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 micro-
film and electronically:

__________________________________________

__________________________________________

__________________________________________

__________________________________________

Co-Chair

Chair

University of California, San Diego

2011
TABLE OF CONTENTS

Signature Page ....................................................... iii
Table of Contents .................................................. iv
List of Figures ....................................................... vi
List of Tables ...................................................... vii
Acknowledgements .................................................. viii
Vita ........................................................................... x
Abstract of the Dissertation ....................................... xi

Chapter 1  Introduction .............................................. 1

Chapter 2  One-shot computation: Time and Energy Complexity .. 4
  2.1  Introduction ..................................................... 5
  2.1.1  Statement of results ........................................ 7
  2.2  Problem Formulation .......................................... 8
  2.2.1  Preliminaries ................................................ 10
  2.3  Noiseless Grid Geometric Networks ......................... 11
  2.4  Noisy Grid Geometric Networks ............................ 15
  2.5  General Network Topologies ................................. 23
  2.5.1  Computing symmetric functions in noiseless networks 23
  2.5.2  Computing symmetric functions in noisy networks ... 25
  2.5.3  A generalized lower bound for symmetric functions ... 26
  2.6  Conclusion ....................................................... 27
  2.6.1  Target functions ............................................. 27
  2.6.2  On the role of \( \epsilon \) and \( \delta \) ......................... 28
  2.6.3  Network models .............................................. 29
  2.1  Computing the arithmetic sum over \( \mathcal{N} (n, 1) \) ....... 29
  2.2  Completion of the proof of Theorem 2.4.3 ................. 32
    2.2.1  Proof of Lemma 2.4.4 .............................. 32
    2.2.2  Proof of Lemma 2.4.5 .............................. 34
  2.3  Scheme for computing partial sums at cell-centers ....... 37

Chapter 3  Function computation over linear channels .......... 39
  3.1  Introduction ................................................... 40
  3.2  Problem Formulation and Notation ......................... 42
    3.2.1  Should we consider \( d > 1 \) at all? ............... 46
LIST OF FIGURES

Figure 2.1: Grid geometric network $\mathcal{N}(n,r)$. ........................................ 6
Figure 2.2: Computation of the identity function in noiseless grid geometric networks. ........................................ 11
Figure 2.3: Computation of the identity function in noiseless grid geometric networks. ........................................ 12
Figure 2.4: Computation of symmetric functions in noiseless grid geometric networks. ........................................ 13
Figure 2.5: Scheduling of cells in noisy broadcast grid geometric networks. .. 16
Figure 2.6: Computation of the identity function in noisy broadcast grid geometric networks. ........................................ 17
Figure 2.7: The $n$-noisy star network. ........................................ 19
Figure 2.8: Computation of symmetric functions in noisy broadcast grid geometric networks. ........................................ 21
Figure 2.9: Computation of symmetric functions in arbitrary noiseless networks. 25
Figure 2.10: Some notation with regards to cell $A^m_i$. ......................... 29
Figure 2.11: Partition of the network $\mathcal{N}(n,1)$ into smaller cells. .......... 30
Figure 2.12: Hierarchical scheme for computing the arithmetic sum of input messages. ........................................ 31

Figure 3.1: Example of a network with $N = 4$, $K = 7$ and $d = 1$. ........ 42
Figure 3.2: A $T$-admissible code for the $k$-threshold function for non-coherent rate-preserving networks. ......................... 64
Figure 3.3: A $T$-admissible code for the $K$-largest values function for non-coherent rate-preserving networks. ......................... 67

Figure 4.1: An example of a multi-edge tree. ........................................ 83
Figure 4.2: Description of the Reverse Butterfly network $\mathcal{N}_1$ and the line network $\mathcal{N}_2$. ............................... 95
Figure 4.3: Description of the network $\mathcal{N}_3$. ........................................ 101
Figure 4.4: Description of the network $\mathcal{N}_4$. ........................................ 103

Figure 5.1: Decomposition of the space of all target functions into various classes. 120
Figure 5.2: Description of the network $\mathcal{N}_4$. ........................................ 130
Figure 5.3: Description of the network $\mathcal{N}_5$. ........................................ 140
Figure 5.4: A network where there is no benefit to using linear coding over routing for computing $f$. ......................... 145
Figure 5.5: The butterfly network and its reverse $\mathcal{N}_6$. ......................... 154
Figure 5.6: The reverse butterfly network with a code that computes the mod $q$ sum target function. ......................... 156
Figure 5.7: The reverse butterfly network with a code that computes the arithmetic sum target function. ......................... 158
LIST OF TABLES

Table 2.1: Results for noiseless grid geometric networks. ......................... 8  
Table 2.2: Results for noisy grid geometric networks. ......................... 8  
Table 4.1: Examples of target functions. ........................................... 81  
Table 5.1: Summary of our main results for certain classes of target functions. 122  
Table 5.2: Definitions of some target functions. ............................... 124  
Table 5.3: Definition of the 4-ary map $f$. ....................................... 130
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 Karam-
chandani 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


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


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 function. 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 \( \Theta(\sqrt{\log n}) \), 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 Bachmann-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 $\rho$ is the sink that has to compute a function $f$ of the input.

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 grid 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 $r = \Theta(\sqrt{\log n})$, 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 \( x \in \{0, 1\}^n \) and permutation \( \pi \) on \( \{1, 2, \ldots, n\} \),

\[
f(x_1, x_2, \ldots, x_n) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}).
\]

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>No. of time slots</th>
<th>No. of transmissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>$\Theta (n/r^2)$</td>
<td>$\Theta (n^{3/2}/r)$</td>
</tr>
<tr>
<td>Symmetric</td>
<td>$\Theta (\sqrt{n}/r)$</td>
<td>$\Theta (n)$</td>
</tr>
</tbody>
</table>

Table 2.2: Results for noisy grid geometric networks.

