_{C \in \Lambda(\mathcal{N})} \frac{1}{\log_{|\mathcal{A}|} R_{C,f}}.$$

Proof. Suppose $\mathcal{N} = (G, S, \rho)$. Consider a Steiner tree $G' = (\mathcal{V}', \mathcal{E}')$ of $\mathcal{N}$, and let $\mathcal{N}' = (G', S, \rho)$. From Lemma 4.3.4 (taking C' to be $\mathcal{E}_o(v)$ in $\mathcal{N}'$), we have

$$\forall\, v \in \mathcal{V}' - \rho,\ \exists\, C \in \Lambda(\mathcal{N}) \text{ such that } I'_{\mathcal{E}_o(v)} = I_C. \tag{4.14}$$

Now we lower bound the computing capacity for the network $\mathcal{N}'$ with respect to target function f.

$$\begin{aligned}
&\mathcal{C}_{\text{cod}}(\mathcal{N}', f) \\
&= \text{min-cut}(\mathcal{N}', f) && [\text{from Theorem 4.3.3}] && (4.15) \\
&= \min_{v \,\in\, \mathcal{V}' - \rho} \frac{1}{\log_{|\mathcal{A}|} R_{I'_{\mathcal{E}_o(v)},f}} && [\text{from Theorem 4.2.1, (4.6), (4.11)}] \\
&\geq \min_{C \in \Lambda(\mathcal{N})} \frac{1}{\log_{|\mathcal{A}|} R_{I_C,f}} && [\text{from (4.14)}]. && (4.16)
\end{aligned}$$

The lower bound in (4.16) is the same for every Steiner tree of $\mathcal{N}$. We will use this uniform bound to lower bound the computing capacity for $\mathcal{N}$ with respect to f. Denote the Steiner trees of $\mathcal{N}$ by $t_1, \ldots, t_T$. Let $\epsilon > 0$ and let r denote the quantity on the right hand side of (4.16). On every Steiner tree t_i, a computing rate of at least $r - \epsilon$ is achievable by (4.16). Using standard arguments for time-sharing between the different Steiner trees of the network $\mathcal{N}$, it follows that a computing rate of at least $(r - \epsilon) \cdot \Pi(\mathcal{N})$ is achievable in $\mathcal{N}$, and by letting $\epsilon \to 0$, the result follows. ■

The lower bound in Theorem 4.3.5 can be readily computed and is sometimes tight. The procedure used in the proof of Theorem 4.3.5 may potentially be improved by maximizing the sum

$$\sum_{t_i \in \mathcal{T}(\mathcal{N})} u_i \, r_i \quad \text{subject to} \quad \begin{cases} u_i \geq 0 \quad \forall \, t_i \in \mathcal{T}(\mathcal{N}) \,, \\ \displaystyle\sum_{i \in J_e} u_i \leq 1 \quad \forall \, e \in \mathcal{E}(G) \end{cases} \qquad (4.17)$$

where r_i is any achievable rate[8] for computing f in the Steiner tree network $\mathcal{N}_i = (t_i, S, \rho)$.

We now obtain a different lower bound on the computing capacity in the special case when the target function is the arithmetic sum. This lower bound is then used to give an alternative lower bound (in Theorem 4.3.7) on the computing capacity for the class of symmetric target functions. The bound obtained in Theorem 4.3.7 is sometimes better than that of Theorem 4.3.5, and sometimes worse (Example 4.3.8 illustrates instances

[8] From Theorem 4.3.3, r_i can be arbitrarily close to min-cut(t_i, f).

of both cases).

Theorem 4.3.6. *If $\mathcal{N}$ is a network with alphabet $\mathcal{A} = \{0, 1, \ldots, q-1\}$ and the arithmetic sum target function f, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) \geq \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_q P_{q,s}}$$

where $P_{q,s}$ denotes the smallest prime number greater than $s(q-1)$.

Proof. Let $p = P_{q,s}$ and let $\mathcal{N}'$ denote the same network as $\mathcal{N}$ but whose alphabet is $\mathbb{F}_p$, the finite field of order p.

Let $\epsilon > 0$. From Theorem 5.5.6, there exists a (k, n) solution for computing the $\mathbb{F}_p$-sum of the source messages in $\mathcal{N}'$ with an achievable computing rate satisfying

$$\frac{k}{n} \geq \min_{C \in \Lambda(\mathcal{N})} |C| - \epsilon.$$

This (k, n) solution can be repeated to derive a (ck, cn) solution for any integer $c \geq 1$ (note that edges in the network $\mathcal{N}$ carry symbols from the alphabet $\mathcal{A} = \{0, 1, \ldots, q-1\}$, while those in the network $\mathcal{N}'$ carry symbols from a larger alphabet $\mathbb{F}_p$). Any (ck, cn) solution for computing the $\mathbb{F}_p$-sum in $\mathcal{N}'$ can be 'simulated' in the network $\mathcal{N}$ by a $\left(ck, \lceil cn \log_q p \rceil\right)$ code (e.g. see [65]). Furthermore, since $p \geq s(q-1)+1$ and the source alphabet is $\{0, 1, \ldots, q-1\}$, the $\mathbb{F}_p$-sum of the source messages in network $\mathcal{N}$ is equal to their arithmetic sum. Thus, by choosing c large enough, the arithmetic sum target function is computed in $\mathcal{N}$ with an achievable computing rate of at least

$$\frac{\displaystyle\min_{C \in \Lambda(\mathcal{N})} |C|}{\log_q p} - 2\epsilon.$$

Since ϵ is arbitrary, the result follows. ■

Theorem 4.3.7. *If $\mathcal{N}$ is a network with alphabet $\mathcal{A} = \{0, 1, \ldots, q-1\}$ and a symmetric target function f, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) \geq \frac{\displaystyle\min_{C \in \Lambda(\mathcal{N})} |C|}{(q-1) \cdot \log_q P(s)}$$

where $P(s)$ is the smallest prime number[9] *greater than s.*

Proof. From Definition 4.1.9, it suffices to evaluate the histogram target function $\hat{f}$ for computing f. For any set of source messages $(x_1, x_2, \ldots, x_s) \in \mathcal{A}^s$, we have

$$\hat{f}\,(x_1, \ldots, x_s) = (c_0, c_1, \ldots, c_{q-1})$$

where $c_i = |\{j : x_j = i\}|$ for each $i \in \mathcal{A}$. Consider the network $\mathcal{N}' = (G, S, \rho)$ with alphabet $\mathcal{A}' = \{0, 1\}$. Then for each $i \in \mathcal{A}$, c_i can be evaluated by computing the arithmetic sum target function in $\mathcal{N}'$ where every source node σ_j is assigned the message 1 if $x_j = i$, and 0 otherwise. Since we know that

$$\sum_{i=0}^{q-1} c_i = s$$

the histogram target function $\hat{f}$ can be evaluated by computing the arithmetic sum target function $q-1$ times in the network $\mathcal{N}'$ with alphabet $\mathcal{A}' = \{0, 1\}$. Let $\epsilon > 0$. From Theorem 4.3.6 in the Appendix, there exists a (k, n) solution for computing the arithmetic sum target function in $\mathcal{N}'$ with an achievable computing rate of at least

$$\frac{k}{n} \geq \frac{\min\limits_{C \in \Lambda(\mathcal{N})} |C|}{\log_2 P(s)} - \epsilon.$$

The above (k, n) solution can be repeated to derive a (ck, cn) solution for any integer $c \geq 1$. Note that edges in the network $\mathcal{N}$ carry symbols from the alphabet $\mathcal{A} = \{0, 1, \ldots, q-1\}$, while those in the network $\mathcal{N}'$ carry symbols from $\mathcal{A}' = \{0, 1\}$. Any (ck, cn) code for computing the arithmetic sum function in $\mathcal{N}'$ can be simulated in the network $\mathcal{N}$ by a $(ck, \lceil cn \log_q 2 \rceil)$ code[10]. Thus by choosing c large enough, the above-mentioned code can be simulated in the network $\mathcal{N}$ to derive a solution for com-

[9] From Bertrand's Postulate [66, p.343], we have $P(s) \leq 2s$.

[10] To see details of such a simulation, we refer the interested reader to [65].

puting the histogram target function $\hat{f}$ with an achievable computing rate[11] of at least

$$\frac{1}{(q-1)} \cdot \frac{1}{\log_q 2} \cdot \frac{\min\limits_{C \in \Lambda(\mathcal{N})} |C|}{\log_2 P(s)} - 2\epsilon.$$

Since ϵ is arbitrary, the result follows. ■

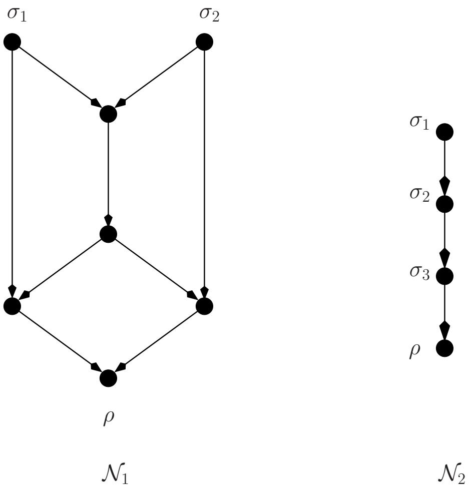


Figure 4.2: The Reverse Butterfly Network $\mathcal{N}_1$ has two binary sources $\{\sigma_1, \sigma_2\}$ and network $\mathcal{N}_2$ has three binary sources $\{\sigma_1, \sigma_2, \sigma_3\}$, each with $\mathcal{A} = \{0, 1\}$. Each network's receiver ρ computes the arithmetic sum of the source messages.

Example 4.3.8. Consider networks $\mathcal{N}_1$ and $\mathcal{N}_2$ in Figure 4.2, each with alphabet $\mathcal{A} = \{0, 1\}$ and the (symmetric) arithmetic sum target function f. Theorem 4.3.7 provides a larger lower bound on the computing capacity $\mathcal{C}_{\text{cod}}(\mathcal{N}_1, f)$ than Theorem 4.3.5, but a smaller lower bound on $\mathcal{C}_{\text{cod}}(\mathcal{N}_2, f)$.

- For network $\mathcal{N}_1$ (in Figure 4.2), we have $\max\limits_{C \in \Lambda(\mathcal{N}_1)} R_{C,f} = 3$ and $\min\limits_{C \in \Lambda(\mathcal{N}_1)} |C| = 2$, both of which occur, for example, when C consists of the two in-edges to the receiver ρ. Also, $(q-1)\log_q P(s, q) = \log_2 3$ and $\Pi(\mathcal{N}_1) = 3/2$, so

$$\begin{aligned} \mathcal{C}_{\text{cod}}(\mathcal{N}_1, f) &\geq (3/2)/\log_2 3 && \text{[from Theorem 4.3.5]} \\ \mathcal{C}_{\text{cod}}(\mathcal{N}_1, f) &\geq 2/\log_2 3 && \text{[from Theorem 4.3.7].} \end{aligned} \tag{4.18}$$

[11] Theorem 4.3.7 provides a uniform lower bound on the achievable computing rate for any symmetric function. Better lower bounds can be found by considering specific functions; for example Theorem 4.3.6 gives a better bound for the arithmetic sum target function.

In fact, we get the upper bound $\mathcal{C}_{\text{cod}}(\mathcal{N}_1, f) \leq 2/\log_2 3$ from Theorem 4.2.1, and thus from (4.18), $\mathcal{C}_{\text{cod}}(\mathcal{N}_1, f) = 2/\log_2 3$.

- For network $\mathcal{N}_2$, we have $\max\limits_{C \in \Lambda(\mathcal{N}_2)} R_{C,f} = 4$ and $\min\limits_{C \in \Lambda(\mathcal{N}_2)} |C| = 1$, both of which occur when $C = \{(\sigma_3, \rho)\}$. Also, $(q-1)\log_q P(s,q) = \log_2 5$ and $\Pi(\mathcal{N}_2) = 1$, so

$$\mathcal{C}_{\text{cod}}(\mathcal{N}_2, f) \geq 1/\log_2 4 \qquad \text{[from Theorem 4.3.5]}$$
$$\mathcal{C}_{\text{cod}}(\mathcal{N}_2, f) \geq 1/\log_2 5 \qquad \text{[from Theorem 4.3.7]}.$$

From Theorem 4.3.3, we have $\mathcal{C}_{\text{cod}}(\mathcal{N}_2, f) = 1/\log_2 4$.

Remark 4.3.9. An open question, pointed out in [59], is whether the coding capacity of a network can be irrational. Like the coding capacity, the computing capacity is the supremum of ratios k/n for which a (k,n) solution exists. Example 4.3.8 demonstrates that the computing capacity of a network (e.g. $\mathcal{N}_1$) with unit capacity links can be irrational when the target function is the arithmetic sum function.

4.4 On the tightness of the min-cut upper bound

In the previous section, Theorems 4.3.1 - 4.3.3 demonstrated three special instances for which the min-cut$(\mathcal{N}, f)$ upper bound is tight. In this section, we use Theorem 4.3.5 and Theorem 4.3.7 to establish further results on the tightness of the min-cut$(\mathcal{N}, f)$ upper bound for different classes of target functions.

The following lemma provides a bound on the footprint size $R_{I,f}$ for any divisible target function f.

Lemma 4.4.1. *For any divisible target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$ and any index set $I \subseteq \{1, 2, \ldots, s\}$, the footprint size satisfies*

$$R_{I,f} \leq |f(\mathcal{A}^s)|.$$

Proof. From the definition of a divisible target function, for any $I \subseteq \{1, 2, \ldots, s\}$, there exist maps f^I, f^{I^c}, and g such that

$$f(x) = g\left(f^I(x_I), f^{I^c}(x_{I^c})\right) \quad \forall\, x \in \mathcal{A}^s$$

where $I^c = \{1, 2, \ldots, s\} - I$. From the definition of the equivalence relation $\equiv$ (see Definition 4.1.4), it follows that $a, b \in \mathcal{A}^{|I|}$ belong to the same equivalence class whenever $f^I(a) = f^I(b)$. This fact implies that $R_{I,f} \le \left|f^I\left(\mathcal{A}^{|I|}\right)\right|$. We need $\left|f^I\left(\mathcal{A}^{|I|}\right)\right| \le |f\left(\mathcal{A}^s\right)|$ to complete the proof which follows from Definition 4.1.8(2). ■

Theorem 4.4.2. *If $\mathcal{N}$ is a network with a divisible target function f, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) \ge \frac{\Pi(\mathcal{N})}{|\mathcal{E}_i(\rho)|} \cdot \text{min-cut}(\mathcal{N}, f)$$

where $\mathcal{E}_i(\rho)$ denotes the set of in-edges of the receiver ρ.

Proof. Let $\mathcal{A}$ be the network alphabet. From Theorem 4.3.5,

$$\begin{aligned} \mathcal{C}_{\text{cod}}(\mathcal{N}, f) &\ge \Pi(\mathcal{N}) \cdot \min_{C \in \Lambda(\mathcal{N})} \frac{1}{\log_{|\mathcal{A}|} R_{C,f}} \\ &\ge \Pi(\mathcal{N}) \cdot \frac{1}{\log_{|\mathcal{A}|} |f\left(\mathcal{A}^s\right)|} \quad [\text{from Lemma 4.4.1}]. \end{aligned} \tag{4.19}$$

On the other hand, for any network $\mathcal{N}$, the set of edges $\mathcal{E}_i(\rho)$ is a cut that separates the set of sources S from ρ. Thus,

$$\begin{aligned} &\text{min-cut}(\mathcal{N}, f) \\ &\le \frac{|\mathcal{E}_i(\rho)|}{\log_{|\mathcal{A}|} R_{\mathcal{E}_i(\rho),f}} \qquad [\text{from (4.4)}] \\ &= \frac{|\mathcal{E}_i(\rho)|}{\log_{|\mathcal{A}|} |f\left(\mathcal{A}^s\right)|} \qquad [\text{from } I_{\mathcal{E}_i(\rho)} = S \text{ and Definition 4.1.5}]. \end{aligned} \tag{4.20}$$

Combining (4.19) and (4.20) completes the proof. ■

Theorem 4.4.3. *If $\mathcal{N}$ is a network with alphabet $\mathcal{A} = \{0, 1, \ldots, q-1\}$ and symmetric* *target function f, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) \geq \frac{\log_q \hat{R}_f}{(q-1)\cdot \log_q P(s)} \cdot \text{min-cut}(\mathcal{N}, f)$$

where $P(s)$ is the smallest prime number greater than s and[12]

$$\hat{R}_f = \min_{I \subseteq \{1,\ldots,s\}} R_{I,f}.$$

Proof. The result follows immediately from Theorem 4.3.7 and since for any network $\mathcal{N}$ and any target function f,

$$\begin{aligned} &\text{min-cut}(\mathcal{N}, f) \leq \\ &\frac{1}{\log_q \hat{R}_f} \cdot \min_{C \in \Lambda(\mathcal{N})} |C| && \text{[from (4.4) and the definition of } \hat{R}_f]. \end{aligned}$$

■

The following results provide bounds on the gap between the computing capacity and the min-cut for λ-exponential and λ-bounded functions (see Definition 4.1.10).

Theorem 4.4.4. *If $\lambda \in (0,1]$ and $\mathcal{N}$ is a network with a λ-exponential target function f, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) \geq \lambda \cdot \text{min-cut}(\mathcal{N}, f).$$

Proof. We have

$$\begin{aligned} &\text{min-cut}(\mathcal{N}, f) \\ &= \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{|\mathcal{A}|} R_{C,f}} \\ &\leq \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\lambda\, |I_C|} && [\text{from } f \text{ being } \lambda\text{-exponential}] \\ &= \frac{1}{\lambda} \cdot \text{min-cut}(\mathcal{N}) && [\text{from (4.3)}]. \end{aligned}$$

[12] From our assumption, $\hat{R}_f \geq 2$ for any target function f.

Therefore,

$$\frac{\text{min-cut}(\mathcal{N}, f)}{\mathcal{C}_{\text{cod}}(\mathcal{N}, f)} \leq \frac{1}{\lambda} \cdot \frac{\text{min-cut}(\mathcal{N})}{\mathcal{C}_{\text{cod}}(\mathcal{N}, f)} \leq \frac{1}{\lambda}$$

where the last inequality follows because a computing rate of min-cut$(\mathcal{N})$ is achievable for the identity target function from Theorem 4.3.1, and the computing capacity for any target function f is lower bounded by the computing capacity for the identity target function (since any target function can be computed from the identity function), i.e., $\mathcal{C}_{\text{cod}}(\mathcal{N}, f) \geq$ min-cut$(\mathcal{N})$. ■

Theorem 4.4.5. *Let $\lambda > 0$. If $\mathcal{N}$ is a network with alphabet $\mathcal{A}$ and a λ-bounded target function f, and all non-receiver nodes in the network $\mathcal{N}$ are sources, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) \geq \frac{\log_{|\mathcal{A}|} \hat{R}_f}{\lambda} \cdot \text{min-cut}(\mathcal{N}, f)$$

where

$$\hat{R}_f = \min_{I \subseteq \{1,\ldots,s\}} R_{I,f}.$$

Proof. For any network $\mathcal{N}$ such that all non-receiver nodes are sources, it follows from Edmond's Theorem [67, p.405, Theorem 8.4.20] that

$$\Pi(\mathcal{N}) = \min_{C \in \Lambda(\mathcal{N})} |C| \, .$$

Then,

$$\begin{aligned}
&\mathcal{C}_{\text{cod}}(\mathcal{N}, f) \\
&\geq \min_{C \in \Lambda(\mathcal{N})} |C| \cdot \min_{C \in \Lambda(\mathcal{N})} \frac{1}{\log_{|\mathcal{A}|} R_{C,f}} && [\text{from Theorem 4.3.5}] \\
&\geq \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\lambda} && [\text{from } f \text{ being } \lambda\text{-bounded}]. && (4.21)
\end{aligned}$$

On the other hand,

$$\begin{aligned}
&\text{min-cut}(\mathcal{N}, f) \\
&= \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{|\mathcal{A}|} R_{C,f}} \\
&\le \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{|\mathcal{A}|} \hat{R}_f} \qquad \text{[from the definition of } \hat{R}_f]. \qquad (4.22)
\end{aligned}$$

Combining (4.21) and (4.22) gives

$$\begin{aligned}
\frac{\text{min-cut}(\mathcal{N}, f)}{\mathcal{C}_{\text{cod}}(\mathcal{N}, f)} &\le \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{|\mathcal{A}|} \hat{R}_f} \cdot \frac{1}{\min\limits_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\lambda}} \\
&= \frac{\lambda}{\log_{|\mathcal{A}|} \hat{R}_f}.
\end{aligned}$$

■

Since the maximum and minimum functions are 1-bounded, and $\hat{R}_f = |\mathcal{A}|$ for each, we get the following corollary.