<table>
<thead>
<tr>
<th>Function</th>
<th>No. of time slots</th>
<th>No. of transmissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>$\max{\Theta(n), \Theta(r^2 \log \log n)}$</td>
<td>$\max{\Theta(n^{3/2}/r), \Theta(n \log \log n)}$</td>
</tr>
<tr>
<td>Symmetric</td>
<td>$\max{\Theta(\sqrt{n}/r), \Theta(r^2 \log \log n)}$</td>
<td>$\max{\Theta(n \log n/r^2), \Theta(n \log \log n)}$</td>
</tr>
</tbody>
</table>

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

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 $\mathcal{N}(i)$ 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,

$$f : \{0, 1\}^n \rightarrow \mathcal{B}$$
where $B$ 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 $\epsilon$ 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 $\epsilon$-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 $\epsilon$ 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 $\{v^t_1, v^t_2, \ldots v^t_{S_t}\}$ and corresponding encoding functions $\{\phi^t_1, \phi^t_2, \ldots, \phi^t_{S_t}\}$. In any time slot $t \in \{1, 2, \ldots, T\}$, node $v^t_j$ computes the function $\phi^t_j : \{0, 1\} \times \{0, 1\}^{\phi^t_j} \rightarrow \{0, 1\}$ of its input bit and the $\varphi^t_j$ 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 $\rho$ 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 $x \in \{0, 1\}^n$. 
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 $\delta$-error scheme for computing $f$ if for any input $x \in \{0, 1\}^n$, $\Pr\left(\hat{f}(x) \neq f(x)\right) \leq \delta$. 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 $\delta$-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 $\rho$. 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 $\delta$-error scheme for computing $f$ over $\mathcal{N}(n, r)$ requires at least $\Omega \left( \frac{n}{r^2} \right)$ time slots and $\Omega \left( \frac{n^{3/2}}{r} \right)$ transmissions.

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

Let a cut be any set of edges separating at least one node from the sink $\rho$. It is easy to verify that there exists a collection of $\Omega \left( \frac{\sqrt{n}}{r} \right)$ disjoint cuts such that each cut separates $\Omega(n)$ nodes from the sink $\rho$, see Figure 2.2 for an example. Thus to ensure that $\rho$ 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.
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 $O(n^{3/2}/r)$ 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 $\rho$. 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 $\rho$ 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 \( O \left( \frac{n^{3/2}}{r} \right) \) transmissions and \( O \left( \frac{n}{r^2} \right) \) 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 \( \delta \)-error scheme for computing \( f \) over \( \mathcal{N}(n, r) \) requires at least \( \Omega \left( \frac{\sqrt{n}}{r} \right) \) 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 \( \Omega \left( \frac{\sqrt{n}}{r} \right) \) time slots to reach \( \rho \), 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 $O\left(\sqrt{n/r}\right)$ time slots and $O\left(n\right)$ transmissions.

Proof. We present a scheme which can compute the arithmetic sum of the input bits over $\mathcal{N}(n, r)$ in at most $O\left(\sqrt{n/r}\right)$ time slots and $O\left(n\right)$ 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 $\rho$, choose $\rho$ 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 $\{0, 1, \ldots, \Theta\left(r^2\right)\}$.

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

- $r \leq \sqrt{8\log n}$ : 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 $\mathcal{N}'(\Theta(\log n), 1)$ where each node observes a message in $\{0, 1, \ldots, \Theta\left(r^2\right)\}$. See Figure 2.4(a) for an illustration. In Appendix 2.1 we present a scheme to complete this phase using $O\left(n/r^2\right)$ transmissions and $O\left(\sqrt{n/r}\right)$ time slots.

- $r > \sqrt{8\log n}$ : The messages at cell-centers are aggregated towards $\rho$ along a tree, see Figure 2.4(b). The value at each cell-center can be viewed as a $[\log n]$-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 $[\log n]$-length binary vector. Note that it requires two time slots and $O\left(\log n\right)$ 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 $\rho$ receives the sum of all the input bits in the network. Since the depth of the aggregation tree is $O(\sqrt{n}/r)$, the phase requires $O(\sqrt{n}/r)$ time slots. There are $O(\log n)$ transmissions in each cell of the network. Hence the phase requires a total of $O(n/r^2 \times \log n) = O(n)$ transmissions.

Adding the costs of the two phases, we conclude that it is possible to compute any symmetric function using $O(n)$ transmissions and $O(\sqrt{n}/r)$ 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 $\delta$-error scheme for computing $f$ over an $\epsilon$-noise grid geometric network $N(n, r)$ requires at least $\max\{n - 1, \Omega(r^2 \log \log n)\}$ time slots and $\max\{\Omega(n^{3/2}/r), \Omega(n \log \log n)\}$ transmissions.

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

We now turn to the number of time slots required. For computing the identity function, the sink node $\rho$ 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 $N(n, r)$ into cells of size $c \times c$ with $c = r/\sqrt{8}$. Since the total number of transmissions in the network is at least $\Omega(n \log \log n)$ and there are $O(n/r^2)$ cells, there is at least one cell where the number of transmissions is at least $\Omega(r^2 \log \log n)$. 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 $\Omega(r^2 \log \log n)$ time 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 \( \delta \)-error scheme for computing \( f \) over an \( \epsilon \)-noise grid geometric network \( N(n, r) \) which requires at most
\[
\max\{O(n), O\left(r^2 \log \log n\right)\} \text{ time slots and } \max\{O\left(n^{3/2}/r\right), O(n \log \log n)\} \text{ transmissions.}
\]

**Proof.** Consider the usual partition of the network \( N(n, r) \) 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 \( 7 \times 7 \) 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
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 \(O(r^2)\) input messages. In each such chain, the rightmost cell-center maps its set of messages into a codeword of length \(O(\sqrt{nr})\) 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 \(O(\sqrt{nr})\), and then transmits it to the next cell-center, and so on. This phase requires at most \(O(\sqrt{nr} \times \sqrt{n}/r)\) time slots and a total of at most \(O(\sqrt{nr} \times n/r^2)\) transmissions in the network. From Theorem 2.2.3, this step can be executed without error with probability at least \(1 - O(1/n)\).

**Third phase:** In the final phase, the messages at the cell-centers of the left-most column are aggregated vertically towards the sink node \(\rho\), see Figure 2.6. Each cell-center maps its set of input messages into a codeword of length \(O(\sqrt{nr})\) 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 \(O(\sqrt{nr} \times \sqrt{n}/r)\) time slots and at most \(O(\sqrt{nr} \times n/r^2)\) transmissions in the network. This phase can also be executed without error with probability at least \(1 - O(1/n)\).

It now follows that at the end of the three phases, the sink node \(\rho\) can compute the identity function with probability of error at most \(O(\log^2 n/n)\). Thus for \(n\) large enough, we have a \(\delta\)-error scheme for computing any symmetric function in the network \(N(n,r)\). Adding the costs of the phases, the scheme requires at most \(\max\{O(n), O(r^2 \log \log n)\}\) time slots and \(\max\{O(n^{3/2}/r), O(n \log \log n)\}\) 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 \(\delta\)-error scheme for computing \(f\) over an \(\epsilon\)-noise grid geometric network \(N(n,r)\) requires at least \(\max\{\Omega(\sqrt{n}/r), \Omega(r^2 \log \log n)\}\) time slots and \(\max\{\Omega(n \log n/r^2), \Omega(n \log \log n)\}\) transmissions.
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 \( \delta \)-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 \( \delta \)-error scheme \( P \) for computing the parity function in an \( \epsilon \)-noise grid geometric network \( \mathcal{N}(n,r) \) which requires \( S \) transmissions. In Lemma 2.4.5 we translate the given scheme \( P \) into a new scheme \( P_1 \) 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 \( P_1 \) is also at most \( \delta \). In Lemma 2.4.4 we derive a lower bound on the probability of error of the scheme \( P_1 \) 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 \( \epsilon \) and let the input \( x \) be distributed uniformly over \( \{0, 1\}^n \). For any scheme \( P_1 \) 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 \(1 - (1 - 2\epsilon)^n\)/2.

**Proof.** See Appendix 2.2.1.

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

**Lemma 2.4.5.** Let \(\alpha \in (0, 1)\). If there is a \(\delta\)-error scheme \(P\) for computing the parity function (on \(n\) input bits) in \(N(n, r)\) with \(S\) transmissions, then there is a \(\delta\)-error scheme \(P_1\) for computing the parity function (on \(\alpha n\) input bits) in an \(\alpha n\)-noisy star network with noise parameter \(e^{O(Sr^2/n)}\), 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 \(\delta\)-error scheme for computing the parity function in \(N(n, r)\) which requires \(S\) transmissions, then by combining the Lemmas 2.4.5 and 2.4.4, the following inequalities must hold:

\[
\delta \geq \frac{1 - \left(1 - 2e^{O(Sr^2/n)}\right)^{\alpha n}}{2},
\]

\[
\iff \left(1 - 2e^{O(Sr^2/n)}\right)^{\alpha n} \geq 1 - 2\delta
\]

\[
\iff \left(2^{-e^{O(Sr^2/n)}}\right)^{\alpha n} \overset{(a)}{\geq} 1 - 2\delta
\]

\[
\implies S \geq \Omega \left(\frac{n (\log n - \log \log (1/(1 - 2\delta))))}{r^2 \log(1/\epsilon)}\right)
\]  

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

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

**Figure 2.8:** Each cell in the network $\mathcal{N}(n, r)$ is divided into sub-cells of side $\Theta\left(\sqrt{\frac{\log n}{\log \log n}}\right)$. 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 $\delta$-error scheme for computing $f$ over an $\epsilon$-noise grid geometric network $\mathcal{N}(n, r)$ which 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.**

**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 $O(n/r^2 \times \log n)$ total transmissions and $O(\log n)$ time slots.

- $r > \sqrt{\log n / \log \log n}$: In this case, we first divide each cell into smaller sub-cells with $\Theta(\log n / \log \log n)$ 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 $O(\log \log n)$ transmissions from each node in the sub-cell. Since there are $O(r^2)$ nodes in each cell and only one node in a cell can transmit in a time slot, this step requires $O(r^2 \log \log n)$ time slots and a total of $O(n \log \log n)$ 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 $O(\log n)$ and transmits it to the corresponding cell-center. This step requires a total of $O(n \log \log n)$ transmissions in the network and $O(r^2 \log \log n)$ 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 $O(n \log \log n)$ transmissions in the network and $O(r^2 \log \log n)$ 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 $\rho$ can compute the sum of all the input bits in the network. We have the following two cases, depending on the connection radius $r$.

- $r \geq (\sqrt{n \log n})^{1/3}$: 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 $\Theta (\log n)$. 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 $\Theta (\log n)$. The process continues till the sink node $\rho$ 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 $O (n \log n/r^2)$ transmissions in the network and $O (\sqrt{n}/r \times \log n)$ time slots.

- $r \leq (\sqrt{n} \log n)^{1/3}$: 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 $O (n \log n/r^2)$ transmissions in the network and $O (\sqrt{n}/r)$ 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 $\delta$-error scheme for computing any symmetric function in the network $\mathcal{N}(n, r)$. It 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.

### 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 $r = \Omega (\sqrt{\log n})$ 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 comput-
ing 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 $\Pi_i = \{P^1_i, P^2_i, \ldots, P^{s_i}_i\}$ of the set of nodes in the network $\mathcal{V}$ into $s_i$ disjoint cells such that each $P^j_i = \bigcup_{k \in T^j_i} P^k_{i-1}$ where $T^j_i \subseteq \{1, 2, \ldots, s_{i-1}\}$, 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 $P^j_i$, a designated cell-center $c^j_i \in P^j_i$. Let $c^1_h$ be the designated sink node $v^*$. 

- For each cell $P^j_i$, let $S^j_i$ 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 $P^k_{i-1}$, i.e., the set of nodes $\bigcup_{k \in T^j_i} c^k_{i-1} \cup c^j_i$. Let $l^j_i$ denote the number of edges in $S^j_i$.