Corollary 4.4.6. *Let $\mathcal{A}$ be any ordered alphabet and let $\mathcal{N}$ be any network such that all non-receiver nodes in the network are sources. If the target function f is either the maximum or the minimum function, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) = \text{ min-cut}(\mathcal{N}, f).$$

Theorems 4.4.2 - 4.4.5 study the tightness of the min-cut$(\mathcal{N}, f)$ upper bound for different classes of target functions. In particular, we show that for λ-exponential (respectively, λ-bounded) target functions, the computing capacity $\mathcal{C}_{\text{cod}}(\mathcal{N}, f)$ is at least a constant fraction of the min-cut$(\mathcal{N}, f)$ for any constant λ and any network $\mathcal{N}$ (respectively, any network $\mathcal{N}$ where all non-receiver nodes are sources). The following theorem shows by means of an example target function f and a network $\mathcal{N}$, that the min-cut$(\mathcal{N}, f)$ upper bound cannot always approximate the computing capacity $\mathcal{C}_{\text{cod}}(\mathcal{N}, f)$ up to a constant fraction. Similar results are known in network coding as well as in

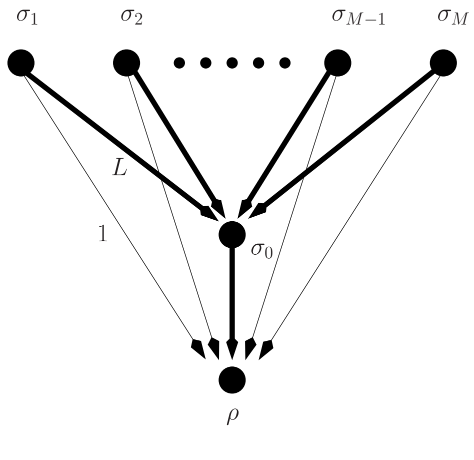


Figure 4.3: Network $\mathcal{N}_{M,L}$ has M binary sources $\{\sigma_1, \sigma_2, \ldots, \sigma_M\}$, with $\mathcal{A} = \{0, 1\}$, connected to the receiver node ρ via a relay σ_0. Each bold edge denotes L parallel capacity-one edges. ρ computes the arithmetic sum of the source messages.

multicommodity flow. It was shown in [61] that when s source nodes communicate independently with the same number of receiver nodes, there exist networks whose maximum multicommodity flow is $O(1/\log s)$ times a well known cut-based upper bound. It was shown in [68] that with network coding there exist networks whose maximum throughput is $O(1/\log s)$ times the best known cut bound (i.e. meagerness). Whereas these results do not hold for single-receiver networks (by Theorem 4.3.1), the following similar bound holds for network computing in single-receiver networks. The proof of Theorem 4.4.7 uses Lemma 4.7.1 which is presented in the Appendix.

Theorem 4.4.7. *For any $\epsilon > 0$, there exists a network $\mathcal{N}$ such that for the arithmetic sum target function f,*

$$\mathcal{C}_{cod}(\mathcal{N}, f) = O\left(\frac{1}{(\log s)^{1-\epsilon}}\right) \cdot \text{min-cut}(\mathcal{N}, f).$$

Proof. Consider the network $\mathcal{N}_{M,L}$ depicted in Figure 4.3 with alphabet $\mathcal{A} = \{0, 1\}$ and the arithmetic sum target function f. Then we have

$$\text{min-cut}(\mathcal{N}_{M,L}, f) = \min_{C \in \Lambda(\mathcal{N}_{M,L})} \frac{|C|}{\log_2 (|I_C| + 1)} \qquad \text{[from (4.5)]}.$$

Let m be the number of sources disconnected from the receiver ρ by a cut C in the network $\mathcal{N}_{M,L}$. For each such source σ, the cut C must contain the edge (σ, ρ) as well as either the L parallel edges (σ, σ_0) or the L parallel edges (σ_0, ρ). Thus,

$$\text{min-cut}(\mathcal{N}_{M,L}, f) = \min_{1 \le m \le M} \left\{ \frac{L+m}{\log_2(m+1)} \right\}. \tag{4.23}$$

Let m^* attain the minimum in (4.23) and define c^*= min-cut$(\mathcal{N}_{M,L}, f)$. Then,

$$\begin{aligned} c^*/\ln 2 &\ge \min_{1 \le m \le M} \left\{ \frac{m+1}{\ln(m+1)} \right\} && (4.24) \\ &\ge \min_{x \ge 2} \left\{ \frac{x}{\ln x} \right\} > \min_{x \ge 2} \left\{ \frac{x}{x-1} \right\} > 1, \\ L &= c^* \log_2 (m^* + 1) - m^* && \text{[from (4.23)]} \\ &\le c^* \log_2 \left(\frac{c^*}{\ln 2} \right) - \left(\frac{c^*}{\ln 2} - 1 \right) && (4.25) \end{aligned}$$

where (4.25) follows since the function $c^* \log_2 (x+1) - x$ attains its maximum value over $(0, \infty)$ at $x = (c^*/\ln 2) - 1$. Let us choose $L = \lceil (\log M)^{1-(\epsilon/2)} \rceil$. We have

$$\begin{aligned} &L = \\ &O(\text{min-cut}(\mathcal{N}_{M,L}, f) \log_2(\text{min-cut}(\mathcal{N}_{M,L}, f))) && \text{[from (4.25)]}, && (4.26) \\ &\text{min-cut}(\mathcal{N}_{M,L}, f) = \Omega((\log M)^{1-\epsilon}) && \text{[from (4.26)]}, && (4.27) \\ &\mathcal{C}_{\text{cod}}(\mathcal{N}_{M,L}, f) \\ &= O(1) && \text{[from Lemma 4.7.1]} \\ &= O\left(\frac{1}{(\log M)^{1-\epsilon}} \right) \cdot \text{min-cut}(\mathcal{N}_{M,L}, f) && \text{[from (4.27)]}. \end{aligned}$$

■

4.5 An example network

In this section, we evaluate the computing capacity for an example network and a target function (which is divisible and symmetric) and show that the min-cut bound is not tight. In addition, the example demonstrates that the lower bounds discussed in

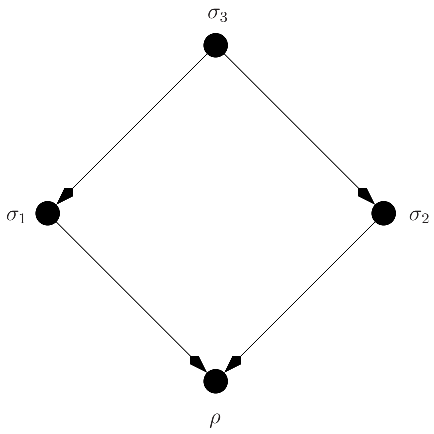


Figure 4.4: Network $\mathcal{N}_3$ has three binary sources, σ_1, σ_2, and σ_3 with $\mathcal{A} = \{0, 1\}$ and the receiver ρ computes the arithmetic sum of the source messages.

Section 4.3 are not always tight and illustrates the combinatorial nature of the computing problem.

Theorem 4.5.1. *The computing capacity of network $\mathcal{N}_3$ with respect to the arithmetic sum target function f is*

$$\mathcal{C}_{cod}(\mathcal{N}_3, f) = \frac{2}{1 + \log_2 3}.$$

Proof. For any (k, n) solution for computing f, let $w^{(1)}, w^{(2)}, w^{(3)} \in \{0, 1\}^k$ denote the message vectors generated by sources σ_1, σ_2, σ_3, respectively, and let $z_1, z_2 \in \{0, 1\}^n$ be the vectors carried by edges (σ_1, ρ) and (σ_2, ρ), respectively.

Consider any positive integers k, n such that k is even and

$$\frac{k}{n} \leq \frac{2}{1 + \log_2 3}. \tag{4.28}$$

Then we have

$$2^n \geq 3^{k/2} 2^{k/2}. \tag{4.29}$$

We will describe a (k, n) network code for computing f in the network $\mathcal{N}_3$. Define vectors $y^{(1)}, y^{(2)} \in \{0,1\}^k$ by:

$$y_i^{(1)} = \begin{cases} w_i^{(1)} + w_i^{(3)} & \text{if } 1 \leq i \leq k/2 \\ w_i^{(1)} & \text{if } k/2 \leq i \leq k \end{cases}$$

$$y_i^{(2)} = \begin{cases} w_i^{(2)} & \text{if } 1 \leq i \leq k/2 \\ w_i^{(2)} + w_i^{(3)} & \text{if } k/2 \leq i \leq k. \end{cases}$$

The first $k/2$ components of $y^{(1)}$ can take on the values $0,1,2$, and the last $k/2$ components can take on the values $0,1$, so there are a total of $3^{k/2}2^{k/2}$ possible values for $y^{(1)}$, and similarly for $y^{(2)}$. From (4.29), there exists a mapping that assigns unique values to z_1 for each different possible value of $y^{(1)}$, and similarly for z_2 and $y^{(2)}$. This induces a solution for $\mathcal{N}_3$ as summarized below.

The source σ_3 sends its full message vector $w^{(3)}$ ($k < n$) to each of the two nodes it is connected to. Source σ_1 (respectively, σ_2) computes the vector $y^{(1)}$ (respectively, $y^{(2)}$), then computes the vector z_1 (respectively, z_2), and finally sends z_1 (respectively, z_2) on its out-edge. The receiver ρ determines $y^{(1)}$ and $y^{(2)}$ from z_1 and z_2, respectively, and then computes $y^{(1)} + y^{(2)}$, whose i-th component is $w_i^{(1)} + w_i^{(2)} + w_i^{(3)}$, i.e., the arithmetic sum target function f. The above solution achieves a computing rate of k/n. From (4.28), it follows that

$$\mathcal{C}_{\text{cod}}(\mathcal{N}_3, f) \geq \frac{2}{1 + \log_2 3}. \tag{4.30}$$

We now prove a matching upper bound on the computing capacity $\mathcal{C}_{\text{cod}}(\mathcal{N}_3, f)$. Consider any (k,n) solution for computing the arithmetic sum target function f in network $\mathcal{N}_3$. For any $p \in \{0,1,2,3\}^k$, let

$$A_p = \{(z_1, z_2) : w^{(1)} + w^{(2)} + w^{(3)} = p\}.$$

That is, each element of A_p is a possible pair of input edge-vectors to the receiver when the function value equals p.

Let j denote the number of components of p that are either 0 or 3. Without loss of generality, suppose the first j components of p belong to $\{0,3\}$ and define $\tilde{w}^{(3)} \in \{0,1\}^k$

by

$$\tilde{w}_i^{(3)} = \begin{cases} 0 & \text{if } p_i \in \{0,1\} \\ 1 & \text{if } p_i \in \{2,3\}. \end{cases}$$

Let

$$T = \{(w^{(1)}, w^{(2)}) \in \{0,1\}^k \times \{0,1\}^k \ : \ w^{(1)} + w^{(2)} + \tilde{w}^{(3)} = p\}$$

and notice that

$$\left\{(z_1, z_2) : (w^{(1)}, w^{(2)}) \in T, w^{(3)} = \tilde{w}^{(3)}\right\} \subseteq A_p. \tag{4.31}$$

If $w^{(1)} + w^{(2)} + \tilde{w}^{(3)} = p$, then:

(i) $p_i - \tilde{w}_i^{(3)} = 0$ implies $w_i^{(1)} = w_i^{(2)} = 0$;

(ii) $p_i - \tilde{w}_i^{(3)} = 2$ implies $w_i^{(1)} = w_i^{(2)} = 1$;

(iii) $p_i - \tilde{w}_i^{(3)} = 1$ implies $(w_i^{(1)}, w_i^{(2)}) = (0,1)$ or $(1,0)$.

Thus, the elements of T consist of k-bit vector pairs $(w^{(1)}, w^{(2)})$ whose first j components are fixed and equal (i.e., both are 0 when $p_i = 0$ and both are 1 when $p_i = 3$), and whose remaining $k - j$ components can each be chosen from two possibilities (i.e., either $(0,1)$ or $(1,0)$, when $p_i \in \{1,2\}$). This observation implies that

$$|T| = 2^{k-j}. \tag{4.32}$$

Notice that if only $w^{(1)}$ changes, then the sum $w^{(1)} + w^{(2)} + w^{(3)}$ changes, and so z_1 must change (since z_2 is not a function of $w^{(1)}$) in order for the receiver to compute the target function. Thus, if $w^{(1)}$ changes and $w^{(3)}$ does not change, then z_1 must still change, regardless of whether $w^{(2)}$ changes or not. More generally, if the pair $(w^{(1)}, w^{(2)})$ changes, then the pair (z_1, z_2) must change. Thus,

$$\left|\left\{(z_1, z_2) : (w^{(1)}, w^{(2)}) \in T, w^{(3)} = \tilde{w}^{(3)}\right\}\right| \geq |T| \tag{4.33}$$

and therefore

$$\begin{aligned}
&|A_p| \\
&\geq \left|\left\{(z_1, z_2) : (w^{(1)}, w^{(2)}) \in T, w^{(3)} = \tilde{w}^{(3)}\right\}\right| && \text{[from (4.31)]} \\
&\geq |T| && \text{[from (4.33)]} \\
&= 2^{k-j}. && \text{[from (4.32)]} \qquad (4.34)
\end{aligned}$$

We have the following inequalities:

$$\begin{aligned}
4^n &\geq \left|\{(z_1, z_2) : w^{(1)}, w^{(2)}, w^{(3)} \in \{0,1\}^k\}\right| \\
&= \sum_{p \in \{0,1,2,3\}^k} |A_p| && (4.35) \\
&= \sum_{j=0}^{k} \sum_{\substack{p \in \{0,1,2,3\}^k \\ |\{i : p_i \in \{0,3\}\}| = j}} |A_p| \\
&\geq \sum_{j=0}^{k} \sum_{\substack{p \in \{0,1,2,3\}^k \\ |\{i : p_i \in \{0,3\}\}| = j}} 2^{k-j} && \text{[from (4.34)]} \\
&= \sum_{j=0}^{k} \binom{k}{j} 2^k 2^{k-j} \\
&= 6^k && (4.36)
\end{aligned}$$

where (4.35) follows since the A_p's must be disjoint in order for the receiver to compute the target function. Taking logarithms of both sides of (4.36), gives

$$\frac{k}{n} \leq \frac{2}{1 + \log_2 3}$$

which holds for all k and n, and therefore

$$\mathcal{C}_{\text{cod}}(\mathcal{N}_3, f) \leq \frac{2}{1 + \log_2 3}. \tag{4.37}$$

Combining (4.30) and (4.37) concludes the proof. ■

Corollary 4.5.2. *For the network $\mathcal{N}_3$ with the arithmetic sum target function f,*

$$\mathcal{C}_{cod}(\mathcal{N}_3, f) < \text{min-cut}(\mathcal{N}_3, f).$$

Proof. Consider the network $\mathcal{N}_3$ depicted in Figure 4.4 with the arithmetic sum target function f. It can be shown that the footprint size $R_{C,f} = |I_C| + 1$ for any cut C, and thus

$$\text{min-cut}(\mathcal{N}_3, f) = 1 \qquad\qquad [\text{from (4.5)}].$$

The result then follows immediately from Theorem 4.5.1. ■

Remark 4.5.3. In light of Theorem 4.5.1, we compare the various lower bounds on the computing capacity of the network $\mathcal{N}_3$ derived in Section 4.3 with the exact computing capacity. It can be shown that $\Pi(\mathcal{N}_3) = 1$. If f is the arithmetic sum target function, then

$$\begin{aligned}
\mathcal{C}_{\text{cod}}(\mathcal{N}_3, f) &\geq 1/2 && [\text{from Theorem 4.3.5}]\\
\mathcal{C}_{\text{cod}}(\mathcal{N}_3, f) &\geq 1/\log_2 5 && [\text{from Theorem 4.3.7}]\\
\mathcal{C}_{\text{cod}}(\mathcal{N}_3, f) &\geq 1/2 && [\text{from Theorem 4.4.2}].
\end{aligned}$$

Thus, this example demonstrates that the lower bounds obtained in Section 4.3 are not always tight and illustrates the combinatorial nature of the problem.

4.6 Conclusions

We examined the problem of network computing. The network coding problem is a special case when the function to be computed is the identity. We have focused on the case when a single receiver node computes a function of the source messages and have shown that while for the identity function the min-cut bound is known to be tight for all networks, a much richer set of cases arises when computing arbitrary functions, as the min-cut bound can range from being tight to arbitrarily loose. One key contribution of the chapter is to show the theoretical breadth of the considered topic, which we hope will lead to further research. This work identifies target functions (most notably, the arithmetic sum function) for which the min-cut bound is not always tight (even up to a constant factor) and future work includes deriving more sophisticated bounds for these scenarios. Extensions to computing with multiple receiver nodes, each computing a (possibly different) function of the source messages, are of interest.

4.7 Appendix

Define the function

$$Q \; : \; \prod_{i=1}^{M} \{0,1\}^k \longrightarrow \{0,1,\ldots,M\}^k$$

as follows. For every $a = (a^{(1)}, a^{(2)}, \ldots, a^{(M)})$ such that each $a^{(i)} \in \{0,1\}^k$,

$$Q(a)_j = \sum_{i=1}^{M} a_j^{(i)} \quad \text{for every } j \in \{1,2,\ldots,k\}. \tag{4.38}$$

We extend Q for $X \subseteq \prod_{i=1}^{M} \{0,1\}^k$ by defining $Q(X) = \{Q(a) \; : \; a \in X\}$.

We now present Lemma 4.7.1. The proof uses Lemma 4.7.2, which is presented thereafter. We define the following function which is used in the next lemma. Let

$$\gamma(x) = \mathcal{H}^{-1}\left(\frac{1}{2}\left(1 - \frac{1}{x}\right)\right) \bigcap \left[0, \frac{1}{2}\right] \quad \text{for } x \geq 1 \tag{4.39}$$

where $\mathcal{H}^{-1}$ denotes the inverse of the binary entropy function $\mathcal{H}(x) = -x \log_2 x - (1-x)\log_2(1-x)$. Note that $\gamma(x)$ is an increasing function of x.

Lemma 4.7.1. *If* $\lim_{M\to\infty} \frac{L}{\log_2 M} = 0$, *then* $\lim_{M\to\infty} \mathcal{C}_{cod}(\mathcal{N}_{M,L}, f) = 1$.

Proof. For any M and L, a solution with computing rate 1 is obtained by having each source σ_i send its message directly to ρ on the edge (σ_i, ρ). Hence $\mathcal{C}_{\text{cod}}(\mathcal{N}_{M,L}, f) \geq 1$. Now suppose that $\mathcal{N}_{M,L}$ has a (k,n) solution with computing rate $k/n > 1$ and for each $i \in \{1,2,\ldots,M\}$, let

$$g_i \; : \; \{0,1\}^k \longrightarrow \{0,1\}^n$$

be the corresponding encoding function on the edge (σ_i, ρ). Then for any $A_1, A_2, \ldots, A_M \subseteq \{0,1\}^k$, we have

$$\left(\prod_{i=1}^{M} \left| g_i \left(A_i \right) \right| \right) \cdot 2^{nL} \geq \left| Q \left(\prod_{i=1}^{M} A_i \right) \right| . \tag{4.40}$$

Each A_i represents a set of possible message vectors of source σ_i. The left-hand side of (4.40) is the maximum number of different possible instantiations of the information carried by the in-edges to the receiver ρ (i.e., $\left| g_i \left(A_i \right) \right|$ possible vectors on each edge (σ_i, ρ) and 2^{nL} possible vectors on the L parallel edges (σ_0, ρ)). The right-hand side of (4.40) is the number of distinct sum vectors that the receiver needs to discriminate, using the information carried by its in-edges.

For each $i \in \{1, 2, \ldots, M\}$, let $z_i \in \{0,1\}^n$ be such that $\left| g_i^{-1} \left(z_i \right) \right| \geq 2^{k-n}$ and choose $A_i = g_i^{-1} \left(z_i \right)$ for each i. Also, let $U^{(M)} = \prod_{i=1}^{M} A_i$. Then we have

$$\left| Q \big(U^{(M)} \big) \right| \leq 2^{nL} \qquad \text{[from } \left| g_i \left(A_i \right) \right| = 1 \text{ and (4.40)]}. \tag{4.41}$$

Thus (4.41) is a necessary condition for the existence of a (k, n) solution for computing f in the network $\mathcal{N}_{M,L}$. Lemma 4.7.2 shows that[13]

$$\left| Q \big(U^{(M)} \big) \right| \geq (M+1)^{\gamma(k/n)k} \tag{4.42}$$

where the function γ is defined in (4.39). Combining (4.41) and (4.42), any (k, n) solution for computing f in the network $\mathcal{N}_{M,L}$ with rate $r = k/n > 1$ must satisfy

$$r \, \gamma(r) \, \log_2(M+1) \leq \frac{1}{n} \log_2 \left| Q \big(U^{(M)} \big) \right| \leq L. \tag{4.43}$$

From (4.43), we have

$$r \, \gamma(r) \leq \frac{L}{\log_2(M+1)}. \tag{4.44}$$

[13] One can compare this lower bound to the upper bound $\left| Q \big(U^{(M)} \big) \right| \leq (M+1)^k$ which follows from (4.38).

The quantity $r\gamma(r)$ is monotonic increasing from 0 to ∞ on the interval $[1, \infty)$ and the right hand side of (4.44) goes to zero as $M \to \infty$. Thus, the rate r can be forced to be arbitrarily close to 1 by making M sufficiently large, i.e. $\mathcal{C}_{\text{cod}}(\mathcal{N}_{M,L}, f) \leq 1$. In summary,

$$\lim_{M \longrightarrow \infty} \mathcal{C}_{\text{cod}}(\mathcal{N}_{M,L}, f) = 1.$$

■

Lemma 4.7.2. *Let k, n, M be positive integers, with $k > n$. For each $i \in \{1, \dots, M\}$, let $A_i \subseteq \{0,1\}^k$ be such that $|A_i| \geq 2^{k-n}$ and let $U^{(M)} = \prod_{i=1}^{M} A_i$. Then,*

$$\left|Q\big(U^{(M)}\big)\right| \geq (M+1)^{\gamma(k/n)k}.$$

Proof. The result follows from Lemmas 4.7.4 and 4.7.7. ■

The remainder of this Appendix is devoted to the proofs of lemmas used in the proof of Lemma 4.7.2. Before we proceed, we need to define some more notation. For every $j \in \{1, 2, \dots, k\}$, define the map

$$h^{(j)} \ : \ \{0, 1, \dots, M\}^k \ \longrightarrow \ \{0, 1, \dots, M\}^k$$

by

$$\left(h^{(j)}(p)\right)_i = \begin{cases} \max\{0, p_i - 1\} & \text{if } i = j \\ p_i & \text{otherwise.} \end{cases} \tag{4.45}$$

That is, the map $h^{(j)}$ subtracts one from the j-th component of the input vector (as long as the result is non-negative) and leaves all the other components the same. For every $j \in \{1, 2, \dots, k\}$, define the map

$$\hat{\phi}^{(j)} : 2^{\{0,1\}^k} \ \times \ \{0,1\}^k \longrightarrow \{0,1\}^k$$

by

$$\hat{\phi}^{(j)}(A,a) = \begin{cases} h^{(j)}(a) & \text{if } h^{(j)}(a) \notin A \\ a & \text{otherwise} \end{cases} \tag{4.46}$$

for every $A \subseteq \{0,1\}^k$ and $a \in \{0,1\}^k$. Define

$$\phi^{(j)} : 2^{\{0,1\}^k} \longrightarrow 2^{\{0,1\}^k}$$

by

$$\phi^{(j)}(A) = \left\{ \hat{\phi}^{(j)}(A,a) \; : \; a \; \in \; A \right\}. \tag{4.47}$$

Note that

$$\left| \phi^{(j)}(A) \right| = |A|\,. \tag{4.48}$$

A set A is said to be *invariant* under the map $\phi^{(j)}$ if the set is unchanged when $\phi^{(j)}$ is applied to it, in which case from (4.46) and (4.47) we would have that for each $a \in A$,

$$h^{(j)}(a) \in A. \tag{4.49}$$

Lemma 4.7.3. *For any $A \subseteq \{0,1\}^k$ and all integers m and t such that $1 \leq m \leq t \leq k$, the set $\phi^{(t)}(\phi^{(t-1)}(\cdots \phi^{(1)}(A)))$ is invariant under the map $\phi^{(m)}$.*

Proof. For any $A' \subseteq \{0,1\}^k$, we have

$$\phi^{(i)}(\phi^{(i)}(A')) = \phi^{(i)}(A') \quad \forall\, i \in \{1,2,\ldots,k\}. \tag{4.50}$$

The proof of the lemma is by induction on t. For the base case $t = 1$, the proof is clear since $\phi^{(1)}(\phi^{(1)}(A)) = \phi^{(1)}(A)$ from (4.50). Now suppose the lemma is true for all $t < \tau$ (where $\tau \geq 2$). Now suppose $t = \tau$. Let $B = \phi^{(\tau-1)}(\phi^{(\tau-2)}(\cdots \phi^{(1)}(A)))$. Since $\phi^{(\tau)}(\phi^{(\tau)}(B)) = \phi^{(\tau)}(B)$ from (4.50), the lemma is true when $m = t = \tau$. In the following arguments, we take $m < \tau$. From the induction hypothesis, B is invariant under the map $\phi^{(m)}$, i.e.,

$$\phi^{(m)}(B) = B. \tag{4.51}$$

Consider any vector $c \in \phi^{(\tau)}(B)$. From (4.49), we need to show that $h^{(m)}(c) \in \phi^{(\tau)}(B)$. We have the following cases.

$c_\tau = 1$:

$$c, h^{(\tau)}(c) \in B \qquad \text{[from } c_\tau = 1,\ c \in \phi^{(\tau)}(B)\text{]} \tag{4.52}$$

$$h^{(m)}(c) \in B \qquad \text{[from (4.51), (4.52)]} \tag{4.53}$$

$$h^{(\tau)}\left(h^{(m)}(c)\right) = h^{(m)}\left(h^{(\tau)}(c)\right) \in B \qquad \text{[from (4.51), (4.52)]} \tag{4.54}$$

$$h^{(m)}(c) \in \phi^{(\tau)}(B) \qquad \text{[from (4.53), (4.54)].}$$

$c_\tau = 0$:

$$\exists\, b \in B \text{ with } h^{(\tau)}(b) = c \qquad \text{[from } c_\tau = 0,\ c \in \phi^{(\tau)}(B)\text{]} \tag{4.55}$$

$$h^{(m)}(b) \in B \qquad \text{[from (4.51), (4.55)]} \tag{4.56}$$

$$h^{(m)}\left(h^{(\tau)}(b)\right) = h^{(\tau)}\left(h^{(m)}(b)\right) \in \phi^{(\tau)}(B) \qquad \text{[from (4.56)]} \tag{4.57}$$

$$h^{(m)}\left(c\right) \in \phi^{(\tau)}(B) \qquad \text{[from (4.55), (4.57)].}$$

Thus, the lemma is true for $t = \tau$ and the induction argument is complete. ■

Let $A_1, A_2, \ldots, A_M \subseteq \{0,1\}^k$ be such that $|A_i| \geq 2^{k-n}$ for each i. Let $U^{(M)} = \prod_{i=1}^{M} A_i$ and extend the definition of $\phi^{(j)}$ in (4.47) to products by

$$\phi^{(j)}(U^{(M)}) = \prod_{i=1}^{M} \phi^{(j)}(A_i).$$

$U^{(M)}$ is said to be *invariant under* $\phi^{(j)}$ if

$$\phi^{(j)}(U^{(M)}) = U^{(M)}.$$

It can be verifed that $U^{(M)}$ is invariant under $\phi^{(j)}$ iff each A_i is invariant under $\phi^{(j)}$. For each $i \in \{1, 2, \ldots, M\}$, let

$$B_i = \phi^{(k)}(\phi^{(k-1)}(\cdots \phi^{(1)}(A_i)))$$

and from (4.48) note that

$$|B_i| = |A_i| \geq 2^{k-n}. \tag{4.58}$$

Let

$$V^{(M)} = \phi^{(k)}(\phi^{(k-1)}(\cdots \phi^{(1)}(U^{(M)}))) = \prod_{i=1}^{M} B_i$$

and recall the definition of the function Q (4.38).

Lemma 4.7.4.

$$\left|Q\big(U^{(M)}\big)\right| \geq \left|Q\big(V^{(M)}\big)\right|.$$

Proof. We begin by showing that

$$\left|Q\big(U^{(M)}\big)\right| \geq \left|Q\big(\phi^{(1)}(U^{(M)})\big)\right|. \tag{4.59}$$

For every $p \in \{0, 1, \ldots, M\}^{k-1}$, let

$$\begin{aligned} \varphi(p) &= \left\{r \in Q\big(U^{(M)}\big) : (r_2, \cdots, r_k) = p\right\} \\ \varphi_1(p) &= \left\{s \in Q\big(\phi^{(1)}(U^{(M)})\big) : (s_2, \cdots, s_k) = p\right\} \end{aligned}$$

and note that

$$Q\big(U^{(M)}\big) = \bigcup_{p \in \{0,1,\ldots,M\}^{k-1}} \varphi(p) \tag{4.60}$$

$$Q\big(\phi^{(1)}(U^{(M)})\big) = \bigcup_{p \in \{0,1,\ldots,M\}^{k-1}} \varphi_1(p) \tag{4.61}$$

where the unions are disjoint. We show that for every $p \in \{0, 1, \ldots, M\}^{k-1}$,

$$|\varphi(p)| \geq |\varphi_1(p)| \tag{4.62}$$

which by (4.60) and (4.61) implies (4.59).

If $|\varphi_1(p)| = 0$, then (4.62) is trivial. Now consider any $p \in \{0, 1, \ldots, M\}^{k-1}$ such that $|\varphi_1(p)| \geq 1$ and let

$$K_p = \max\{i \ : (i, p_1, \cdots, p_{k-1}) \in \varphi_1(p)\}.$$

Then we have

$$|\varphi_1(p)| \le K_p + 1. \tag{4.63}$$

Since $(K_p, p_1, \cdots, p_{k-1}) \in \varphi_1(p)$, there exists $(a^{(1)}, a^{(2)}, \ldots, a^{(M)}) \in U^{(M)}$ such that

$$\sum_{i=1}^{M} \hat{\phi}^{(1)}\left(A_i, a^{(i)}\right) = (K_p, p_1, \cdots, p_{k-1}). \tag{4.64}$$

Then from the definition of $\hat{\phi}^{(1)}$ in (4.46), there are K_p of the $a^{(i)}$'s from among $\{a^{(1)}, \ldots, a^{(M)}\}$ such that $a_1^{(i)} = 1$ and $\hat{\phi}^{(1)}\left(A_i, a^{(i)}\right) = a^{(i)}$. Let $I = \{i_1, \ldots, i_{K_p}\} \subseteq \{1, 2, \ldots, M\}$ be the index set for these vectors and let $\hat{a}^{(i)} = h^{(1)}(a^{(i)})$ for each $i \in I$. Then for each $i \in I$, we have

$$\begin{aligned} a^{(i)} &= \left(1, a_2^{(i)}, \ldots, a_k^{(i)}\right) \in A_i \\ \hat{a}^{(i)} &= \left(0, a_2^{(i)}, \ldots, a_k^{(i)}\right) \in A_i \text{ [from } \hat{\phi}^{(1)}\left(A_i, a^{(i)}\right) = a^{(i)}, \text{(4.46)]}. \end{aligned}$$

Let

$$R = \left\{ \sum_{i=1}^{M} b^{(i)} \ : \ \begin{array}{ll} b^{(i)} \in \{a^{(i)}, \hat{a}^{(i)}\} & \text{for } i \in I, \\ b^{(i)} = a^{(i)} & \text{for } i \notin I \end{array} \right\} \subseteq \varphi(p). \tag{4.65}$$

From (4.64) and (4.65), for every $r \in R$ we have

$$\begin{aligned} r_1 &\in \{0, 1, \ldots, |I|\}, \\ r_i &= p_i \ \ \forall\, i \in \{2, 3, \ldots, k\} \end{aligned}$$

and thus

$$|R| = |I| + 1 = K_p + 1. \tag{4.66}$$

Hence, we have

$$\begin{aligned} |\varphi(p)| &\ge |R| & &\text{[from (4.65)]} \\ &= K_p + 1 & &\text{[from (4.66)]} \\ &\ge |\varphi_1(p)| & &\text{[from (4.63)]} \end{aligned}$$

and then from (4.60) and (4.61), it follows that

$$\left|Q\big(U^{(M)}\big)\right| \geq \left|Q\big(\phi^{(1)}(U^{(M)})\big)\right|.$$

For any $A \subseteq \{0,1\}^k$ and any $j \in \{1,2,\dots,k\}$, we know that $\left|\phi^{(j)}(A)\right| \subseteq \{0,1\}^k$. Thus, the same arguments as above can be repeated to show that

$$\begin{aligned}\left|Q\big(\phi^{(1)}(U^{(M)})\big)\right| &\geq \left|Q\big(\phi^{(2)}(\phi^{(1)}(U^{(M)}))\big)\right| \\ &\geq \left|Q\big(\phi^{(3)}(\phi^{(2)}(\phi^{(1)}(U^{(M)})))\big)\right| \\ &\quad\vdots \\ &\geq \left|Q\big(\phi^{(k)}(\phi^{(k-1)}(\cdots\phi^{(1)}(U^{(M)})))\big)\right| \\ &= \left|Q\big(V^{(M)}\big)\right|.\end{aligned}$$

■

For any $s, r \in \mathbb{Z}^k$, we say that $s \leq r$ if $s_l \leq r_l$ for every $l \in \{1,2,\dots,k\}$.

Lemma 4.7.5. *Let* $p \in Q\big(V^{(M)}\big)$. *If* $q \in \{0,1,\dots,M\}^k$ *and* $q \leq p$, *then* $q \in Q\big(V^{(M)}\big)$.

Proof. Since $q \leq p$, it can be obtained by iteratively subtracting 1 from the components of p, i.e., there exist $t \geq 0$ and $i_1, i_2, \dots, i_t \in \{1,2,\dots,k\}$ such that

$$q = h^{(i_1)}\left(h^{(i_2)}\left(\cdots\left(h^{(i_t)}(p)\right)\right)\right).$$

Consider any $i \in \{1,2,\dots,k\}$. We show that $h^{(i)}(p) \in Q\big(V^{(M)}\big)$, which implies by induction that $q \in Q\big(V^{(M)}\big)$. If $p_i = 0$, then $h^{(i)}(p) = p$ and we are done. Suppose that $p_i > 0$. Since $p \in Q\big(V^{(M)}\big)$, there exists $b^{(j)} \in B_j$ for every $j \in \{1,2,\dots,M\}$ such that

$$p = \sum_{j=1}^{M} b^{(j)}$$

and $b_i^{(m)} = 1$ for some $m \in \{1,2,\dots,M\}$. From Lemma 4.7.3, $V^{(M)}$ is invariant under

$\phi^{(i)}$ and thus from (4.49), $h^{(i)}(b^{(m)}) \in B_m$ and

$$h^{(i)}(p) = \sum_{j=1}^{m-1} b^{(j)} + h^{(i)}(b^{(m)}) + \sum_{j=m+1}^{M} b^{(j)}$$

is an element of $Q\big(V^{(M)}\big)$. ■

The lemma below is presented in [44] without proof, as the proof is straightforward.

Lemma 4.7.6. *For all positive integers* k, n, M*, and* $\delta \in (0,1)$*,*

$$\min_{\substack{0 \,\le\, m_i \,\le\, M, \\ \sum_{i=1}^{k} m_i \,\ge\, \delta M k}} \prod_{i=1}^{k} (1 + m_i) \ge (M+1)^{\delta k} . \tag{4.67}$$

For any $a \in \{0,1\}^k$, let $|a|_H$ denote the Hamming weight of a, i.e., the number of non-zero components of a. The next lemma uses the function γ defined in (4.39).

Lemma 4.7.7.

$$\big|Q\big(V^{(M)}\big)\big| \ge (M+1)^{\gamma(k/n)k}.$$

Proof. Let $\delta = \gamma(k/n)$. The number of distinct elements in $\{0,1\}^k$ with Hamming weight at most $\lfloor \delta k \rfloor$ equals

$$\begin{aligned} \sum_{j=0}^{\lfloor \delta k \rfloor} \binom{k}{j} &\le 2^{k\mathcal{H}(\delta)} && \big[\text{from [69, p.15, Theorem 1]}\big] \\ &= 2^{(k-n)/2} && \big[\text{from (4.39)}\big]. \end{aligned}$$

For each $i \in \{1, 2, \ldots, M\}$, $|B_i| \ge 2^{k-n}$ from (4.58) and hence there exists $b^{(i)} \in B_i$ such that $\big|b^{(i)}\big|_H \ge \delta k$. Let

$$p = \sum_{i=1}^{M} b^{(i)} \ \in Q\big(V^{(M)}\big) .$$

It follows that $p_j \in \{0, 1, 2, \ldots, M\}$ for every $j \in \{1, 2, \ldots, k\}$, and

$$\sum_{j=1}^{k} p_j = \sum_{i=1}^{M} \left|b^{(i)}\right|_H \geq \delta M k. \tag{4.68}$$

The number of vectors q in $\{0, 1, \ldots, M\}^k$ such that $q \preceq p$ equals $\prod_{j=1}^{k} (1 + p_j)$, and from Lemma 4.7.5, each such vector is also in $Q\big(V^{(M)}\big)$. Therefore,

$$\begin{aligned} \left|Q\big(V^{(M)}\big)\right| &\geq \prod_{j=1}^{k} (1 + p_j) \\ &\geq (M+1)^{\delta k} && \big[\text{from (4.68) and Lemma 4.7.6}\big]. \end{aligned}$$

Since $\delta = \gamma(k/n)$, the result follows. ■

Chapter 4, in part, is a reprint of the material as it appears in R. Appuswamy, M. Franceschetti, N. Karamchandani and K. Zeger, “Network Coding for Computing: Cut-set bounds”, *IEEE Transactions on Information Theory*, vol. 57, no. 2, February 2011. The dissertation author was a primary investigator and author of this paper.