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 $\{P_k^1\}_{k=1}^4$ smaller cells from the previous level in the hierarchy. Each of the cell-centers $c_k^1$ (denoted by the green nodes) holds the sum of the input bits in the corresponding cell $P_k^1$. 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

$$\sum_{t=0}^{h-1} \sum_{j=1}^{s_{t+1}} l_j^{t+1} \cdot \log(|P_j^t|).$$

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 $h = \log \sqrt{n}$, $s_t = n/2^{2t}$, $l_j^t \leq 4 \cdot 2^{t-1}$, $|P_j^t| = 2^{2t}$, 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** $C(z)$: a network $\mathcal{N}$ of $n$ nodes is said to satisfy the clique-cover property $C(z)$ 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 $\delta \in (0, \frac{1}{2})$, $\epsilon \in (0, \frac{1}{2})$ and $\mathcal{N}$ be any connected network of $n$ nodes with $n \geq 2/\delta$. For $z \geq 1$, if $\mathcal{N}$ satisfies $C(z)$, then there exists a $\delta$-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 $\mathcal{N}(i)$ denotes the set of neighbors for any node $i$. For any network, define the average degree as

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

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

**Theorem 2.5.2.** Let $\delta \in (0, \frac{1}{2})$ and let $\epsilon \in (0, \frac{1}{2})$. There exists a symmetric target function $f$ such that any $\delta$-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 $\sigma$ 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 $\alpha n^2$ of such nodes in the network since the average degree is $d(n)$. Thus, we obtain an $(\frac{\alpha n^2}{2}, (1 - \frac{\alpha}{2})n)$ 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 $\delta \in (0, \frac{1}{2})$, and let $\epsilon \in (0, \frac{1}{2})$. Any $\delta$-error scheme for computing $f$ over any ring or tree network of $n$ nodes requires at least $\Omega(n \log n)$ 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 \( f(x) = 0 \) for all \( x \) 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 \( \delta \)-error scheme for computing \( f \) over an \( \epsilon \)-noise network \( N(n, r) \) 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 \( \delta \)-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 \( \epsilon \) and \( \delta \)

Throughout the paper, the channel error parameter \( \epsilon \) and the threshold \( \delta \) 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 \( \epsilon \) is considered, and always achieve a probability of error \( \delta \) 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 \( \epsilon \) and increase with smaller values of \( \delta \). 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 \( \epsilon \) 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 \( \epsilon \) as \( 1/(-\log(\epsilon(1-\epsilon))) \). The two expressions are close for small values of \( \epsilon \).
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 $N(n, 1)$

Consider a noiseless network $N(n, 1)$ 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
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 \( A^m_i \) and the node in the lower-left corner of \( A^m_i \) by \( u(A^m_i) \). For any \( m \) which is a power of 2, \( m \geq 2 \), \( A^m_i \) can be divided into 4 smaller cells, each of size \( m/2 \times m/2 \), see Figure 2.10. Denote these cells by \( \left\{ A^{m/2}_{ij} \right\}_{j=1}^4 \).

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 \( \left\{ A^{2k+1}_i \right\}_{i=1}^{n \over 2^{2(k+1)}} \) each of size \( 2^{k+1} \times 2^{k+1} \).
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 $u\left(A_{ij}^{2^k}\right)$ has the sum of the input messages corresponding to the nodes in the cell $A_{ij}^{2^k}$. 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 $\{A_{ij}^{2^k}, \ldots, A_{ij}^{2^k}\}$, see Figure 2.12. Each corner node $u\left(A_{ij}^{2^k}\right)$, $j = 1, 2, 3, 4$ possesses the sum of the input messages corresponding to the nodes in the cell $A_{ij}^{2^k}$.

The partial sums stored at $u\left(A_{ij}^{2^k}\right)$, $j = 1, 2, 3, 4$ 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

$$2 \left(2^k + \log \left(2^{2(k+1)q}\right)\right)$$
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 \( \rho \) can compute the sum of the input messages for any input \( x \in \{0, 1, \ldots, q-1\}^n \). The total number of steps in the scheme is \( \log \sqrt{n} \).

The number of time slots that the scheme takes is at most
\[ \sum_{k=0}^{\log \sqrt{n}-1} 2 \left( 2^k + \log \left( 2^{2(k+1)}q \right) \right) \leq O \left( \log q \cdot \log n + \sqrt{n} \right). \]

The total number of transmissions made by the scheme is at most
\[ \sum_{k=0}^{\log \sqrt{n}-1} \frac{4n(k + 1 + \log q)}{2^{k+1}} \leq O \left( n + \log q \right). \]

### 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 \( A^\ast \) receives. Denote the received vector by \( y \). The objective of \( A^\ast \) is to compute the parity of the input bits \( x_1, x_2, \ldots, x_n \). Thus, the target function \( f \) is defined as
\[ f \left( x \right) = x_1 \oplus x_2 \oplus \ldots \oplus x_n. \]

Since the input \( x \) is uniformly distributed, we have \( \Pr \left( f \left( x \right) = 0 \right) = \Pr \left( f \left( x \right) = 1 \right) = 1/2 \). 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. \( f \left( y \right) \) as an estimate for \( f \left( x \right) \), and then compute the corresponding probability of error. From the definition of Maximum
Likelihood estimation, we have

\[
\hat{f} = \begin{cases} 
1 & \text{if } \Pr(y|f(x) = 1) > \Pr(y|f(x) = 0) \\
0 & \text{otherwise}
\end{cases}
\]

Next,

\[
\Pr(y|f(x) = 1) = \sum_{x \in \{0,1\}^n \text{ s.t } f(x) = 1} \Pr(x|f(x) = 1) \cdot \Pr(y|x)
\]

\[
\overset{(a)}{=} \kappa \sum_{x \in \{0,1\}^n \text{ s.t } f(x) = 1} \Pr(y_1|x_1) \Pr(y_2|x_2) \ldots \Pr(y_n|x_n)
\]

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

\[
\Pr(y|f(x) = 0) = \kappa \sum_{x \in \{0,1\}^n \text{ s.t } f(x) = 0} \Pr(y_1|x_1) \Pr(y_2|x_2) \ldots \Pr(y_n|x_n).
\]