Chapter 5

Linear Codes, Target Function Classes, and Network Computing Capacity

We study the use of linear codes for network computing in single-receiver networks with various classes of target functions of the source messages. Such classes include reducible, injective, semi-injective, and linear target functions over finite fields. Computing capacity bounds and achievability are given with respect to these target function classes for network codes that use routing, linear coding, or nonlinear coding.

5.1 Introduction

Network coding concerns networks where each receiver demands a subset of messages generated by the source nodes and the objective is to satisfy the receiver demands at the maximum possible throughput rate. Accordingly, research efforts have studied coding gains over routing [32, 34, 68], whether linear codes are sufficient to achieve the capacity [70–73], and cut-set upper bounds on the capacity and the tightness of such bounds [34, 63, 68].

Network computing, on the other hand, considers a more general problem in which each receiver node demands a target function of the source messages [17, 45, 54, 56, 58, 74]. Most problems in network coding are applicable to network computing as well. Network computing problems arise in various networks including sensor networks and vehicular networks.

In [74], a network computing model was proposed where the network is modeled by a directed, acyclic graph with independent, noiseless links. The sources generate independent messages and a single receiver node computes a target function f of these messages. The objective is to characterize the maximum rate of computation, that is, the maximum number of times f can be computed per network usage. Each node in the network sends out symbols on its out-edges which are arbitrary, but fixed, functions of the symbols received on its in-edges and any messages generated at the node. In linear network computing, this encoding is restricted to be linear operations. Existing techniques for computing in networks use routing, where the codeword sent out by a node consists of symbols either received by that node, or generated by the node if it is a source (e.g. [75]).

In network coding, it is known that linear codes are sufficient to achieve the coding capacity for multicast networks [32], but they are not sufficient in general to achieve the coding capacity for non-multicast networks [71]. In network computing, it is known that when multiple receiver nodes demand a scalar linear target function of the source messages, linear network codes may not be sufficient in general for solvability [76]. However, it has been shown that for single-receiver networks, linear coding is sufficient for solvability when computing a scalar linear target function [56, 65]. Analogous to the coding capacity for network coding, the notion of computing capacity was defined

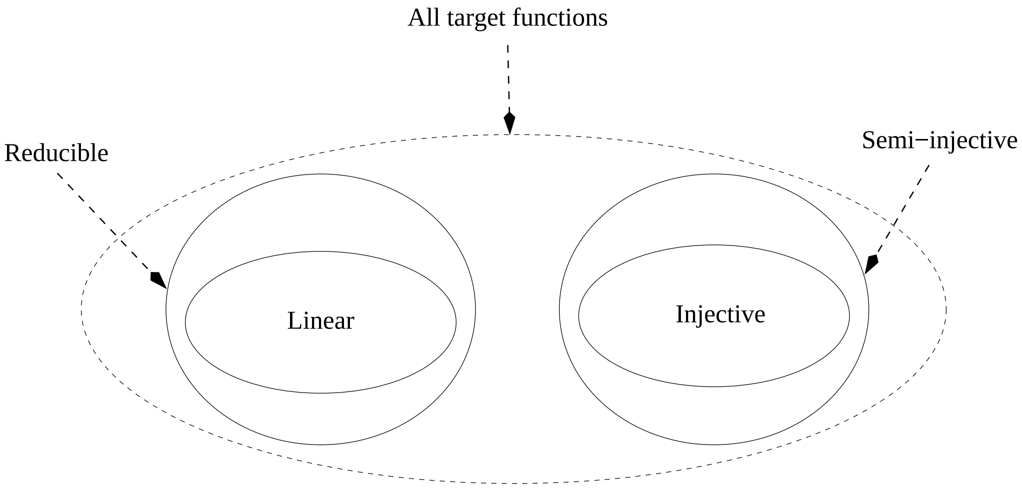


Figure 5.1: Decomposition of the space of all target functions into various classes.

for network computing in [17] and is the supremum of achievable rates of computing the network's target function.

One fundamental objective in the present chapter is to understand the performance of linear network codes for computing different types of target functions. Specifically, we compare the linear computing capacity with that of the (nonlinear) computing capacity and the routing computing capacity for various different classes of target functions in single-receiver networks. Such classes include reducible, injective, semi-injective, and linear target functions over finite fields. Informally, a target function is semi-injective if it uniquely maps at least one of its inputs, and a target function is reducible if it can be computed using a linear transformation followed by a function whose domain has a reduced dimension. Computing capacity bounds and achievability are given with respect to the target function classes studied for network codes that use routing, linear coding, or nonlinear coding.

Our specific contributions will be summarized next.

5.1.1 Contributions

Section 5.2 gives many of the formal definitions used in the chapter (e.g. target function classes and computing capacity types). We show that routing messages through the intermediate nodes in a network forces the receiver to obtain all the messages even though only a function of the messages is required (Theorem 5.2.10), and we bound the computing capacity gain of using nonlinear versus routing codes (Theorem 5.2.12).

In Section 5.3, we demonstrate that the performance of optimal linear codes may depend on how ‘linearity’ is defined (Theorem 5.3.2). Specifically, we show that the linear computing capacity of a network varies depending on which ring linearity is defined over on the source alphabet.

In Sections 5.4 and 5.5, we study the computing capacity gain of using linear coding over routing, and nonlinear coding over linear coding. In particular, we study various classes of target functions, including injective, semi-injective, reducible, and linear. The relationships between these classes is illustrated in Figure 5.1.

Section 5.4 studies linear coding for network computing. We show that if a target function is not reducible, then the linear computing capacity and routing computing capacity are equal whenever the source alphabet is a finite field (Theorem 5.4.8); the same result also holds for semi-injective target functions over rings. We also show that whenever a target function is injective, routing obtains the full computing capacity of a network (Theorem 5.4.9), although whenever a target function is neither reducible nor injective, there exists a network such that the computing capacity is larger than the linear computing capacity (Theorem 5.4.11). Thus for non-injective target functions that are not reducible, any computing capacity gain of using coding over routing must be obtained through nonlinear coding. This result is tight in the sense that if a target function is reducible, then there always exists a network where the linear computing capacity is larger than the routing capacity (Theorem 5.4.12). We also show that there exists a reducible target function and a network whose computing capacity is strictly greater than its linear computing capacity, which in turn is strictly greater than its routing computing capacity. (Theorem 5.4.14).

Section 5.5 focuses on computing linear target functions over finite fields. We characterize the linear computing capacity for linear target functions over finite fields in arbitrary networks (Theorem 5.5.6). We show that linear codes are sufficient for linear target functions and we upper bound the computing capacity gain of coding (linear or nonlinear) over routing (Theorem 5.5.7). This upper bound is shown to be achievable for every linear target function and an associated network, in which case the computing

Table 5.1: Summary of our main results for certain classes of target functions. The quantities $\mathcal{C}_{\text{cod}}(\mathcal{N}, f)$, $\mathcal{C}_{\text{lin}}(\mathcal{N}, f)$, and $\mathcal{C}_{\text{rout}}(\mathcal{N}, f)$ denote the computing capacity, linear computing capacity, and routing computing capacity, respectively, for a network $\mathcal{N}$ with s sources and target function f. The columns labeled f and $\mathcal{A}$ indicate contraints on the target function f and the source alphabet $\mathcal{A}$, respectively.

Result	f	$\mathcal{A}$	Location
$\forall f \; \forall \mathcal{N} \; \mathcal{C}_{\text{lin}}(\mathcal{N}, f) = \mathcal{C}_{\text{rout}}(\mathcal{N}, f)$	non-reducible	field	Theorem 5.4.8
	semi-injective	ring	
$\forall f \; \forall \mathcal{N} \; \mathcal{C}_{\text{cod}}(\mathcal{N}, f) = \mathcal{C}_{\text{rout}}(\mathcal{N}, f)$	injective		Theorem 5.4.9
$\forall f \; \exists \mathcal{N} \; \mathcal{C}_{\text{cod}}(\mathcal{N}, f) > \mathcal{C}_{\text{lin}}(\mathcal{N}, f)$	non-injective & non-reducible	field	Theorem 5.4.11
$\forall f \; \exists \mathcal{N} \; \mathcal{C}_{\text{lin}}(\mathcal{N}, f) > \mathcal{C}_{\text{rout}}(\mathcal{N}, f)$	reducible	ring	Theorem 5.4.12
$\exists f \; \exists \mathcal{N} \; \mathcal{C}_{\text{cod}}(\mathcal{N}, f) > \mathcal{C}_{\text{lin}}(\mathcal{N}, f) > \mathcal{C}_{\text{rout}}(\mathcal{N}, f)$	reducible		Theorem 5.4.14
$\forall f \; \forall \mathcal{N} \; \mathcal{C}_{\text{cod}}(\mathcal{N}, f) = \mathcal{C}_{\text{lin}}(\mathcal{N}, f) \leq s \, \mathcal{C}_{\text{rout}}(\mathcal{N}, f)$	linear	field	Theorem 5.5.7
$\forall f \; \exists \mathcal{N} \; \mathcal{C}_{\text{lin}}(\mathcal{N}, f) = s \, \mathcal{C}_{\text{rout}}(\mathcal{N}, f)$	linear	field	Theorem 5.5.8
$\exists f \; \exists \mathcal{N} \; \mathcal{C}_{\text{cod}}(\mathcal{N}, f)$ is irrational	arithmetic sum		Theorem 5.6.3

capacity is equal to the routing computing capacity times the number of network sources (Theorem 5.5.8).

Finally, Section 5.6 studies an illustrative example for the computing problem, namely the reverse butterfly network – obtained by reversing the direction of all the edges in the multicast butterfly network (the butterfly network studied in [32] illustrated the capacity gain of network coding over routing). For this network and the arithmetic sum target function, we evaluate the routing and linear computing capacity (Theorem 5.6.1) and the computing capacity (Theorem 5.6.3). We show that the latter is strictly larger than the first two, which are equal to each other. No network with such properties is presently known for network coding. Among other things, the reverse butterfly network also illustrates that the computing capacity can be a function of the coding alphabet (i.e. the domain of the target function f). In contrast, for network coding, the coding capacity and routing capacity are known to be independent of the coding alphabet used [59].

Our main results are summarized in Table 5.1.

5.2 Network model and definitions

In this chapter, a *network* $\mathcal{N} = (G, S, \rho)$ consists of a finite, directed acyclic multigraph $G = (\mathcal{V}, \mathcal{E})$, a set $S = \{\sigma_1, \ldots, \sigma_s\} \subseteq \mathcal{V}$ of s distinct *source nodes* and a single *receiver* $\rho \in \mathcal{V}$. We assume that $\rho \notin S$, and that the graph[1] G contains a directed path from every node in $\mathcal{V}$ to the receiver ρ. For each node $u \in \mathcal{V}$, let $\mathcal{E}_i(u)$ and $\mathcal{E}_o(u)$ denote the in-edges and out-edges of u respectively. We assume (without loss of generality) that if a network node has no in-edges, then it is a source node. If $e = (u, v) \in \mathcal{E}$, we will use the notation *head(*e*)* $= u$ and *tail(*e*)* $= v$.

An *alphabet* is a finite set of size at least two. Throughout this chapter, $\mathcal{A}$ will denote a *source alphabet* and $\mathcal{B}$ will denote a *receiver alphabet*. For any positive integer m, any vector $x \in \mathcal{A}^m$, and any $i \in \{1, 2, \ldots, m\}$, let x_i denote the i-th component of x. For any index set $I = \{i_1, i_2, \ldots, i_q\} \subseteq \{1, 2, \ldots, m\}$ with $i_1 < i_2 < \ldots < i_q$, let x_I denote the vector $(x_{i_1}, x_{i_2}, \ldots, x_{i_q}) \in \mathcal{A}^{|I|}$. Sometimes we view $\mathcal{A}$ as an algebraic structure such as a ring, i.e., with multiplication and addition. Throughout this chapter, vectors will always be taken to be row vectors. Let $\mathbb{F}_q$ denote a finite field of order q. A superscript t will denote the transpose for vectors and matrices.

5.2.1 Target functions

For a given network $\mathcal{N} = (G, S, \rho)$, we use s throughout the chapter to denote the number $|S|$ of receivers in $\mathcal{N}$. For given network $\mathcal{N}$, a *target function* is a mapping

$$f : \mathcal{A}^s \longrightarrow \mathcal{B}.$$

The goal in network computing is to compute f at the receiver ρ, as a function of the source messages. We will assume that all target functions depend on all the network sources (i.e. a target function cannot be a constant function of any one of its arguments). Some example target functions that will be referenced are listed in Table 5.2.

Definition 5.2.1. Let alphabet $\mathcal{A}$ be a ring. A target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$ is said to be *reducible* if there exists an integer λ satisfying $\lambda < s$, an $s \times \lambda$ matrix T with elements

[1] Throughout the remainder of the chapter, we use “graph” to mean a multigraph, and in the context of network computing we use “network” to mean a single-receiver network.

Table 5.2: Definitions of some target functions.

Target function f	Alphabet $\mathcal{A}$	$f\,(x_1, \ldots, x_s)$	Comments
identity	arbitrary	$(x_1, \ldots, x_s)$	$\mathcal{B} = \mathcal{A}^s$
arithmetic sum	$\{0, \ldots, q-1\}$	$x_1 + \cdots + x_s$	'$+$' is ordinary integer addition, $\mathcal{B} = \{0, 1, \cdots, s(q-1)\}$
mod r sum	$\{0, \ldots, q-1\}$	$x_1 \oplus \ldots \oplus x_s$	$\oplus$ is $\bmod r$ addition, $\mathcal{B} = \mathcal{A}$
linear	ring	$a_1x_1 + \ldots + a_sx_s$	arithmetic in the ring, $\mathcal{B} = \mathcal{A}$
maximum	ordered set	$\max\{x_1, \ldots, x_s\}$	$\mathcal{B} = \mathcal{A}$

in $\mathcal{A}$, and a map $g : \mathcal{A}^\lambda \longrightarrow \mathcal{B}$ such that for all $x \in \mathcal{A}^s$,

$$g(xT) = f(x). \tag{5.1}$$

Reducible target functions are not injective, since, for example, if x and y are distinct elements of the null-space of T, then

$$f(x) = g(xT) = g(0) = g(yT) = f(y).$$

Example 5.2.2. Suppose the alphabet is $\mathcal{A} = \mathbb{F}_2$ and the target function is

$$f : \mathbb{F}_2^3 \longrightarrow \{0, 1\},$$

where

$$f(x) = (x_1 + x_2)x_3.$$

Then, by choosing $\lambda = 2$,

$$T = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix},$$

and $g(y_1, y_2) = y_1 y_2$, we get

$$\begin{aligned} g(xT) &= g(x_1 + x_2, x_3) \\ &= (x_1 + x_2)x_3 \\ &= f(x). \end{aligned}$$

Thus the target function f is reducible.

Example 5.2.3. The notion of reducibility requires that for a target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$, the set $\mathcal{A}$ must be a ring. If we impose any ring structure to the domains of the identity, arithmetic sum, maximum, and minimum target functions, then these can be shown (via our Example 5.4.2 and Lemma 5.4.3) to be non-reducible.

5.2.2 Network computing and capacity

Let k and n be positive integers. Given a network $\mathcal{N}$ with source set S and alphabet $\mathcal{A}$, a *message generator* is any mapping

$$\alpha \; : \; S \longrightarrow \mathcal{A}^k.$$

For each source $\sigma_i \in S$, $\alpha(\sigma_i)$ is called a *message vector* and its components

$$\alpha(\sigma_i)_1 \, , \ldots , \alpha(\sigma_i)_k$$

are called *messages*[2].

Definition 5.2.4. A (k, n) *network code in a network* $\mathcal{N}$ consists of the following:

(i) *Encoding functions* $h^{(e)}$, for every out-edge $e \in \mathcal{E}_o(v)$ of every node $v \in \mathcal{V} - \rho$,

[2] For simplicity we assume each source has associated with it exactly one message vector, but all of the results in this chapter can readily be extended to the more general case.

of the form:

$$\begin{aligned} h^{(e)} &: \left(\prod_{\hat{e} \in \mathcal{E}_i(v)} \mathcal{A}^n \right) \times \mathcal{A}^k \longrightarrow \mathcal{A}^n \quad \text{if } v \text{ is a source node} \\ h^{(e)} &: \prod_{\hat{e} \in \mathcal{E}_i(v)} \mathcal{A}^n \longrightarrow \mathcal{A}^n \qquad\qquad \text{otherwise.} \end{aligned}$$

(ii) A *decoding function* ψ of the form:

$$\psi : \prod_{\hat{e} \in \mathcal{E}_i(v)} \mathcal{A}^n \longrightarrow \mathcal{B}^k.$$

Furthermore, given a (k, n) network code, every edge $e \in \mathcal{E}$ carries a vector z_e of at most n alphabet symbols[3], which is obtained by evaluating the encoding function $h^{(e)}$ on the set of vectors carried by the in-edges to the node and the node's message vector if the node is a source. The objective of the receiver is to compute the target function f of the source messages, for any arbitrary message generator α. More precisely, the receiver constructs a vector of k alphabet symbols, such that for each $i \in \{1, 2, \ldots, k\}$, the i-th component of the receiver's computed vector equals the value of the desired target function f, applied to the i-th components of the source message vectors, for any choice of message generator α.

Definition 5.2.5. Suppose in a network $\mathcal{N}$, the in-edges of ρ are $e_1, e_2, \ldots, e_{|\mathcal{E}_i(\rho)|}$. A (k, n) network code is said to *compute f in $\mathcal{N}$* if for each $j \in \{1, 2, \ldots, k\}$, and for each message generator α, the decoding function satisfies

$$\psi\left(z_{e_1}, \cdots, z_{e_{|\mathcal{E}_i(\rho)|}}\right)_j = f\Big(\left(\alpha(\sigma_1)_j, \cdots, \alpha(\sigma_s)_j\right)\Big). \tag{5.2}$$

If there exists a (k, n) code that computes f in $\mathcal{N}$, then the rational number k/n is said to be an *achievable computing rate*.

In the network coding literature, one definition of the *coding capacity* of a network is the supremum of all achievable coding rates [59]. We use an analogous defini-

[3] By default, we assume that edges carry exactly n symbols.

tion for the computing capacity.

Definition 5.2.6. The *computing capacity* of a network $\mathcal{N}$ with respect to a target function f is

$$\mathcal{C}_{\text{cod}}(\mathcal{N}, f) \;=\; \sup \left\{ \frac{k}{n} \;:\; \exists\, (k,n) \text{ network code that computes } f \text{ in } \mathcal{N} \right\}.$$

The notion of linear codes in networks is most often studied with respect to finite fields. Here we will sometimes use more general ring structures.

Definition 5.2.7. Let alphabet $\mathcal{A}$ be a ring. A (k,n) network code in a network $\mathcal{N}$ is said to be a *linear network code (over $\mathcal{A}$)* if the encoding functions are linear over $\mathcal{A}$.

Definition 5.2.8. The *linear computing capacity* of a network $\mathcal{N}$ with respect to target function f is

$$\mathcal{C}_{\text{lin}}(\mathcal{N}, f) \;=\; \sup \left\{ \frac{k}{n} \;:\; \exists\, (k,n) \text{ linear network code that computes } f \text{ in } \mathcal{N} \right\}.$$

The *routing computing capacity* $\mathcal{C}_{\textit{rout}}(\mathcal{N}, f)$ is defined similarly by restricting the encoding functions to routing. We call the quantity $\mathcal{C}_{\text{cod}}(\mathcal{N}, f) - \mathcal{C}_{\text{lin}}(\mathcal{N}, f)$ the *computing capacity gain* of using nonlinear coding over linear coding. Similar “gains”, such as, $\mathcal{C}_{\text{cod}}(\mathcal{N}, f) - \mathcal{C}_{\text{rout}}(\mathcal{N}, f)$ and $\mathcal{C}_{\text{lin}}(\mathcal{N}, f) - \mathcal{C}_{\text{rout}}(\mathcal{N}, f)$ are defined.

Note that Definition 5.2.7 allows linear codes to have nonlinear decoding functions. In fact, since the receiver alphabet $\mathcal{B}$ need not have any algebraic structure to it, linear decoding functions would not make sense in general. We do, however, examine a special case where $\mathcal{B} = \mathcal{A}$ and the target function is linear, in which case we show that linear codes with linear decoders can be just as good as linear codes with nonlinear decoders (Theorem 5.5.7).

Definition 5.2.9. A set of edges $C \subseteq \mathcal{E}$ in network $\mathcal{N}$ is said to *separate* sources $\sigma_{m_1}, \ldots, \sigma_{m_d}$ from the receiver ρ, if for each $i \in \{1, 2, \ldots, d\}$, every directed path from σ_{m_i} to ρ contains at least one edge in C. Define

$$I_C = \{i : C \text{ separates } \sigma_i \text{ from the receiver}\}\,.$$

The set C is said to be a *cut* in $\mathcal{N}$ if it separates at least one source from the receiver (i.e. $|I_C| \geq 1$). We denote by $\Lambda(\mathcal{N})$ the collection of all cuts in $\mathcal{N}$.

Since I_C is the number of sources disconnected by C and there are s sources, we have

$$|I_C| \leq s. \tag{5.3}$$

For network coding with a single receiver node and multiple sources (where the receiver demands all the source messages), routing is known to be optimal [63]. Let $\mathcal{C}_{\text{rout}}(\mathcal{N})$ denote the routing capacity of the network $\mathcal{N}$, or equivalently the routing computing capacity for computing the identity target function. It was observed in [63, Theorem 4.2] that for any single-receiver network $\mathcal{N}$,

$$\mathcal{C}_{\text{rout}}(\mathcal{N}) = \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{|I_C|}. \tag{5.4}$$

The following theorem shows that if the intermediate nodes in a network are restricted to perform routing, then in order to compute a target function the receiver is forced to obtain all the source messages. This fact motivates the use of coding for computing functions in networks.

Theorem 5.2.10. *If $\mathcal{N}$ is a network with target function f, then*

$$\mathcal{C}_{\textit{rout}}(\mathcal{N}, f) = \mathcal{C}_{\textit{rout}}(\mathcal{N}) \,.$$

Proof. Since any routing code that computes the identity target function can be used to compute any target function f, we have

$$\mathcal{C}_{\text{rout}}(\mathcal{N}, f) \geq \mathcal{C}_{\text{rout}}(\mathcal{N}) \,.$$

Conversely, it is easy to see that every component of every source message must be received by ρ in order to compute f, so

$$\mathcal{C}_{\text{rout}}(\mathcal{N}, f) \leq \mathcal{C}_{\text{rout}}(\mathcal{N}) \,.$$

■

Theorem 5.2.12 below gives a general upper bound on how much larger the computing capacity can be relative to the routing computing capacity. It will be shown later, in Theorem 5.5.7, that for linear target functions over finite fields, the bound in Theorem 5.2.12 can be tightened by removing the logarithm term.

Lemma 5.2.11. *If $\mathcal{N}$ is network with a target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$, then*

$$\mathcal{C}_{\textit{cod}}(\mathcal{N}, f) \leq (\log_2 |\mathcal{A}|) \min_{C \in \Lambda(\mathcal{N})} |C| \, .$$

Proof. Using [74, Theorem II.1], one finds the term min-cut$(\mathcal{N}, f)$ defined in [74, Equation (3)] in terms of a quantity $R_{I_C,f}$, which in turn is defined in [74, Definition 1.5]. Since target functions are restricted to not being constant functions of any of their arguments, we have $R_{I_C,f} \geq 2$, from which the result follows. ■

Theorem 5.2.12. *If $\mathcal{N}$ is network with a target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$, then*

$$\mathcal{C}_{\textit{cod}}(\mathcal{N}, f) \leq s \; (\log_2 |\mathcal{A}|) \; \mathcal{C}_{\textit{rout}}(\mathcal{N}, f)$$

Proof.

$$\begin{aligned} \mathcal{C}_{\text{cod}}(\mathcal{N}, f) &\leq (\log_2 |\mathcal{A}|) \min_{C \in \Lambda(\mathcal{N})} |C| && \text{[from Lemma 5.2.11]} \\ &\leq s \; (\log_2 |\mathcal{A}|) \; \mathcal{C}_{\text{rout}}(\mathcal{N}, f) \, . && \text{[from (5.3), (5.4), and Theorem 5.2.10]} \end{aligned}$$

■

5.3 Linear coding over different ring alphabets

Whereas the size of a finite field characterizes the field, there are, in general, different rings of the same size, so one must address whether the linear computing capacity of a network might depend on which ring is chosen for the alphabet. In this section, we illustrate this possibility with a specific computing problem.

Let $\mathcal{A} = \{a_0, a_1, a_2, a_3\}$ and let $f : \mathcal{A}^2 \longrightarrow \{0, 1, 2\}$ be as defined in Table 5.3. We consider different rings R of size 4 for $\mathcal{A}$ and evaluate the linear computing capacity of the network $\mathcal{N}_4$ shown in Figure 5.2 with respect to the target function f. Specifically, we let R be either the ring $\mathbb{Z}_4$ of integers modulo 4 or the product ring $\mathbb{Z}_2 \times \mathbb{Z}_2$ of 2-dimensional binary vectors. Denote the linear computing capacity here by

Table 5.3: Definition of the 4-ary map f.

f	a_0	a_1	a_2	a_3
a_0	0	1	1	2
a_1	1	0	2	1
a_2	1	2	0	1
a_3	2	1	1	0

$$\mathcal{C}_{\text{lin}}(\mathcal{N}_4)^R = \sup \left\{ \frac{k}{n} \ : \ \exists \ (k, n) \ R\text{-linear code that computes } f \text{ in } \mathcal{N} \right\}.$$

The received vector z at ρ can be viewed as a function of the source vectors generated at σ_1 and σ_2. For any (k, n) R-linear code, there exist $k \times n$ matrices M_1 and M_2 such that z can be written as

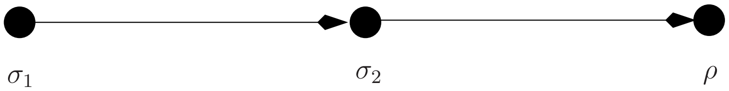


Figure 5.2: Network $\mathcal{N}_4$ has two sources σ_1 and σ_2 and a receiver ρ.

$$z(\alpha(\sigma_1)\,, \alpha(\sigma_2)) = \alpha(\sigma_1)\, M_1 + \alpha(\sigma_2)\, M_2. \tag{5.5}$$

Let $m_{i,1}, \cdots, m_{i,k}$ denote the row vectors of M_i, for $i \in \{1,2\}$.

Lemma 5.3.1. *Let $\mathcal{A}$ be the ring $\mathbb{Z}_4$ and let $f : \mathcal{A}^2 \longrightarrow \{0,1,2\}$ be the target function shown in Table 5.3, where $a_i = i$, for each i. If a (k,n) linear code over $\mathcal{A}$ computes f in $\mathcal{N}_4$ and ρ receives a zero vector, then $\alpha(\sigma_1) = \alpha(\sigma_2) \in \{0,2\}^k$.*

Proof. If $\alpha(\sigma_1) = \alpha(\sigma_2) = 0$, then ρ receives a 0 by (5.5) and must decode a 0 since $f((0,0)) = 0$ (from Table 5.3). Thus, ρ always decodes a 0 upon receiving a 0. But $f((x_1,x_2)) = 0$ if and only if $x_1 = x_2$ (from Table 5.3), so whenever ρ receives a 0, the source messages satisfy $\alpha(\sigma_1) = \alpha(\sigma_2)$.

Now suppose, contrary to the lemma's assertion, that there exist messages $\alpha(\sigma_1)$ and $\alpha(\sigma_2)$ such that $z(\alpha(\sigma_1), \alpha(\sigma_2)) = 0$ and $\alpha(\sigma_1)_j \notin \{0,2\}$ for some $j \in \{1,\ldots,k\}$. Since $\alpha(\sigma_1)_j$ is invertible in $\mathbb{Z}_4$ (it is either 1 or 3), we have from (5.5) that

$$m_{1,j} = \sum_{\substack{i=1\\ i\neq j}}^{k} -\alpha(\sigma_1)_j^{-1}\,\alpha(\sigma_1)_i\, m_{1,i} + \sum_{i=1}^{k} -\alpha(\sigma_1)_j^{-1}\,\alpha(\sigma_2)_i\, m_{2,i} \tag{5.6}$$

$$= y^{(1)} M_1 + y^{(2)} M_2 \tag{5.7}$$

where $y^{(1)}$ and $y^{(2)}$ are k-dimensional vectors defined by

$$y_i^{(1)} = \begin{cases} -\alpha(\sigma_1)_j^{-1}\,\alpha(\sigma_1)_i & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

$$y_i^{(2)} = -\alpha(\sigma_1)_j^{-1}\,\alpha(\sigma_2)_i\,. \tag{5.8}$$

Also, define the k-dimensional vector x by

$$x_i = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j. \end{cases} \tag{5.9}$$

We have from (5.5) that $z(x,0) = m_{1,j}$ and from (5.5) and (5.7) that $z(y^{(1)}, y^{(2)}) = m_{1,j}$. Thus, in order for the code to compute f, we must have $f(x_j, 0) = f(y_j^{(1)}, y_j^{(2)})$. But

$f(x_j, 0) = f(1, 0) = 1$ and

$$\begin{aligned} f(y_j^{(1)}, y_j^{(2)}) &= f(0, -\alpha(\sigma_1)_j^{-1}\,\alpha(\sigma_2)_j) \\ &= f(0, -\alpha(\sigma_1)_j^{-1}\,\alpha(\sigma_1)_j) && [\text{from } \alpha(\sigma_1) = \alpha(\sigma_2)] \\ &= f(0, -1) \\ &= f(0, 3) && [\text{from } 3 = -1 \text{ in } \mathbb{Z}_4] \\ &= 2 && [\text{from Table 5.3}], \end{aligned}$$

a contradiction. Thus, $\alpha(\sigma_1) \in \{0, 2\}^k$. ■

Theorem 5.3.2. *The network $\mathcal{N}_4$ in Figure 5.2 with alphabet $\mathcal{A} = \{a_0, a_1, a_2, a_3\}$ and target function $f : \mathcal{A}^2 \longrightarrow \{0, 1, 2\}$ shown in Table 5.3, satisfies*

$$\mathcal{C}_{\mathit{lin}}(\mathcal{N}_4, f)^{\mathbb{Z}_4} \leq \frac{2}{3}$$
$$\mathcal{C}_{\mathit{lin}}(\mathcal{N}_4, f)^{\mathbb{Z}_2 \times \mathbb{Z}_2} = 1.$$

(For $\mathcal{A} = \mathbb{Z}_4$, we identify $a_i = i$, for each i, and for $\mathcal{A} = \mathbb{Z}_2 \times \mathbb{Z}_2$, we identify each a_i with the 2-bit binary representation of i.)

Proof. Consider a (k, n) $\mathbb{Z}_2 \times \mathbb{Z}_2$-linear code that computes f. From (5.5), we have $z(x, 0) = 0$ whenever $xM_1 = 0$. Since $f((0, 0)) \neq f((x_i, 0))$ (whenever $x_i \neq 0$), it must therefore be the case that $xM_1 = 0$ only when $x = 0$, or in other words, the rows of M_1 must be independent, so $n \geq k$. Thus,

$$\mathcal{C}_{\text{lin}}(\mathcal{N}, f)^{\mathbb{Z}_2 \times \mathbb{Z}_2} \leq 1. \tag{5.10}$$

Now suppose that $\mathcal{A}$ is the ring $\mathbb{Z}_2 \times \mathbb{Z}_2$ where, $a_0 = (0, 0)$, $a_1 = (0, 1)$, $a_2 = (1, 0)$, and $a_3 = (1, 1)$ and let $\oplus$ denote the addition over $\mathcal{A}$. For any $x \in \mathcal{A}^2$, the value $f(x)$, as defined in Table 5.3, is seen to be the Hamming distance between x_1 and x_2. If $k = n = 1$ and $M_1 = M_2 = [a_3]$ (i.e., the 1×1 identity matrix), then ρ receives $x_1 \oplus x_2$ from which f can be computed by summing its components. Thus, a computing rate of

$k/n = 1$ is achievable. From (5.10), it then follows that

$$\mathcal{C}_{\text{lin}}(\mathcal{N}, f)^{\mathbb{Z}_2 \times \mathbb{Z}_2} = 1.$$

We now prove that $\mathcal{C}_{\text{lin}}(\mathcal{N}, f)^{\mathbb{Z}_4} \leq 2/3$. Let $\mathcal{A}$ denote the ring $\mathbb{Z}_4$ where $a_i = i$ for $0 \leq i \leq 3$. For a given (k, n) linear code over $\mathcal{A}$ that computes f, the n-dimensional vector received by ρ can be written as in (5.5). Let $\mathcal{K}$ denote the collection of all message vector pairs $(\alpha(\sigma_1), \alpha(\sigma_2))$ such that $z(\alpha(\sigma_1), \alpha(\sigma_2)) = 0$. Define the $2k \times n$ matrix

$$M = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix}$$

and notice that $\mathcal{K} = \{y \in \mathcal{A}^{2k} : yM = 0\}$. Then,

$$\begin{aligned}
4^n &= |\mathcal{A}|^n \\
&\geq \left|\{yM : y \in \mathcal{A}^{2k}\}\right| && [\text{from } y \in \mathcal{A}^{2k} \Longrightarrow yM \in \mathcal{A}^n] \\
&\geq \frac{|\mathcal{A}|^{2k}}{|\mathcal{K}|} && [\text{from } y^{(1)}, y^{(2)} \in \mathcal{A}^{2k},\ y^{(1)}M = y^{(2)}M \Longrightarrow y^{(1)} - y^{(2)} \in \mathcal{K}] \\
&\geq \frac{|\mathcal{A}|^{2k}}{2^k} && [\text{from Lemma 5.3.1}] \\
&= 4^{3k/2}. && [\text{from } |\mathcal{A}| = 4]
\end{aligned}$$

Thus, $k/n \leq 2/3$, so $\mathcal{C}_{\text{lin}}(\mathcal{N}_4, f)^{\mathbb{Z}_4} \leq \frac{2}{3}$. ■

5.4 Linear network codes for computing target functions

Theorem 5.2.10 showed that if intermediate network nodes use routing, then a network's receiver learns all the source messages irrespective of the target function it demands. In Section 5.4.1, we prove a similar result when the intermediate nodes use linear network coding. It is shown that whenever a target function is not reducible the linear computing capacity coincides with the routing capacity and the receiver must learn all the source messages. We also show that there exists a network such that the computing capacity is larger than the routing capacity whenever the target function is non-injective. Hence, if the target function is not reducible, such capacity gain must be obtained from nonlinear coding. Section 5.4.2 shows that linear codes may provide a computing capacity gain over routing for reducible target functions and that linear codes may not suffice to obtain the full computing capacity gain over routing.

5.4.1 Non-reducible target functions

Verifying whether or not a given target function is reducible may not be easy. We now define a class of target functions that are easily shown to not be reducible.

Definition 5.4.1. A target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$ is said to be *semi-injective* if there exists $x \in \mathcal{A}^s$ such that $f^{-1}(\{f(x)\}) = \{x\}$.

Note that injective functions are semi-injective.

Example 5.4.2. If f is the arithmetic sum target function, then f is semi-injective (since $f(x) = 0$ implies $x = 0$) but not injective (since $f(0,1) = f(1,0) = 1$). Other examples of semi-injective target functions include the identity, maximum, and minimum functions.

Lemma 5.4.3. *If alphabet $\mathcal{A}$ is a ring, then semi-injective target functions are not reducible.*

Proof. Suppose that a target function f is reducible. Then there exists an integer λ satisfying $\lambda < s$, matrix $T \in \mathcal{A}^{s\times\lambda}$, and map $g : \mathcal{A}^\lambda \longrightarrow \mathcal{B}$ such that

$$g(xT) = f(x) \ \text{ for each } x \in \mathcal{A}^s. \tag{5.11}$$

Since $\lambda < s$, there exists a non-zero $d \in \mathcal{A}^s$ such that $dT = 0$. Then for each $x \in \mathcal{A}^s$,

$$f(d+x) = g((d+x)T) = g(xT) = f(x) \tag{5.12}$$

so f is not semi-injective. ■

Definition 5.4.4. Let $\mathcal{A}$ be a finite field and let $\mathcal{M}$ be a subspace of the vector space $\mathcal{A}^s$ over the scalar field $\mathcal{A}$. Let

$$\mathcal{M}^{\perp} = \left\{ y \in \mathcal{A}^s : xy^t = 0 \text{ for all } x \in \mathcal{M} \right\}$$

and let $\dim(\mathcal{M})$ denote the dimension of $\mathcal{M}$ over $\mathcal{A}$.

Lemma 5.4.5. [4] *If $\mathcal{A}$ is a finite field and $\mathcal{M}$ is a subspace of vector space $\mathcal{A}^s$, then $(\mathcal{M}^{\perp})^{\perp}$ = $\mathcal{M}$.*

Lemma 5.4.6 will be used in Theorem 5.4.8. The lemma states an alternative characterization of reducible target functions when the source alphabet is a finite field and of semi-injective target functions when the source alphabet is a group.

Lemma 5.4.6. *Let $\mathcal{N}$ be a network with target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$ and alphabet $\mathcal{A}$.*

(i) Let $\mathcal{A}$ be a finite field. f is reducible if and only if there exists a non-zero $d \in \mathcal{A}^s$ such that for each $a \in \mathcal{A}$ and each $x \in \mathcal{A}^s$,

$$f(ad + x) = f(x).$$

(ii) Let $\mathcal{A}$ be a group. f is semi-injective if and only if there exists $x \in \mathcal{A}^s$ such that for every non-zero $d \in \mathcal{A}^s$,

$$f(d + x) \neq f(x).$$

[4] This lemma is a standard result in coding theory regarding dual codes over finite fields, even though the operation xy^t is not an inner product (e.g. [77, Theorem 7.5] or [78, Corollary 3.2.3]). An analogous result for orthogonal complements over inner product spaces is well known in linear algebra (e.g. [79, Theorem 5 on pg. 286]).

(The arithmetic in $ad+x$ and $d+x$ is performed component-wise over the corresponding $\mathcal{A}$.)