Putting things together, we have

\[
\Pr(y|f(x) = 0) - \Pr(y|f(x) = 1) = \kappa \left( \sum_{x \in \{0,1\}^n \text{ s.t } f(x) = 0} \prod_{i=1}^n \Pr(y_i|x_i) - \sum_{x \in \{0,1\}^n \text{ s.t } f(x) = 1} \prod_{i=1}^n \Pr(y_i|x_i) \right)
\]

\[
\overset{(a)}{=} \kappa \prod_{i=1}^n \left( \Pr(y_i|x_i = 0) - \Pr(y_i|x_i = 1) \right)
\]

\[
= \kappa (-1)^{n_1(y)} (1 - 2\epsilon)^n,
\]

where \( n_1(y) \) is the number of components in \( y \) with value 1. The above equality \( (a) \) can be verified 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

\[
\Pr(y|f(x) = 0) - \Pr(y|f(x) = 1) = \begin{cases} 
> 0 & \text{if } f(y) = 0 \\
< 0 & \text{if } f(y) = 1.
\end{cases}
\]

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

\[
\Pr_{ML}(\text{Error}) = \Pr(f(y) \neq f(x)) = \sum_{j=1}^{\lceil \frac{n}{2} \rceil} \Pr\left( \sum_{i=1}^{n} x_i \oplus y_i = 2j - 1 \right) = \sum_{j=1}^{\lceil \frac{n}{2} \rceil} \left( 2j - 1 \right) \epsilon^{2j-1} (1 - \epsilon)^{n-2j+1} = 1 - \frac{(1 - 2\epsilon)^n}{2}.
\]

Hence, for any scheme \( P_1 \) which computes the parity function \( f \) in an \( n \)-noisy star network, the probability of error is at least \( (1 - (1 - 2\epsilon)^n) / 2 \).

### 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 \( \sigma \) 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 \( P \) 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 \( \sigma \) 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 \( \epsilon \)-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 $\delta$-error scheme $\mathcal{P}$ for computing the parity function in an $\epsilon$-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 $\delta$-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 $\delta$.

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

**Lemma 2.2.1.** If there is a $\delta$-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 $\delta$-error, $O(S/n)$-bounded scheme $\mathcal{P}_d$ for computing the parity function (on $\alpha n$ 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 $\sigma$ and auxiliary nodes $A$ 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 $\alpha n$ 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 $\delta$-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 $\delta$-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 \( f : \{0, 1\}^{\alpha n} \rightarrow \{0, 1\} \) and any \( \delta \)-error, \( O(S/n) \)-bounded scheme \( P_s \) 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 \( \delta \)-error scheme \( P_1 \) for computing \( f \) in an \( \alpha n \)-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 \( P_s \), 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 \( \epsilon \)-noisy copies of each such input bit. Since \( P_s \) 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 \( \epsilon \)-noisy copies of each input bit.

Next, we use the scheme \( P_s \) to construct a scheme \( P_1 \) for computing \( f \) in an \( \alpha n \)-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 \( \epsilon \)-noisy copies for each input bit. Then onwards, the auxiliary node \( A^* \) can simulate the scheme \( P_s \). This is true since for \( P_s \) 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 \( \gamma \)-noisy copy of \( 0 \) (respectively of \( 1 \)), then the output is a sequence of independent \( \epsilon \)-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 \( \lambda = -\ln(4\epsilon(1-\epsilon)) \). 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 \( 8 \log n \lambda \) 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.

\[
\Pr(\text{Error}) \leq \left( 4 \left( c^2 e^{-\frac{4 \log n}{c^2}} + e^{-\frac{4 \log n}{c^2}} \right) \left( 1 - (c^2 e^{-\frac{4 \log n}{c^2}} + e^{-\frac{4 \log n}{c^2}}) \right) \right)^c^2
\]

\[
\leq \left( 4 \cdot 9 c^2 e^{-\frac{4 \log n}{c^2}} \right)^c^2
\]

\[
\leq \frac{1}{n^2}, \text{ for } n \text{ large enough}
\]

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 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. We consider both cases, when the linear transformation is known at the sources and the sink, as well as the case when it is not. 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

Consider a network where multiple sources are connected to a sink via relays. We ask the following question: assuming that the relays perform a given operation, for example randomized network coding, can we calculate different functions of the source data at the sink efficiently without altering the relays operation? This question is motivated by practical considerations in sensor network applications.

Energy efficient data collection is the main task of wireless sensor networks (WSN). Data measurements (such as temperature readings at different locations in a building) are collected at the sensor sources. The sink is interested in a function of these measurements (for example, the average temperature in each room). The communication challenge is that the relays have limited available energy while use of the wireless transceiver absorbs a large part of the energy budget. This challenge has stimulated research efforts towards in-network aggregation and function computation, both from a systems [17], [18], and a theoretical viewpoint [3], [19]. This problem has also been examined in the context of network coding: for example in [20], [21], the authors examine the problem of designing network codes that allow function computation at designated sinks. Function computation has also been examined using information theoretical tools, see for example [22], [23], and references therein.

Our work is motivated by practical considerations, resulting from a system’s perspective. It is well known that implementing networking protocols for WSN is hard [24], therefore implementing a new network protocol for every function is extremely costly. The main difficulty in implementing robust protocols is the distributed nature of the system; it is thus advisable to reuse already deployed protocols whenever possible. This however clashes with the need to optimize network operation for the specific function being computed.

To resolve this conflict, recent work in systems [24, 25] advocates 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. This solution is potentially sub-optimal from an energy efficiency standpoint but has the advantage of simplifying the development of support for new functions.
Inspired by this, in this chapter we study this system’s approach from a combinatorial point of view. We assume that the network operation is fixed, i.e., the relay nodes always perform a fixed operation regardless of the function being computed. On the contrary, each source adapts its operation to the function being computed. In particular we assume that sources map their measurements to symbols from a finite field using a function-specific codebook and send them to the relays. The relay nodes transmit fixed linear combinations of the received symbols. The sink therefore receives linear combinations of the input symbols, and the whole network operation can thus be represented by a single linear transform of the symbols sent by the sources. We are interested in combinatorial designs and bounds for function computation in this network model.

This model is substantially different from existing work on function computation where one has the choice of changing the relays operation, according to the particular target function. We underline that a scheme which optimizes the intermediate node operations according to the function to be computed might need fewer transmissions than our schemes. But our approach has the advantage of simplifying the implementation of support for new functions. Moreover as a byproduct of our analysis we will show that it is possible to easily develop protocols that do not require knowledge of the topology of the network.

We select to model the relay operation with linear operations because this captures the behavior of well studied network protocols: routing, random network coding [26, 27] and average computation (see for example [28]). When performing average computation (and routing) both the sink and the sources are aware of the linear transform being performed by the network. On the other hand, with random network coding, if we do not use coding vectors\(^1\), only rank properties of the matrix that describes the network operation are known. We thus distinguish between two types of communication: coherent communication where the network operation is known at the sources and the sink and non-coherent communication where the linear transform performed by the network is not known (but has some known properties). The study of non-coherent communication will also give us intuition on how to implement a function computation protocol.

\(^1\)Note that even if we do use coding vectors, only the sink would learn the channel linear transformation.
that does not require topology information.

The chapter is organized as follows. In Section 3.2, we introduce the model and we discuss some of its fundamental properties. We study the two cases of coherent and non-coherent communication in Sections 3.3 and 3.4 respectively. Finally, we conclude in Section 3.5.

### 3.2 Problem Formulation and Notation

![Network Diagram](image)

**Figure 3.1:** Example of a network with $N = 4$, $K = 7$ and $d = 1$.

We consider a set of $N$ sources $\sigma_1, \ldots, \sigma_N$ connected to a sink $\rho$ via a network $\mathcal{N}$. The network is composed of $R$ relays $r_1, \ldots, r_R$. The relays, sources, and sink are arranged in a directed acyclic multigraph $G = (\{\sigma_i\} \cup \{r_i\} \cup \{\rho\}, E)$ that represents the connectivity of each node. Given an edge $e = (v_1, v_2)$, we call $t(e) = v_1$ the tail of the edge and $h(e) = v_2$ its head. We assume that each source has no in-coming edges and therefore it has no knowledge of what the other sources are sending. Moreover we assume that it has exactly $d$ out-going edges\(^2\). Each source $\sigma_i$ is either inactive or observes a message $u_i \in \mathcal{A}$, where $\mathcal{A}$ is a finite alphabet. For ease of notation, we will set $u_i = \phi$ when the source is inactive and we say the source chooses a symbol from $\bar{\mathcal{A}} = \mathcal{A} \cup \{\phi\}$. The sink needs to compute a target function $f$ of the source messages, where $f$ is of the form:

$$f : \bar{\mathcal{A}}^N \to \mathcal{B}.$$\(^2\)

\(^2\)We assume that all sources have the same out-degree for simplicity. This constraint could be dropped but we will show in the following that considering degree 1 is enough for our purposes.
Time is slotted and function computation is performed over $T$ timeslots. During every timeslot $t$, each active source $\sigma_i$ sends on each of its out-going edges a symbol chosen according to a codebook $C_i$ that maps each input message $u_i$, timeslot $t$, and out-going edge index $j$ to a symbol $x_{i,j,t} \in \mathbb{F}_q$.Inactive sources do not send any symbol on their out-going edges. The relays assume that those edges are carrying the symbol $0 \in \mathbb{F}_q$.

The rest of the network is performing linear network coding: during each timeslot, every relay transmits on each out-going edge $e$ a symbol $z_e \in \mathbb{F}_q$ constructed as follows:

$$ z_e = \sum_{e': h(e') = t(e)} b_{e,e'} \cdot z_{e'} $$

where $b_{e,e'} \in \mathbb{F}_q$. In other words, the output on a given edge is a linear combination of the symbols received by the transmitting relay. We can therefore represent the symbols received by the sink as:

$$ Y = A \cdot X $$

where $Y$ is an $M \times T$ matrix that represent the symbols received by the sink, $X$ is a $d \cdot N \times T$ matrix that represent the symbols sent by the sources and $A$ is an $M \times d \cdot N$ matrix called the channel matrix that describes the relationship between input and output symbols of the network. The $j$-th column of $X$ contains the $d \cdot N$ symbols sent by the sources on their out-going edges during timeslot $j$. A given row of $X$ contains all the symbols sent on a specific out-going edge from a specific source. The $j$-th column of $Y$ contains the $M$ symbols received by the sink during timeslot $j$ on its $M$ incoming edges. Each row of $Y$ represents the $T$ symbols received on a specific edge. Matrix $A$ can be computed from all the coefficients $b_{e,e'}$.

In this chapter we assume that network operation is fixed. We will study what is the smallest time $T$ such that there exists a set of $N$ codebooks $\{C_i\}$ that allow the sink to evaluate the target function $f$ correctly over the given network. This minimum time will be denoted by $T_c(f,A)$ for the case when both the sources and sink know the network matrix $A$ (coherent communication), and by $T_{nc}(f,S)$ where $S$ is a set of possible channel matrices when the the channel matrix is unknown to all (non-coherent communication). We say that for a given function and a given network described by a matrix (respectively by a set of possible channel matrices) there exists a $T$-feasible
coherent (non-coherent) code if it is possible to correctly compute the function in less than $T$ timeslots on that specific network.

Our ultimate goal is to minimize the number of symbol sent by the sources, that is $T \cdot N \cdot d$. This criterion is particularly useful in the case when the limiting factor for the network lifespan is the energy budget available at the sources.

For the matrix $A$, we will consider the following two cases:

- **Rate-preserving network**: the matrix $A$ is invertible (and therefore $M = dN$). In this network the sink (if it knows $A$) can reconstruct all the symbols sent by the sources. This is the best situation we can hope for in the sense that the network delivers the maximum amount of information from the sources to the sink.