Proof. (i) If f is reducible, then there exists an integer λ satisfying $\lambda < s$, matrix $T \in \mathcal{A}^{s\times\lambda}$, and map $g : \mathcal{A}^{\lambda} \longrightarrow \mathcal{B}$ such that

$$g(xT) = f(x) \text{ for each } x \in \mathcal{A}^s. \tag{5.13}$$

Since $\lambda < s$, there exists a non-zero $d \in \mathcal{A}^s$ such that $dT = 0$. Then for each $a \in \mathcal{A}$ and each $x \in \mathcal{A}^s$,

$$f(ad + x) = g((ad + x)T) = g(xT) = f(x). \tag{5.14}$$

Conversely, suppose that there exists a non-zero d such that (5.14) holds for every $a \in \mathcal{A}$ and every $x \in \mathcal{A}^s$ and let $\mathcal{M}$ be the one-dimensional subspace of $\mathcal{A}^s$ spanned by d. Then

$$f(t + x) = f(x) \text{ for every } t \in \mathcal{M}, x \in \mathcal{A}^s. \tag{5.15}$$

Note that $\dim(\mathcal{M}^{\perp}) = s - 1$. Let $\lambda = s - 1$, let $T \in \mathcal{A}^{s\times\lambda}$ be a matrix such that its columns form a basis for $\mathcal{M}^{\perp}$, and let $\mathcal{R}_T$ denote the row space of T. Define the map

$$g : \mathcal{R}_T \longrightarrow f(\mathcal{A}^s)$$

as follows. For any $y \in \mathcal{R}_T$ such that $y = xT$ for $x \in \mathcal{A}^s$, let

$$g(y) = g(xT) = f(x). \tag{5.16}$$

Note that if $y = x^{(1)}T = x^{(2)}T$ for $x^{(1)} \neq x^{(2)}$, then

$$\begin{aligned} (x^{(1)} - x^{(2)})T &= 0 \\ x^{(1)} - x^{(2)} &\in (\mathcal{M}^{\perp})^{\perp} && \text{[from construction of } T\text{]} \\ x^{(1)} - x^{(2)} &\in \mathcal{M} && \text{[from Lemma 5.4.5]} \\ f(x^{(1)}) &= f((x^{(1)} - x^{(2)}) + x^{(2)}) \\ &= f(x^{(2)}). && \text{[from (5.15)]} \end{aligned}$$

Thus g is well defined. Then from (5.16) and Definition 5.2.1, f is reducible.

(ii) Since f is semi-injective, there exists a $x \in \mathcal{A}^s$ such that $\{x\} = f^{-1}(\{f(x)\})$, which in turn is true if and only if for each non-zero $d \in \mathcal{A}^s$, we have $f(d + x) \neq f(x)$. ■

The following example shows that if the alphabet $\mathcal{A}$ is not a finite field, then the assertion in Lemma 5.4.6(i) may not be true.

Example 5.4.7. Let $\mathcal{A} = \mathbb{Z}_4$, let $f : \mathcal{A} \longrightarrow \mathcal{A}$ be the target function defined by $f(x) = 2x$, and let $d = 2$. Then, for all $a \in \mathcal{A}$,

$$\begin{aligned} f(2a + x) &= 2(2a + x) \\ &= 2x && \text{[from } 4 = 0 \text{ in } \mathbb{Z}_4\text{]} \\ &= f(x) \end{aligned}$$

but, f is not reducible, since $s = 1$.

Theorem 5.4.8 establishes for a network with a finite field alphabet, whenever the target function is not reducible, linear computing capacity is equal to the routing computing capacity, and therefore if a linear network code is used, the receiver ends up learning all the source messages even though it only demands a function of these messages.

For network coding (i.e. when f is the identity function), many multi-receiver networks have a larger linear capacity than their routing capacity. However, all single-receiver networks are known to achieve their coding capacity with routing [63]. For network computing, the next theorem shows that with non-reducible target functions there is no advantage to using linear coding over routing.[5]

Theorem 5.4.8. *Let $\mathcal{N}$ be a network with target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$ and alphabet $\mathcal{A}$. If $\mathcal{A}$ is a finite field and f is not reducible, or $\mathcal{A}$ is a ring with identity and f is semi-injective, then*

$$\mathcal{C}_{\mathit{lin}}(\mathcal{N}, f) = \mathcal{C}_{\mathit{rout}}(\mathcal{N}, f)\,.$$

Proof. Since any routing code is in particular a linear code,

$$\mathcal{C}_{\text{lin}}(\mathcal{N}, f) \geq \mathcal{C}_{\text{rout}}(\mathcal{N}, f)\,.$$

Now consider a (k, n) linear code that computes the target function f in $\mathcal{N}$ and let C be a cut. We will show that for any two collections of source messages, if the messages agree at sources not separated from ρ by C and the vectors agree on edges in C, then there exist two other source message collections with different target function values, such that the receiver ρ cannot distinguish this difference. In other words, the receiver cannot properly compute the target function in the network.

For each $e \in C$, there exist $k \times n$ matrices $M(e)_1, \ldots, M(e)_s$ such that the vector carried on e is

$$\sum_{i=1}^{s} \alpha(\sigma_i)\, M(e)_i.$$

For any matrix M, denote its j-th column by $M^{(j)}$. Let w and y be different $k \times s$ matrices over $\mathcal{A}$, whose j-th columns agree for all $j \notin I_C$.

Let us suppose that the vectors carried on the edges of C, when the the column vectors of w are the source messages, are the same as when the the column vectors of y are the source messages. Then, for all $e \in C$,

$$\sum_{i=1}^{s} w^{(i)} M(e)_i = \sum_{i=1}^{s} y^{(i)} M(e)_i. \tag{5.17}$$

We will show that this leads to a contradiction, namely that ρ cannot compute f. Let m be an integer such that if d denotes the m-th row of $w - y$, then $d \neq 0$. For the case

[5] As a reminder, “network” here refers to single-receiver networks in the context of computing.

where $\mathcal{A}$ is a field and f is not reducible, by Lemma 5.4.6(i), there exist $a \in \mathcal{A}$ and $x \in \mathcal{A}^s$ such that $ad \neq 0$ and

$$f(ad + x) \neq f(x). \tag{5.18}$$

In the case where $\mathcal{A}$ is a ring with identity and f is semi-injective, we obtain (5.18) from Lemma 5.4.6(ii) in the special case of $a = 1$.

Let u be any $k \times s$ matrix over $\mathcal{A}$ whose m-th row is x and let $v = u + a(w - y)$. From (5.18), the target function f differs on the m-th rows of u and v. Thus, the vectors on the in-edges of the receiver ρ must differ between two cases: (1) when the sources messages are the columns of u, and (2) when the sources messages are the columns of v. The vector carried by any in-edge of the receiver is a function of each of the message vectors $\alpha(\sigma_j)$, for $j \notin I_C$, and the vectors carried by the edges in the cut C. Furthermore, the j-th columns of u and v agree if $j \notin I_C$. Thus, at least one of the vectors on an edge in C must change when the set of source message vectors changes from u to v. However this is contradicted by the fact that for all $e \in C$, the vector carried on e when the columns of u are the source messages is

$$\begin{aligned}\sum_{i=1}^{s} u^{(i)} M(e)_i &= \sum_{i=1}^{s} u^{(i)} M(e)_i + a \sum_{i=1}^{s} (w^{(i)} - y^{(i)})) M(e)_i \qquad \text{[from (5.17)]} \\ &= \sum_{i=1}^{s} v^{(i)} M(e)_i \qquad (5.19)\end{aligned}$$

which is also the vector carried on e when the columns of v are the source messages.

Hence, for any two different matrices w and y whose j-th columns agree for all $j \notin I_C$, at least one vector carried by an edge in the cut C has to differ in value in the case where the source messages are the columns of w from the case where the source messages are the columns of y. This fact implies that

$$(|\mathcal{A}|^n)^{|C|} \geq (|\mathcal{A}|^k)^{|I_C|}$$

and thus

$$\frac{k}{n} \leq \frac{|C|}{|I_C|}.$$

Since the cut C is arbitrary, we conclude (using (5.4)) that

$$\frac{k}{n} \leq \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{|I_C|} = \mathcal{C}_{\text{rout}}(\mathcal{N}, f)\,.$$

Taking the supremum over all (k, n) linear network codes that compute f in $\mathcal{N}$, we get

$$\mathcal{C}_{\text{lin}}(\mathcal{N}, f) \leq \mathcal{C}_{\text{rout}}(\mathcal{N}, f)\,.$$

■

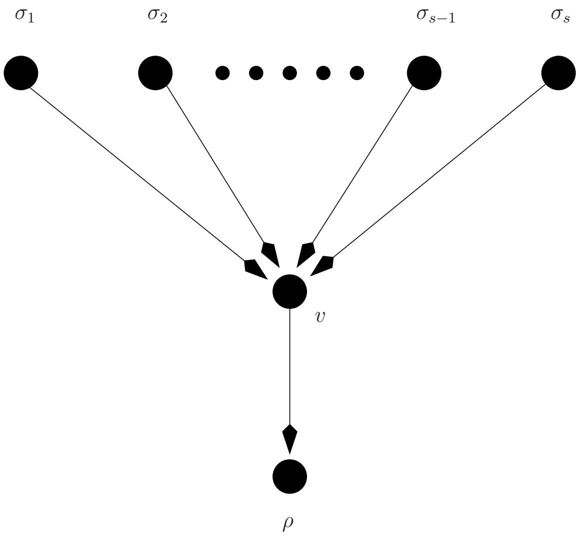


Figure 5.3: Network $\mathcal{N}_{5,s}$ has sources $\sigma_1, \sigma_2, \ldots, \sigma_s$, each connected to the relay v by an edge and v is connected to the receiver by an edge.

Theorem 5.4.8 showed that if a network's target function is not reducible (e.g. semi-injective target functions) then there can be no computing capacity gain of using linear coding over routing. The following theorem shows that if the target function is injective, then there cannot even be any nonlinear computing gain over routing.

Note that if the identity target function is used in Theorem 5.4.9, then the result states that there is no coding gain over routing for ordinary network coding. This is consistent since our stated assumption in Section 5.2 is that only single-receiver networks are considered here (for some networks with two or more receivers, it is well known that linear coding may provide network coding gain over network routing).

Theorem 5.4.9. *If $\mathcal{N}$ is a network with an injective target function f, then*

$$\mathcal{C}_{cod}(\mathcal{N}, f) = \mathcal{C}_{rout}(\mathcal{N}, f) \, .$$

Proof. It follows from [63, Theorem 4.2] that for any single-receiver network $\mathcal{N}$ and the identity target function f, we have $\mathcal{C}_{\text{cod}}(\mathcal{N}, f) = \mathcal{C}_{\text{rout}}(\mathcal{N}, f)$. This can be straightforwardly extended to injective target functions for network computing. ■

Theorem 5.4.8 showed that there cannot be linear computing gain for networks whose target functions are not reducible, and Theorem 5.4.9 showed that the same is true for target functions that are injective. However, Theorem 5.4.11 will show via an example network that nonlinear codes may provide a capacity gain over linear codes if the target function is not injective. This reveals a limitation of linear codes compared to nonlinear ones for non-injective target functions that are not reducible. For simplicity, in Theorem 5.4.11 we only consider the case when there are two or more sources. We need the following lemma first.

Lemma 5.4.10. *The computing capacity of the network $\mathcal{N}_{5,s}$ shown in Figure 5.3, with respect to a target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$, satisfies*

$$\mathcal{C}_{cod}(\mathcal{N}_{5,s}, f) \geq \min \left\{ 1, \ \frac{1}{\log_{|\mathcal{A}|} |f(\mathcal{A}^s)|} \right\} .$$

Proof. Suppose

$$\log_{|\mathcal{A}|} |f(\mathcal{A}^s)| < 1. \tag{5.20}$$

Let $k = n = 1$ and assume that each source node sends its message to node v. Let

$$g \ : \ f(\mathcal{A}^s) \longrightarrow \mathcal{A}$$

be any injective map (which exists by (5.20)). Then the node v can compute g and send it to the receiver. The receiver can compute the value of f from the value of g and thus a rate of 1 is achievable, so $\mathcal{C}_{\text{cod}}(\mathcal{N}_{5,s}, f) \geq 1$.

Now suppose

$$\log_{|\mathcal{A}|} |f(\mathcal{A}^s)| \geq 1. \tag{5.21}$$

Choose integers k and n such that

$$\frac{1}{\log_{|\mathcal{A}|} |f(\mathcal{A}^s)|} - \epsilon \leq \frac{k}{n} \leq \frac{1}{\log_{|\mathcal{A}|} |f(\mathcal{A}^s)|}. \tag{5.22}$$

Now choose an arbitrary injective map (which exists by (5.22))

$$g \,:\, (f(\mathcal{A}^s))^k \longrightarrow \mathcal{A}^n.$$

Since $n \geq k$ (by (5.21) and (5.22)), we can still assume that each source sends its k-length message vector to node v. Node v computes f for each of the k sets of source messages, encodes those values into an n-length vector over $\mathcal{A}$ using the injective map g and transmits it to the receiver. The existence of a decoding function which satisfies (5.2) is then obvious from the fact that g is injective. From (5.22), the above code achieves a computing rate of

$$\frac{k}{n} \geq \frac{1}{\log_{|\mathcal{A}|} |f(\mathcal{A}^s)|} - \epsilon.$$

Since ϵ was arbitrary, it follows that the computing capacity $\mathcal{C}_{\text{cod}}(\mathcal{N}_{5,s}, f)$ is at least $1/\log_{|\mathcal{A}|} |f(\mathcal{A}^s)|$. ■

Theorem 5.4.11. *Let $\mathcal{A}$ be a finite field alphabet. Let $s \geq 2$ and let f be a target function that is neither injective nor reducible. Then there exists a network $\mathcal{N}$ such that*

$$\mathcal{C}_{cod}(\mathcal{N}, f) > \mathcal{C}_{lin}(\mathcal{N}, f)\,.$$

Proof. If $\mathcal{N}$ is the network $\mathcal{N}_{5,s}$ shown in Figure 5.3 with alphabet $\mathcal{A}$, then

$$\begin{aligned}
\mathcal{C}_{\text{lin}}(\mathcal{N}, f) &= 1/s && \text{[from Theorem 5.4.8 and (5.4)]} \\
&< \min\left\{1, \ \frac{1}{\log_{|\mathcal{A}|} |f(\mathcal{A}^s)|}\right\} && \text{[from } s \geq 2 \text{ and } |f(\mathcal{A}^s)| < |\mathcal{A}|^s\text{]} \\
&\leq \mathcal{C}_{\text{cod}}(\mathcal{N}, f) \, . && \text{[from Lemma 5.4.10]}
\end{aligned}$$

■

The same proof of Theorem 5.4.11 shows that it also holds if the alphabet $\mathcal{A}$ is a ring with identity and the target function f is semi-injective but not injective.

5.4.2 Reducible target functions

In Theorem 5.4.12, we prove a converse to Theorem 5.4.8 by showing that if a target function is reducible, then there exists a network in which the linear computing capacity is larger than the routing computing capacity. Theorem 5.4.14 shows that, even if the target function is reducible, linear codes may not achieve the full (nonlinear) computing capacity of a network.

Theorem 5.4.12. *Let $\mathcal{A}$ be a ring. If a target function $f : \mathcal{A}^s \longrightarrow \mathcal{B}$ is reducible, then there exists a network $\mathcal{N}$ such that*

$$\mathcal{C}_{lin}(\mathcal{N}, f) > \mathcal{C}_{rout}(\mathcal{N}, f) \, .$$

Proof. Since f is reducible, there exist $\lambda < s$, a matrix $T \in \mathcal{A}^{s\times\lambda}$, and a map $g : \mathcal{A}^\lambda \longrightarrow f(\mathcal{A}^s)$ such that

$$g(xT) = f(x) \ \text{ for every } x \in \mathcal{A}^s. \qquad \text{[from Definition 5.2.1]} \qquad (5.23)$$

Let $\mathcal{N}$ denote the network $\mathcal{N}_{5,s}$ with alphabet $\mathcal{A}$ and target function f. Let $k = 1$, $n = \lambda$ and let the decoding function be $\psi = g$. Since $n \geq 1$, we assume that all the source nodes transmit their messages to node v. For each source vector

$$x = (\alpha(\sigma_1)\,, \alpha(\sigma_2)\,, \ldots, \alpha(\sigma_s))$$

node v computes xT and sends it to the receiver. Having received the n-dimensional vector xT, the receiver computes

$$\begin{aligned} \psi(xT) &= g(xT) && [\text{from } \psi = g] \\ &= f(x). && [\text{from (5.23)}] \end{aligned}$$

Thus there exists a linear code that computes f in $\mathcal{N}$ with an achievable computing rate of

$$\begin{aligned} \frac{k}{n} &= \frac{1}{\lambda} \\ &> 1/s && [\text{from } \lambda \le s-1] \\ &= \mathcal{C}_{\text{rout}}(\mathcal{N}) && [\text{from (5.4)}] \end{aligned}$$

which is sufficient to establish the claim. ■

For target functions that are not reducible, any improvement on achievable rate of computing using coding must be provided by nonlinear codes (by Theorem 5.4.8). However, within the class of reducible target functions, it turns out that there are target functions for which linear codes are optimal (i.e., capacity achieving) as shown in Theorem 5.5.7, while for certain other reducible target functions, nonlinear codes might provide a strictly larger achievable computing rate compared to linear codes.

Remark 5.4.13. It is possible for a network $\mathcal{N}$ to have a reducible target function f but satisfy $\mathcal{C}_{\text{lin}}(\mathcal{N}, f) = \mathcal{C}_{\text{rout}}(\mathcal{N}, f)$ since the network topology may not allow coding to exploit the structure of the target function to obtain a capacity gain. For example, the 3-node network in Figure 5.4 with $f(x_1, x_2) = x_1 + x_2$ and finite field alphabet $\mathcal{A}$ has

$$\mathcal{C}_{\text{lin}}(\mathcal{N}, f) = \mathcal{C}_{\text{rout}}(\mathcal{N}, f) = 1.$$

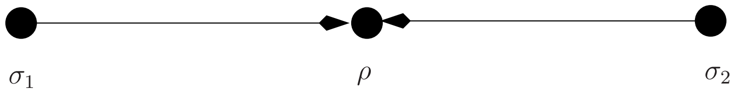


Figure 5.4: A network where there is no benefit to using linear coding over routing for computing f.

Theorem 5.4.11 demonstrates that for every non-injective, non-reducible target function, some network has a nonlinear computing gain over linear coding, and Theorem 5.4.12 shows that for every reducible (hence non-injective) target function, some network has a linear computing gain over routing. The following theorem shows that for some reducible target function, some network has both of these linear and nonlinear computing gains.

Theorem 5.4.14. *There exists a network $\mathcal{N}$ and a reducible target function f such that:*

$$\mathcal{C}_{cod}(\mathcal{N}, f) > \mathcal{C}_{lin}(\mathcal{N}, f) > \mathcal{C}_{rout}(\mathcal{N}, f) .$$

Proof. Let $\mathcal{N}$ denote the network $\mathcal{N}_{5,3}$ shown in Figure 5.3 with $s = 3$, alphabet $\mathcal{A} = \mathbb{F}_2$, and let f be the target function in Example 5.2.2. The routing capacity is given by

$$\mathcal{C}_{\text{rout}}(\mathcal{N}, f) = 1/3. \qquad \text{[from (5.4)]} \tag{5.24}$$

Let $k = n = 1$. Assume that the sources send their respective messages to node v. The target function f can then be computed at v and sent to the receiver. Hence, $k/n = 1$ is an achievable computing rate and thus

$$\mathcal{C}_{\text{cod}}(\mathcal{N}, f) \geq 1. \tag{5.25}$$

Now consider any (k, n) linear code that computes f in $\mathcal{N}$. Such a linear code immediately implies a (k, n) linear code that computes the target function $g(x_1, x_2) = x_1x_2$ in network $\mathcal{N}_{5,2}$ as follows. From the (k, n) linear code that computes f in $\mathcal{N}$, we get a

$3k \times n$ matrix M such that the node v in network $\mathcal{N}$ computes

$$\begin{pmatrix} \alpha(\sigma_1) & \alpha(\sigma_2) & \alpha(\sigma_3) \end{pmatrix} M$$

and the decoding function computes f from the resulting vector. Now, in $\mathcal{N}_{5,2}$, we let the node v compute

$$\begin{pmatrix} \alpha(\sigma_1) & 0 & \alpha(\sigma_2) \end{pmatrix} M$$

and send it to the receiver. The receiver can compute the function g from the received n-dimensional vector using the relation $g(x_1, x_2) = f(x_1, 0, x_2)$. Using the fact that the function g is not reducible (in fact, it is semi-injective),

$$\begin{aligned} \frac{k}{n} &\leq \mathcal{C}_{\text{lin}}(\mathcal{N}_{5,2}, g) \\ &= \mathcal{C}_{\text{rout}}(\mathcal{N}_{5,2}, g) && \text{[from Theorem 5.4.8]} \\ &= 1/2. && \text{[from (5.4)]} \end{aligned}$$

Consequently,

$$\mathcal{C}_{\text{lin}}(\mathcal{N}, f) \leq 1/2. \tag{5.26}$$

Now we will construct a $(1, 2)$ linear code that computes f in $\mathcal{N}$. Let $k = 1$, $n = 2$ and

$$M = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Let the sources send their respective messages to v while v computes

$$\begin{pmatrix} \alpha(\sigma_1) & \alpha(\sigma_2) & \alpha(\sigma_3) \end{pmatrix} M$$

and transmits the result to the receiver from which f is computable. Since the above code achieves a computing rate of $1/2$, combined with (5.26), we get

$$\mathcal{C}_{\text{lin}}(\mathcal{N}, f) = 1/2. \tag{5.27}$$

The claim of the theorem now follows from (5.24), (5.25), and (5.27). ■

5.5 Computing linear target functions

We have previously shown that for reducible target functions there may be a computing capacity gain for using linear codes over routing. In this section, we show that for a special subclass of reducible target functions, namely linear target functions[6] over finite fields, linear network codes achieve the full (nonlinear) computing capacity. We now describe a special class of linear codes over finite fields that suffice for computing linear target functions over finite fields at the maximum possible rate.

Throughout this section, let $\mathcal{N}$ be a network and let k, n, and c be positive integers such that $k/n = c$. Each k symbol message vector generated by a source $\sigma \in S$ can be viewed as a c-dimensional vector

$$\alpha(\sigma) = \left(\alpha(\sigma)_1, \alpha(\sigma)_2, \ldots, \alpha(\sigma)_c\right) \in \mathbb{F}_{q^k}$$

where $\alpha(\sigma)_i \in \mathbb{F}_{q^n}$ for each i. Likewise, the decoder ψ generates a vector of k symbols from $\mathbb{F}_q$, which can be viewed as a c-dimensional vector of symbols from $\mathbb{F}_{q^n}$. For each $e \in \mathcal{E}$, the edge vector z_e is viewed as an element of $\mathbb{F}_{q^n}$.

For every node $u \in \mathcal{V} - \rho$, and every out-edge $e \in \mathcal{E}_o(u)$, we choose an encoding function $h^{(e)}$ whose output is:

$$\begin{cases} \displaystyle\sum_{\hat{e} \in \mathcal{E}_i(u)} \gamma_{\hat{e}}^{(e)} z_{\hat{e}} + \sum_{j=1}^{c} \beta_j^{(e)} \alpha(u)_j & \text{if } u \in S \\ \displaystyle\sum_{\hat{e} \in \mathcal{E}_i(u)} \gamma_{\hat{e}}^{(e)} z_{\hat{e}} & \text{otherwise} \end{cases} \tag{5.28}$$

for some $\gamma_{\hat{e}}^{(e)}, \beta_j^{(e)} \in \mathbb{F}_{q^n}$ and we use a decoding function ψ whose j-th component output ψ_j is:

$$\sum_{e \in \mathcal{E}_i(\rho)} \delta_j^{(e)} z_e \quad \text{for all } j \in \{1, 2, \ldots, c\} \tag{5.29}$$

for certain $\delta_j^{(e)} \in \mathbb{F}_{q^n}$. Here we view each $h^{(e)}$ as a function of the in-edges to e and

[6] The definition of “linear target function” was given in Table 5.2.

the source messages generated by u and we view ψ as a function of the inputs to the receiver. The chosen encoder and decoder are seen to be linear.

Let us denote the edges in $\mathcal{E}$ by $e_1, e_2, \ldots, e_{|\mathcal{E}|}$. For each source σ and each edge $e_j \in \mathcal{E}_o(\sigma)$, let $x_1^{(e_j)}, \ldots, x_c^{(e_j)}$ be variables, and for each $e_j \in \mathcal{E}_i(\rho)$, let $w_1^{(e_j)}, \ldots, w_c^{(e_j)}$ be variables. For every $e_i, e_j \in \mathcal{E}$ such that *head*(e_i) $=$ *tail*(e_j), let $y_{e_i}^{(e_j)}$ be a variable. Let x, y, w be vectors containing all the variables $x_i^{(e_j)}$, $y_{e_i}^{(e_j)}$, and $w_i^{(e_j)}$, respectively. We will use the short hand notation $\mathbb{F}[y]$ to mean the ring of polynomials $\mathbb{F}[\cdots, y_{e_i}^{(e_j)}, \cdots]$ and similarly for $\mathbb{F}[x, y, w]$.

Next, we define matrices $A_\tau(x)$, $F(y)$, and $B(w)$.

(i) For each $\tau \in \{1, 2, \cdots, s\}$, let $A_\tau(x)$ be a $c \times |\mathcal{E}|$ matrix $A_\tau(x)$, given by

$$(A_\tau(x))_{i,j} = \begin{cases} x_i^{(e_j)} & \text{if } e_j \in \mathcal{E}_o(\sigma_\tau) \\ 0 & \text{otherwise} \end{cases} \tag{5.30}$$

(ii) Let $F(y)$ be a $|\mathcal{E}| \times |\mathcal{E}|$ matrix, given by

$$(F(y))_{i,j} = \begin{cases} y_{e_i}^{(e_j)} & \text{if } e_i, e_j \in \mathcal{E} \text{ and } \textit{head}(e_i) = \textit{tail}(e_j) \\ 0 & \text{otherwise} \end{cases} \tag{5.31}$$

(iii) Let $B(w)$ be a $c \times |\mathcal{E}|$ matrix, given by

$$(B(w))_{i,j} = \begin{cases} w_i^{(e_j)} & \text{if } e_j \in \mathcal{E}_i(\rho) \\ 0 & \text{otherwise.} \end{cases} \tag{5.32}$$

Consider an (nc, n) linear code of the form in (5.28)–(5.29).

Since the graph G associated with the network is acyclic, we can assume that the edges $e_1, e_2, \ldots$ are ordered such that the matrix F is strictly upper-triangular, and thus we can apply Lemma 5.5.1. Let I denote the identity matrix of suitable dimension.

Lemma 5.5.1. *(Koetter-Médard [73, Lemma 2]) The matrix $I - F(y)$ is invertible over the ring $\mathbb{F}_q[y]$.*

Lemma 5.5.2. *(Koetter-Médard [73, Theorem 3]) For $s = 1$ and for all $\tau \in \{1, \ldots, s\}$, the decoder in* (5.29) *satisfies*

$$\psi = \alpha(\sigma_1)\, A_\tau(\beta)(I - F(\gamma))^{-1}B(\delta)^t.$$

Lemma 5.5.3. *(Alon [80, Theorem 1.2]) Let $\mathbb{F}$ be an arbitrary field, and let $g = g(x_1, \ldots, x_m)$ be a polynomial in $\mathbb{F}[x_1, \ldots, x_m]$. Suppose the degree $deg(g)$ of g is $\sum_{i=1}^m t_i$, where each t_i is a nonnegative integer, and suppose the coefficient of $\prod_{i=1}^m x_i^{t_i}$ in g is nonzero. Then, if $S_1, \ldots, S_m$ are subsets of $\mathbb{F}$ with $|S_i| > t_i$, there are $s_1 \in S_1$, $s_2 \in S_2, \ldots, s_m \in S_m$ so that*

$$g(s_1, \ldots, s_m) \neq 0.$$

For each $\tau \in \{1, 2, \ldots, s\}$, define the $c \times c$ matrix

$$M_\tau(x, y, w) = A_\tau(x)(I - F(y))^{-1}B(w)^t \tag{5.33}$$

where the components of $M_\tau(x, y, w)$ are viewed as lying in $\mathbb{F}_q[x, y, w]$.

Lemma 5.5.4. *If for all $\tau \in \{1, 2, \ldots, s\}$,*

$$\det\left(M_\tau(x, y, w)\right) \neq 0$$

in the ring $\mathbb{F}_q[x, y, w]$, then there exists an integer $n > 0$ and vectors β, γ, δ over $\mathbb{F}_{q^n}$ such that for all $\tau \in \{1, 2, \ldots, s\}$ the matrix $M_\tau(\beta, \gamma, \delta)$ is invertible in the ring of $c \times c$ matrices with components in $\mathbb{F}_{q^n}$.

Proof. The quantity

$$\det\left(\prod_{\tau=1}^{s} M_\tau(x, y, w)\right)$$

is a nonzero polynomial in $\mathbb{F}_q[x, y, w]$ and therefore also in $\mathbb{F}_{q^n}[x, y, w]$ for any $n \geq 1$. Therefore, we can choose n large enough such that the degree of this polynomial is less than q^n. For such an n, Lemma 5.5.3 implies there exist vectors β, γ, δ (whose components correspond to the components of the vector variables x, y, w) over $\mathbb{F}_{q^n}$ such that

$$\det\left(\prod_{\tau=1}^{s} M_\tau(\beta,\gamma,\delta)\right) \neq 0. \tag{5.34}$$

and therefore, for all $\tau \in \{1, 2, \ldots, s\}$

$$\det\left(M_\tau(\beta,\gamma,\delta)\right) \neq 0.$$

Thus, each $M_\tau(\beta,\gamma,\delta)$ is invertible.

■

The following lemma improves upon the upper bound of Lemma 5.2.11 in the special case where the target function is linear over a finite field.

Lemma 5.5.5. *If $\mathcal{N}$ is network with a linear target function f over a finite field, then*

$$\mathcal{C}_{\textit{cod}}(\mathcal{N}, f) \leq \min_{C\in\Lambda(\mathcal{N})} |C|\,.$$

Proof. The same argument is used as in the proof of Lemma 5.2.11, except instead of using $R_{I_C,f} \geq 2$, we use the fact that $R_{I_C,f} = |\mathcal{A}|$ for linear target functions. ■

Theorem 5.5.6. *If $\mathcal{N}$ is a network with a linear target function f over finite field $\mathbb{F}_q$, then*

$$\mathcal{C}_{\textit{lin}}(\mathcal{N}, f) = \min_{C\in\Lambda(\mathcal{N})} |C|\,.$$

Proof. We have

$$\begin{aligned} \mathcal{C}_{\text{lin}}(\mathcal{N}, f) &\leq \mathcal{C}_{\text{cod}}(\mathcal{N}, f) \\ &\leq \min_{C\in\Lambda(\mathcal{N})} |C|\,. && \text{[from Lemma 5.5.5]} \end{aligned}$$

For a lower bound, we will show that there exists an integer n and an (nc, n) linear code that computes f with a computing rate of $c = \min\limits_{C\in\Lambda(\mathcal{N})} |C|$.

From Lemma 5.5.1, the matrix $I - F(y)$ in invertible over the ring $\mathbb{F}_q[x, y, w]$ and therefore also over $\mathbb{F}_{q^n}[x, y, w]$. Since any minimum cut between the source σ_τ and the receiver ρ has at least c edges, it follows from [73, Theorem 2][7] that $\det\left(M_\tau(x,y,w)\right) \neq 0$ for every $\tau \in \{1,2,\ldots,s\}$. From Lemma 5.5.4, there exists an integer $n > 0$ and vectors β,γ,δ over $\mathbb{F}_{q^n}$ such that $M_\tau(\beta,\gamma,\delta)$ is invertible for every $\tau \in \{1,2,\ldots,s\}$. Since f is linear, we can write

$$f(u_1,\ldots,u_s) = a_1u_1 + \cdots + a_su_s.$$

For each $\tau \in \{1,2,\ldots,s\}$, let

$$\hat{A}_\tau(\beta) = a_\tau \left(M_\tau(\beta,\gamma,\delta)\right)^{-1} A_\tau(\beta). \tag{5.35}$$

If a linear code corresponding to the matrices $\hat{A}_\tau(\beta), B(\delta)$, and $F(\gamma)$ is used in network $\mathcal{N}$, then the c-dimensional vector over $\mathbb{F}_{q^n}$ computed by the receiver ρ is

$$\begin{aligned}
\psi &= \sum_{\tau=1}^{s} \alpha(\sigma_\tau)\,\hat{A}_\tau(\beta)(I-F(\gamma))^{-1}B(\delta)^t && \text{[from Lemma 5.5.2]}\\
&= \sum_{\tau=1}^{s} \alpha(\sigma_\tau)\,a_\tau\left(M_\tau(\beta,\gamma,\delta)\right)^{-1}A_\tau(\beta)(I-F(\gamma))^{-1}B(\delta)^t && \text{[from (5.35)]}\\
&= \sum_{\tau=1}^{s} a_\tau\,\alpha(\sigma_\tau) && \text{[from (5.33)]}\\
&= \left(f\left(\alpha(\sigma_1)_1,\ldots,\alpha(\sigma_s)_1\right),\ldots,f\left(\alpha(\sigma_1)_c,\ldots,\alpha(\sigma_s)_c\right)\right)
\end{aligned}$$

which proves that the linear code achieves a computing rate of c. ■

Theorem 5.5.7 below proves the optimality of linear codes for computing linear target functions in a single-receiver network. It also shows that the computing capacity of a network for a given target function cannot be larger than the number of network sources times the routing computing capacity for the same target function. This bound tightens the general bound given in Theorem 5.2.12 for the special case of linear target functions over finite fields. Theorem 5.5.8 shows that this upper bound can be tight.

Theorem 5.5.7. *If $\mathcal{N}$ is network with s sources and linear target function f over finite*

[7] Using the implication $(1) \Longrightarrow (3)$ in [73, Theorem 2].

field $\mathbb{F}_q$, then

$$\mathcal{C}_{\mathit{lin}}(\mathcal{N}, f) = \mathcal{C}_{\mathit{cod}}(\mathcal{N}, f) \leq s\, \mathcal{C}_{\mathit{rout}}(\mathcal{N}, f)\,.$$

Proof.

$$\begin{aligned} s\, \mathcal{C}_{\text{rout}}(\mathcal{N}, f) &\geq \min_{C \in \Lambda(\mathcal{N})} |C| && \text{[from (5.4) and Theorem 5.2.10]} \\ &\geq \mathcal{C}_{\text{cod}}(\mathcal{N}, f) && \text{[from Lemma 5.5.5]} \\ &\geq \mathcal{C}_{\text{lin}}(\mathcal{N}, f) \\ &= \min_{C \in \Lambda(\mathcal{N})} |C|\,. && \text{[from Theorem 5.5.6]} \end{aligned}$$

■

We note that the inequality in Theorem 5.5.7 can be shown to apply to certain target functions other than linear functions over finite fields, such as the minimum, maximum, and arithmetic sum target functions.

Theorem 5.5.8. *For every s, if a target function $f : \mathcal{A}^s \longrightarrow \mathcal{A}$ is linear over finite field $\mathbb{F}_q$, then there exists a network $\mathcal{N}$ with s sources, such that*

$$\mathcal{C}_{\mathit{lin}}(\mathcal{N}, f) = s\, \mathcal{C}_{\mathit{rout}}(\mathcal{N}, f)\,.$$

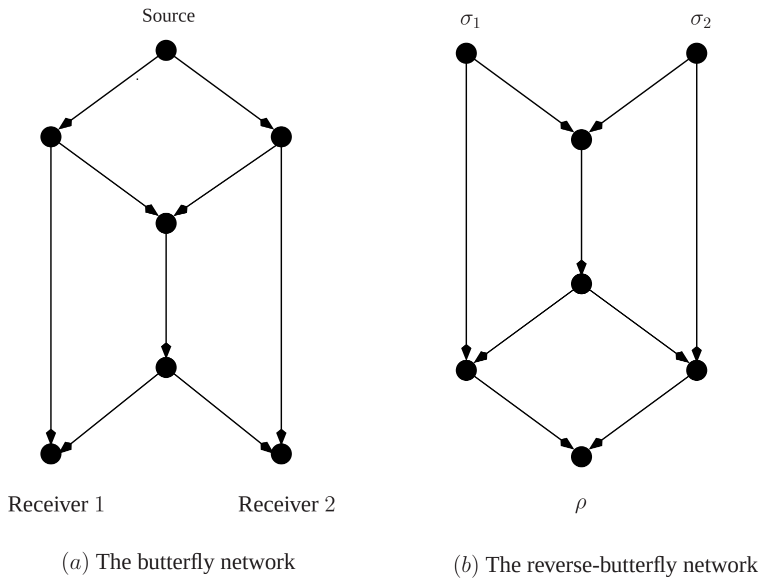
Proof. Let $\mathcal{N}$ denote the network $\mathcal{N}_{5,s}$ shown in Figure 5.3. Then

$$\begin{aligned} \mathcal{C}_{\text{lin}}(\mathcal{N}, f) &= 1 && \text{[from Theorem 5.5.6]} \\ \mathcal{C}_{\text{rout}}(\mathcal{N}, f) &= \mathcal{C}_{\text{rout}}(\mathcal{N}) && \text{[from Theorem 5.2.10]} \\ &= 1/s. && \text{[from (5.4)]} \end{aligned}$$

■

5.6 The reverse butterfly network

In this section we study an example network which illustrates various concepts discussed previously in this chapter and also provides some interesting additional results for network computing.



(a) The butterfly network

(b) The reverse-butterfly network

Figure 5.5: The butterfly network and its reverse $\mathcal{N}_6$.

The network $\mathcal{N}_6$ shown in Figure 5.5(b) is called the *reverse butterfly network*. It has $S = \{\sigma_1, \sigma_2\}$, receiver node ρ, and is obtained by reversing the direction of all the edges of the multicast butterfly network shown in Figure 5.5(a).

Theorem 5.6.1. *The routing and linear computing capacities of the reverse butterfly network $\mathcal{N}_6$ with alphabet $\mathcal{A} = \{0, 1, \ldots, q-1\}$ and arithmetic sum target function $f : \mathcal{A}^2 \longrightarrow \{0, 1, \ldots, 2(q-1)\}$ are*

$$\mathcal{C}_{\textit{rout}}(\mathcal{N}_6, f) = \mathcal{C}_{\textit{lin}}(\mathcal{N}_6, f) = 1.$$

Proof. We have

$$\begin{aligned} \mathcal{C}_{\text{lin}}(\mathcal{N}_6, f) &= \mathcal{C}_{\text{rout}}(\mathcal{N}_6) && \text{[from Theorem 5.4.8]} \\ &= 1. && \text{[from (5.4)]} \end{aligned}$$

■

Remark 5.6.2. The arithmetic sum target function can be computed in the reverse butterfly network at a computing rate of 1 using only routing (by sending σ_1 down the left side and σ_2 down the right side of the graph). Combined with Theorem 5.6.1, it follows that the routing computing capacity is equal to 1 for all $q \geq 2$.

Theorem 5.6.3. *The computing capacity of the reverse butterfly network $\mathcal{N}_6$ with alphabet $\mathcal{A} = \{0, 1, \ldots, q-1\}$ and arithmetic sum target function $f : \mathcal{A}^2 \longrightarrow \{0, 1, \ldots, 2(q-1)\}$ is*

$$\mathcal{C}_{cod}(\mathcal{N}_6, f) = \frac{2}{\log_q (2q-1)}.$$

Remark 5.6.4. The computing capacity $\mathcal{C}_{\text{cod}}(\mathcal{N}_6, f)$ obtained in Theorem 5.6.3 is a function of the coding alphabet $\mathcal{A}$ (i.e. the domain of the target function f). In contrast, for ordinary network coding (i.e. when the target function is the identity map), the coding capacity and routing capacity are known to be independent of the coding alphabet used [59]. For the reverse butterfly network, if, for example, $q = 2$, then $\mathcal{C}_{\text{cod}}(\mathcal{N}_6, f)$ is approximately equal to 1.26 and increases asymptotically to 2 as $q \to \infty$.