- **Constant-rate network**: $A$ is a $1 \times dN$ matrix with each $(A)_{i,j} \neq 0$. In this setup, the sink receives the minimum number of symbols but this single symbol depends on each of the symbols sent by the sources and therefore the sources can potentially use all their out-going channels to convey their message to the sink.

We focus our attention on these two cases because they model some interesting and well-known protocols. Assume that each edge in $G$ represents a packet exchanged in the network:

- Rate-preserving networks naturally model two types of protocols: routing and random network coding.

In routing, packets produced at the sources are forwarded towards the sink by relays using a routing tree. Assume each source sends exactly one packet. To represent this protocol with our model, we first consider the following network graph: relays and sources are connected accordingly to the routing tree; the number of edges between any parent and child is equal to the number of sources that route their traffic through the child. Relays transmit each of the symbols they receive on one of their out-going edges (they send therefore a linear combination of only one of the the incoming symbols). The sink has exactly $N$ in-coming links, on each of which it receives one of the sources symbols. By appropriately numbering these in-coming edges we can see that the channel matrix is $A = I$. 
In random network coding, nodes are again organized in a routing tree and every relay transmits as many packets as it has received. Each transmitted packet is a (fixed) random linear combination of the packets received by the relay. The sink therefore receives $N$ linear combinations of the source symbols. If the field size $q$ over which the network operates is large enough, then with high probability the received linear combinations are linearly independent. It is easy to see that we can describe this protocol using the same graph proposed for routing. The channel matrix in this case will not be the identity matrix but instead it will be a random invertible matrix.

- Constant-rate networks model aggregation protocols that use a spanning tree connecting all the sources to the sink. Every node in the network sends at most one packet. Every relay linearly combines all the received symbols in a single symbol that is forwarded. What is obtained at the sink is a weighted sum (over $\mathbb{F}_q$) of the symbols transmitted by the sources.

In the rest of the chapter, we will consider the network as a black box that delivers a linear transformation of the inputs at the sink. Therefore the results that we present are valid for other instantiations of the network operation. For instance if the network is running any distributed average computation protocol, then it can be modeled as a constant-rate network even if in the specific protocol being used at each relay cannot be described as linear combination of the inputs.

An important observation for the rest of the chapter is that if we consider the symbols sent by a given source on a given edge $e$ as a vector, i.e., $x_e = (x_{e,1}, \ldots, x_{e,T})^\top$, then the vector which is received by the sink on each edge $\hat{e}$, $y_{\hat{e}} = (y_{\hat{e},1}, \ldots, y_{\hat{e},T})^\top$, is a linear combination of the input vectors. The exact linear combination that is received depends on $A$, but in any case the subspace spanned by the received vectors $\{y_{\hat{e}}\}$ is contained in the subspace spanned by the input vectors $\{x_e\}$, with equality if the matrix $A$ is invertible. When $A$ is not known, the sink must rely solely on the received subspace for decoding. Thus, sources can be thought of as transmitting $d$-dimensional subspaces into the network and their codebooks can be seen as being mappings from the alphabet $\tilde{A}$ to subspaces of $\mathbb{F}_q^T$ of dimension $d$. This approach is called subspace coding [29–31].
In the rest of the chapter, we will use this observation to greatly simplify our analysis for non-coherent communication. In particular we will call \( \pi_i^j \) the \( d \)-dimensional subspace of \( \mathbb{R}^T_q \) sent by source \( \sigma_i \) when it observes the message \( u_j \).

### 3.2.1 Should we consider \( d > 1 \) at all?

As already mentioned, \( d \) is a design parameter in our setup. However, the next proposition shows that, if the network is either rate-preserving or constant-rate, then the sources send at least as many symbols in networks with \( d > 1 \) as in networks with \( d = 1 \). For this reason, in the following we will restrict our attention to the case when \( d = 1 \).

**Proposition 3.2.1.** Consider two rate-preserving (constant-rate) networks: \( N \) with source degree \( d \) and channel matrix \( A \), and \( N' \) with source degree 1 and channel matrix \( A' \). If the network \( N \) admits a coherent (non-coherent) code \( \{C_i\} \) of length \( T \), then the network \( N' \) admits a coherent (non-coherent) code \( \{C'_i\} \) of length \( dT \).

The proof is given in the appendix.

Since we are interested in characterizing the minimum number of symbols that sources need to transmit, in the rest of the chapter we will always consider \( d \) to be 1. To simplify the notation, we will let \( x_{i,t} \) denote the only symbol sent from source \( \sigma_i \) at time \( t \). Rows in the matrix \( X \) will be arranged such that \( (X)_{i,t} = x_{i,t} \). We will denote the symbols sent by source \( \sigma_i \) with \( x_i = (x_{i,1} \ldots x_{i,T}) \).

### 3.3 Coherent communication

In this section we assume that both sources and sink know the channel matrix and we study \( T_c(f, A) \), the minimal number of timeslots necessary to compute the function \( f \) over the known channel \( A \).

### 3.3.1 A lower bound for arbitrary networks

First we consider a generic network \( A \). Proposition 3.3.1 provides a lower bound on \( T_c(f, A) \) that formalizes the following simple observation. Rank properties of the
matrix $A$ and the field size $q$ limit the number of possible output values that the sink observes in each timeslot. If for a given function $f$, the number of outputs of the network in $T$ timeslots is smaller than the possible output values for the function, then $f$ cannot be computed in $T$ timeslots.