Remark 5.6.5. The ratio of the coding capacity to the routing capacity for the multicast butterfly network with two messages was computed in [59] to be $4/3$ (i.e. coding provides a gain of about 33%). The corresponding ratio for the reverse butterfly network increases as a function of q from approximately 1.26 (i.e. 26%) when $q = 2$ to 2 (i.e. 100%) when $q = \infty$. Furthermore, in contrast to the multicast butterfly network, where the coding capacity is equal to the linear coding capacity, in the reverse butterfly network the computing capacity is strictly greater than the linear computing capacity.

Remark 5.6.6. Recall that capacity is defined as the supremum of a set of rational numbers k/n such that a (k, n) code that computes a target function exists. It was pointed out in [59] that it remains an open question whether the coding capacity of a network can be irrational. Our Theorem 5.6.3 demonstrates that the computing capacity of a network (e.g. the reverse butterfly network) with unit capacity links can be irrational when the target function to be computed is the arithmetic sum target function of the source messages.

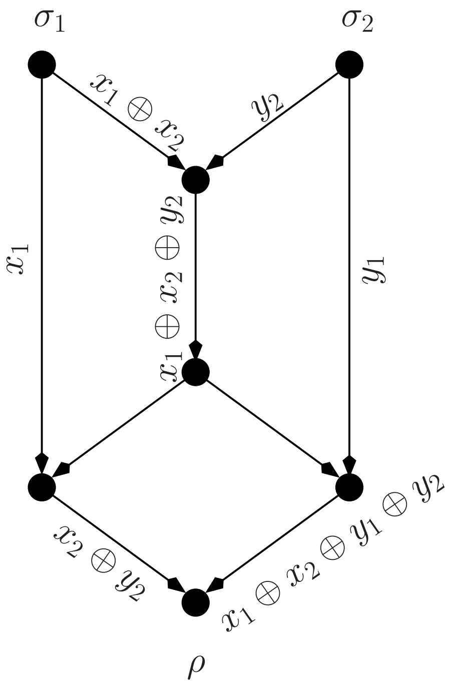


Figure 5.6: The reverse butterfly network with a code that computes the mod q sum target function.

The following lemma is used to prove Theorem 5.6.3.

Lemma 5.6.7. *The computing capacity of the reverse butterfly network* $\mathcal{N}_6$ *with* $\mathcal{A} = \{0, 1, \ldots, q-1\}$ *and the mod* q *sum target function* f *is*

$$\mathcal{C}_{cod}(\mathcal{N}_6, f) = 2.$$

Proof. The upper bound of 2 on $\mathcal{C}_{\text{cod}}(\mathcal{N}_6, f)$ follows from [74, Theorem II.1]. To establish the achievability part, let $k = 2$ and $n = 1$. Consider the code shown in Figure 5.6, where '$\oplus$' indicates the $\bmod\ q$ sum. The receiver node ρ gets $\alpha(\sigma_1)_1 \oplus \alpha(\sigma_2)_1$ and $\alpha(\sigma_1)_1 \oplus \alpha(\sigma_2)_1 \oplus \alpha(\sigma_1)_2 \oplus \alpha(\sigma_2)_2$ on its in-edges, from which it can compute $\alpha(\sigma_1)_2 \oplus \alpha(\sigma_2)_2$. This code achieves a rate of 2. ■

Proof of Theorem 5.6.3: We have

$$\mathcal{C}_{\text{cod}}(\mathcal{N}_6, f) \leq 2/\log_q(2q-1). \qquad \text{[from [74, Theorem II.1]]}$$

To establish the lower bound, we use the fact the that arithmetic sum of two elements from $\mathcal{A} = \{0, 1, \ldots, q-1\}$ is equal to their $\bmod 2q-1$ sum. Let the reverse butterfly network have alphabet $\hat{\mathcal{A}} = \{0, 1, \ldots, 2(q-1)\}$. From Lemma 5.6.7 (with alphabet $\hat{\mathcal{A}}$), the $\bmod 2q-1$ sum target function can be computed in $\mathcal{N}_6$ at rate 2. Indeed for every $n \geq 1$, there exists a $(2n, n)$ network code that computes the mod $2q-1$ sum target function at rate 2. So for the remainder of this proof, let $k = 2n$. Furthermore, every such code using $\hat{\mathcal{A}}$ can be "simulated" using $\mathcal{A}$ by a corresponding $(2n, \lceil n \log_q (2q-1) \rceil)$ code for computing the mod $2q-1$ sum target function, as follows. Let n' be the smallest integer such that $q^{n'} \geq (2q-1)^n$, i.e., $n' = \lceil n \log_q (2q-1) \rceil$. Let $g : \hat{\mathcal{A}}^n \to \mathcal{A}^{n'}$ be an injection (which exists since $q^{n'} \geq (2q-1)^n$) and let the function g^{-1} denote the inverse of g on it's image $g(\hat{\mathcal{A}})$. Let $x^{(1)}, x^{(2)}$ denote the first and last, respectively, halves of the message vector $\alpha(\sigma_1) \in \mathcal{A}^{2n}$, where we view $x^{(1)}$ and $x^{(2)}$ as lying in $\hat{\mathcal{A}}^n$ (since $\mathcal{A} \subseteq \hat{\mathcal{A}}$). The corresponding vectors $y^{(1)}, y^{(2)}$ for the source σ_2 are similarly defined.

Figure 5.7 illustrates a $(2n, n')$ code for network $\mathcal{N}_6$ using alphabet $\mathcal{A}$ where '$\oplus$' denotes the $\bmod 2q-1$ sum. Each of the nodes in $\mathcal{N}_6$ converts each of the received vectors over $\mathcal{A}$ into a vector over $\hat{\mathcal{A}}$ using the function g^{-1}, then performs coding in Figure 5.6 over $\hat{\mathcal{A}}$, and finally converts the result back to $\mathcal{A}$. Similarly, the receiver node T computes the component-wise arithmetic sum of the source message vectors $\alpha(\sigma_1)$ and $\alpha(\sigma_2)$ using

$$\begin{aligned}
&\alpha(\sigma_1) + \alpha(\sigma_2) \\
&= \big(g^{-1}(g(x^{(1)} \oplus x^{(2)} \oplus y^{(1)} \oplus y^{(2)})) \ominus g^{-1}(g(x^{(2)} \oplus y^{(2)})), \\
&\qquad g^{-1}(g(x^{(2)} \oplus y^{(2)}))\big) \\
&= (x^{(1)} \oplus y^{(1)}, x^{(2)} \oplus y^{(2)}).
\end{aligned}$$

For any $n \geq 1$, the above code computes the arithmetic sum target function in $\mathcal{N}$ at a rate of

$$\frac{k}{n'} = \frac{2n}{\lceil n \log_q (2q-1) \rceil}.$$

Thus for any $\epsilon > 0$, by choosing n large enough we obtain a code that computes the

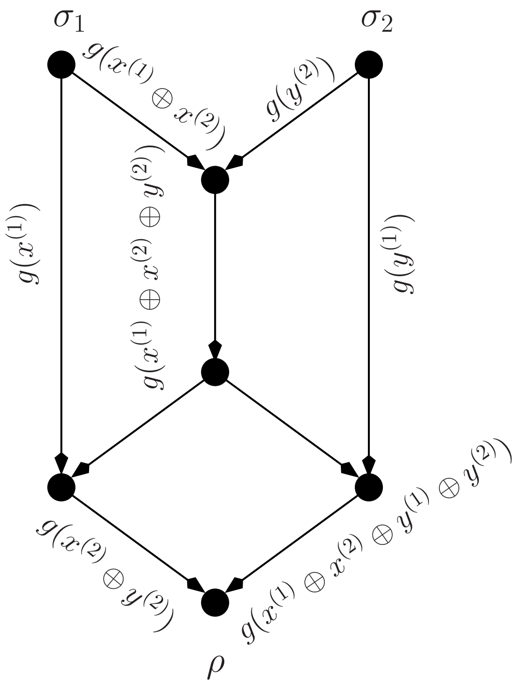

σ_1
σ_2
$g(x^{(1)} \oplus x^{(2)})$
$g(y^{(2)})$
$g(x^{(1)})$
$g(x^{(1)} \oplus x^{(2)} \oplus y^{(2)})$
$g(y^{(1)})$
$g(x^{(2)} \oplus y^{(2)})$
$g(x^{(1)} \oplus x^{(2)} \oplus y^{(1)} \oplus y^{(2)})$
ρ

Figure 5.7: The reverse butterfly network with a code that computes the arithmetic sum target function. '$\oplus$' denotes $\bmod 2q - 1$ addition.

arithmetic sum target function, and which achieves a computing rate of at least

$$\frac{2}{\log_q (2q-1)} - \epsilon.$$

■

Chapter 5, in part, has been submitted for publication of the material. The dissertation author was a primary investigator and author of this paper.

Bibliography

[1] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[2] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[3] A. Giridhar and P. R. Kumar. Computing and communicating functions over sensor networks. *IEEE Journal on Selected Areas in Communication*, 23(4):755–764, April 2005.

[4] S. Subramanian, P. Gupta, and S. Shakkottai. Scaling bounds for function computation over large networks. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 136–140, 2007.

[5] N. Khude, A. Kumar, and A. Karnik. Time and energy complexity of distributed computation in wireless sensor networks. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 2625–2637, 2005.

[6] A. El Gamal. Reliable communication of highly distributed information. In T. M. Cover and B. Gopinath, editors, *Open Problems in Communication and Computation*, pages 60–62. Springer-Verlag, 1987.

[7] R. G. Gallager. Finding parity in a simple broadcast network. *IEEE Transactions on Information Theory*, 34(2):176–180, March 1988.

[8] N. Goyal, G. Kindler, and M. Saks. Lower bounds for the noisy broadcast problem. *SIAM Journal on Computing*, 37(6):1806–1841, March 2008.

[9] C. Li, H. Dai, and H. Li. Finding the k largest metrics in a noisy broadcast network. In *Proceedings of the Annual Allerton Conference on Communication, Control, and Computing*, pages 1184–1190, 2008.

[10] L. Ying, R. Srikant, and G. E. Dullerud. Distributed symmetric function computation in noisy wireless sensor networks. *IEEE Transactions on Information Theory*, 53(12):4826–4833, December 2007.

[11] Y. Kanoria and D. Manjunath. On distributed computation in noisy random planar networks. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 626–630, 2007.

[12] C. Dutta, Y. Kanoria, D. Manjunath, and J. Radhakrishnan. A tight lower bound for parity in noisy communication networks. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete Algorithms*, pages 1056–1065, 2008.

[13] C. Li and H. Dai. Towards efficient designs for in-network computing with noisy wireless channels. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–8, 2010.

[14] M. Franceschetti and R. Meester. Random networks for communication. *Cambridge University press*, 2007.

[15] R. G. Gallager. *Information Theory and Reliable Communication*. John Wiley and Sons, New York, 1968.

[16] S. Rajagopalan and L. J. Schulman. A coding theorem for distributed computation. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 790–799, 1994.

[17] A. Giridhar and P. R. Kumar. Toward a theory of in-network computation in wireless sensor networks. *IEEE Communications Magazine*, 44(4):98–107, April 2006.

[18] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger. Network coding for computing: Cut-set bounds. *IEEE Transactions on Information Theory*, 57(2):1015–1030, February 2011.

[19] Jeongyeup Paek, Ben Greenstein, Omprakash Gnawali, Ki-Young Jang, August Joki, Marcos Vieira, John Hicks, Deborah Estrin, Ramesh Govindan, and Eddie Kohler. The tenet architecture for tiered sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 6:34:1–34:44, July 2010.

[20] O. Gnawali, K. Jang, J. Paek, Marcos Vieira, Ramesh Govindan, Ben Greenstein, August Joki, Deborah Estrin, and Eddie Kohler. The tenet architecture for tiered sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 153–166. ACM, Oct 2006.

[21] T. Ho, M. Medard, R. Koetter, D. R Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, October 2006.

[22] R. Koetter and F. R. Kschischang. Coding for errors and erasures in random network coding. *IEEE Transactions on Information Theory*, 54(8):3579–3591, Aug 2008.

[23] M. Jafari Siavoshani, C. Fragouli, and S. Diggavi. Noncoherent multisource network coding. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, pages 817–821. IEEE, Jul 2008.

[24] C. Fragouli, M. Jafari Siavoshani, S. Mohajer, and S. Diggavi. On the capacity of non-coherent network coding. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, pages 273–277. IEEE, Jun 2009.

[25] L. Keller, N. Karamchandani, and C. Fragouli. Function computation over linear channels. In *Proceedings of the IEEE International Symposium on Network Coding (NetCod)*. IEEE, 2010.

[26] D. Silva, F. R. Kschischang, and R. Koetter. A rank-metric approach to error control in random network coding. *IEEE Transactions on Information Theory*, 54(9):3951–3967, Sep 2008.

[27] H. Witsenhausen. The zero-error side information problem and chromatic numbers. *IEEE Transactions on Information Theory*, 22(5):592–593, September 1976.

[28] V. Doshi, D. Shah, M. Medard, and S. Jaggi. Graph coloring and conditional graph entropy. In *Proceedings of the Fortieth Asilomar Conference on Signals, Systems and Computers*, pages 2137–2141, 2006.

[29] A. Orlitsky and J. R. Roche. Coding for computing. *IEEE Transactions on Information Theory*, 47(3):903–917, March 2001.

[30] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Mathematical Library, 1977.

[31] L. Keller, M.J. Siavoshani, C. Fragouli, K. Argyraki, and S. Diggavi. Identity aware sensor networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*. IEEE, April 2009.

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

[33] R. W. Yeung. *A First Course in Information theory*. Springer, 2002.

[34] N. J. A. Harvey, R. Kleinberg, and A. R. Lehman. On the capacity of information networks. *IEEE Transactions on Information Theory & IEEE/ACM Transactions on Networking (joint issue)*, 52(6):2345–2364, June 2006.

[35] C. K. Ngai and R. W. Yeung. Network coding gain of combination networks. In *Proceedings of the IEEE Information Theory Workshop*, pages 283–287, 2004.

[36] J. Körner and K. Marton. How to encode the modulo-two sum of binary sources. *IEEE Transactions on Information Theory*, 25(2):29–221, March 1979.

[37] V. Doshi, D. Shah, M. Medard, and S. Jaggi. Distributed functional compression through graph coloring. In *Proceedings of the Data Compression Conference*, pages 93–102, 2007.

[38] N. Ma and P. Ishwar. Two-terminal distributed source coding with alternating messages for function computation. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 51–55, 2008.

[39] P. Cuff, H. Su, and A. El Gamal. Cascade multiterminal source coding. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 1199–1203, 2009.

[40] H. Yamamoto. Wyner - ziv theory for a general function of the correlated sources. *IEEE Transactions on Information Theory*, 28(5):803–807, September 1982.

[41] H. Feng, M. Effros, and S. Savari. Functional source coding for networks with receiver side information. In *Proceedings of the forty-second Allerton Conference on Computation, Communication and Control*, pages 1419–1427, 2004.

[42] V. Doshi, D. Shah, and M. Medard. Source coding with distortion through graph coloring. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 1501–1505, 2007.

[43] N. Karamchandani, R. Appuswamy, and M. Franceschetti. Distributed computation of symmetric functions with binary inputs. In *Proceedings of the IEEE Information Theory Workshop*, pages 76–80, 2009.

[44] O. Ayaso, D. Shah, and M. Dahleh. Lower bounds on information rates for distributed computation via noisy channels. In *Proceedings of the forty-fifth Allerton Conference on Computation, Communication and Control*, 2007.

[45] B. Nazer and M. Gastpar. Computing over multiple-access channels. *IEEE Transactions on Information Theory*, 53(10):3498–3516, October 2007.

[46] N. Ma, P. Ishwar, and P. Gupta. Information-theoretic bounds for multiround function computation in collocated networks. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 2306–2310, 2009.

[47] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, June 2006.

[48] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the forty-fourth annual IEEE Symposium on Foundations of Computer Science*, pages 482–491, 2003.

[49] D. Mosk-Aoyama and D. Shah. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, July 2008.

[50] O. Ayaso, D. Shah, and M. Dahleh. Counting bits for distributed function computation. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 652–656, 2008.

[51] A. G. Dimakis, A. D. Sarwate, and M. J. Wainwright. Geographic gossip: efficient aggregation for sensor networks. In *Proceedings of the fifth international conference on Information Processing in Sensor Networks*, pages 69–76, 2006.

[52] F. Benezit, A. G. Dimakis, P. Thiran, and M. Vetterli. Gossip along the way: Order-optimal consensus through randomized path averaging. In *Proceedings of the forty-fifth Allerton Conference on Computation, Communication and Control*, 2007.

[53] A. C. Yao. Some complexity questions related to distributive computing. In *Proceedings of the eleventh annual ACM Symposium on Theory of Computing*, pages 209–213, 1979.

[54] A. Ramamoorthy. Communicating the sum of sources over a network. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 1646–1650, 2008.

[55] M. Langberg and A. Ramamoorthy. Communicating the sum of sources in a 3-sources/3-terminals network. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 2121–2125, 2009.

[56] B. K. Rai, B. K. Dey, and S. Shenvi. Some bounds on the capacity of communicating the sum of sources. In *ITW 2010, Cairo*, 2010.

[57] B. K. Rai and B. K. Dey. Feasible alphabets for communicating the sum of sources over a network. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 1353–1357, 2009.

[58] H. Kowshik and P. R. Kumar. Zero-error function computation in sensor networks. In *Proceedings of the IEEE Conference on Decision and Control*, pages 3787–3792, 2009.

[59] J. Cannons, R. Dougherty, C. Freiling, and K. Zeger. Network routing capacity. *IEEE Transactions on Information Theory*, 52(3):777–788, March 2006.

[60] R. Dougherty, C. Freiling, and K. Zeger. Unachievability of network coding capacity. *IEEE Transactions on Information Theory & IEEE/ACM Transactions on Networking (joint issue)*, 52(6):2365–2372, June 2006.

[61] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, November 1999.

[62] V. V. Vazirani. *Approximation Algorithms*. Springer, first edition, 2004.

[63] A. R. Lehman and E. Lehman. Complexity classification of network information flow problems. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 142–150, 2003.

[64] K. Jain, M. Mahdian, and M. R. Salavatipour. Packing steiner trees. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 266–274, 2003.

[65] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger. Network computing capacity for the reverse butterfly network. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 259–262, 2009.

[66] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, fifth edition, 1979.

[67] D. B. West. *Introduction to Graph Theory*. Prentice-Hall, 2001.

[68] N. J. A. Harvey, R. D. Kleinberg, and A. R. Lehman. Comparing network coding with multicommodity flow for the k-pairs communication problem. *M.I.T. LCS, Tech. Rep. 964*, 2004.

[69] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

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

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

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

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

[74] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger. Network coding for computing: cut-set bounds. *IEEE Transactions on Information Theory*, 57(2):1015–1030, February 2011.

[75] J. Paek, B. Greenstein, O. Gnawali, K. Jang, A. Joki, M. Vieira, J. Hicks, D. Estrin, R. Govindan, and E. Kohler. The tenet architecture for tiered sensor networks. *ACM Transactions on Sensor Networks*, 6(4), July 2010.

[76] B. K. Rai and B. K. Dey. Sum-networks: System of polynomial equations, unachievability of coding capacity, reversibility, insufficiency of linear network coding. 2009.

[77] R. Hill. *A First Course in Coding Theory*. Oxford University Press, 1990.

[78] G. Nebe, E. M. Rains, and N. J. A. Sloane. *Self-Dual Codes and Invariant Theory*. Springer, 2006.

[79] K. M. Hoffman and R. Kunze. *Linear Algebra*. Prentice Hall, 1971.

[80] N. Alon. Combinatorial nullstellensatz. *Combinatorics, Probability, and Computing*, 8(1):7–29, 1999.