**Proposition 3.3.1.** Let $S \subseteq \{1, 2, \ldots, N\}$ be a subset of the source indices, and denote these indices as $S = \{i_1, \ldots, i_{|S|}\}$. Let $\hat{u}_{i_1}, \ldots, \hat{u}_{i_{|S|}} \in \hat{A}$ be the set of messages observed by the sources in $S$. Let $A$ be a channel matrix over $\mathbb{F}_q$ and let $\hat{A}$ be the matrix obtained by deleting columns $i_1, \ldots, i_{|S|}$ from $A$. Then,

$$T_c(f, A) \geq q^{-\text{rank}(\hat{A})} \cdot |\{f(u) | u \in \hat{A}^N : u_i = \hat{u}_i \forall i \in S\}|$$

**Proof.** Assume that function $f$ can be computed in $T$ timeslots. Let $i_{|S|+1}, \ldots, i_N$ be the indices of the sources not in $S$. Let $\bar{A}$ be the matrix obtained by keeping only the columns $i_1, \ldots, i_{|S|}$ of $A$. Let $\bar{X} = (x_{i_1}, \ldots, x_{i_{|S|}})^\top$ and $\bar{X} = (x_{i_{|S|+1}}, \ldots, x_{i_N})^\top$. The network output can be rewritten as

$$Y = \bar{A} \cdot \bar{X} + \hat{A} \cdot \bar{X}.$$ 

Note that since the messages of the sources in $S$ are fixed, $\bar{A} \cdot \bar{X}$ remains constant. Since the column space of $\bar{A}$ has dimension $\text{rank}(\bar{A})$, it follows that there are $T \cdot q^{\text{rank}(\bar{A})}$ possible outputs. Clearly the number of possible network outputs has to be bigger than the number of possible function values when fixing $u_{i_1}, \ldots, u_{i_{|S|}}$. This fact then can be used to derive the bound on the minimal $T$. ■

**Corollary 3.3.2.** Let $A$ be a channel matrix over $\mathbb{F}_q$ and $f$ a function then:

$$T_c(f, A) \geq q^{-\text{rank}(A)} \cdot |\text{range}(f)|.$$ 

In the following we will see that for some channel matrices and functions the bound in Proposition 3.3.1 can be achieved however in general this is not the case. The following situation provides an example:
Example 3.3.3. Assume that $N = 2$, $\mathcal{A} = \{0, 1\}$, $q = 2$, $T = 1$ and $A = (1 \ldots 1)$. Let $x_i^j$ denote the symbol sent by the source $\sigma_i$ when $u_i = j \in \{0, 1\}$. Consider the function $f(u_1, u_2) = u_1 \text{ AND } u_2$ that takes the value 1 if and only if $u_1 = u_2 = 1$. Clearly, each source needs to send a different symbol corresponding to observed values 0 and 1, i.e.,

$$x_1^0 \neq x_1^1 \text{ and } x_2^0 \neq x_2^1. \quad (3.1)$$

Moreover, we need that $(x_1^0 + x_2^0) \mod 2 \neq (x_1^1 + x_2^1) \mod 2$, and hence either $x_1^0 \neq x_2^0$ or $x_1^1 \neq x_2^1$ must hold, which contradicts (3.1). The same argument extends to any arbitrary number of sources $N$.

### 3.3.2 Rate-preserving networks

We now consider rate-preserving networks, i.e. networks characterized by an invertible channel matrix $A$. In these networks the sink can reconstruct, by multiplying the received matrix with the inverse of the channel matrix, the symbols sent by the sources. This makes the code design quite straightforward.

For each source $\sigma_i$, let $c_i$ be a coloring of $\tilde{A}$, a function that associates a color to each element of $\tilde{A}$, with the smallest number of colors and that satisfies the following property. For any $a, b \in \tilde{A}$, we have $c_i(a) = c_i(b)$ iff for any $u_1, u_2 \in \tilde{A}^N$ such that $u_{1i} = a$ and $u_{2i} = b$ we have $f(u_1) = f(u_2)$. In other words $c_i$ colors the input messages in such a way that if each source transmits the color corresponding to its message, the sink can correctly reconstruct the function value.

**Proposition 3.3.4.** Given a network with channel matrix $A$ and a target function $f$,

$$T_c(f, A) = \max_{i \in \{1, 2, \ldots, N\}} \lceil \log_q \left( \left| \{ c_i(a) : a \in \tilde{A} \} \right| \right) \rceil$$

**Proof.** Assume that $T$ satisfies the above criterion. Then the code is as follows: each source transmits the color of its message as a sequence of $T$ symbols over $\mathbb{F}_q$. Since we have a rate-preserving coherent network, the sink can reconstruct all the colors sent by
the sources. From the definition of the colorings $c_i$'s, it is easily seen that the code is admissible.

For the converse, it is easy to verify that any source $\sigma_i$ must transmit distinct $T$-length vectors into the network for any $a, b \in \tilde{A}$ such that $c_i(a) \neq c_i(b)$. Thus we have

$$q^T \geq |\{c_i(a) : a \in \tilde{A}\}|.$$  

The result then follows.

Notice also that in the previous proposition $T_c(f, A)$ depends only on the field size over which the network operates and not on the channel matrix.

### 3.3.3 Constant-rate networks

Now we turn our attention to constant-rate networks, i.e. networks where $A$ is a $1 \times N$ matrix with non zero entries. Observe that it is not necessary to consider arbitrary $A$:

**Lemma 3.3.5.** Let $1 \times N$ matrix $A$ over a finite field $\mathbb{F}_q$ be the channel matrix of a constant-rate network, then

$$T_c(f, A) = T_c(f, B)$$

where $B$ is a $1 \times N$ all ones matrix over $\mathbb{F}_q$.

**Proof.** Given a coherent code for a network with channel $B = (1\ldots1)$, it is possible to construct an admissible code for any other constant-rate network with channel $A$ by setting $x'_{i,t} = (A)^{-1}_{i,i} \cdot x_{i,1,t}$.

For this reason, henceforth whenever we discuss coherent communication over constant-rate networks we will only consider the channel $B = (1\ldots1)$. We call such a networks *sum network* and the corresponding channel matrix $B$ as the *sum channel*.

In the following section we will study for some widely used function $f$ for which $q$ we have that $T_c(f, B) = 1$, that is for which $q$ the function can be computed in one timeslot.
One timeslot computation

In this section we will consider only $q$ prime because, as we will discuss in Section 3.3.4, computation over networks that operate with non prime $q = p^m$ where $p$ is a prime can be seen as multiple uses of a network that operates over $p$.

**$m$-state function** Each source observes a value in the set $A = \{1, \ldots, K\}$, and maps it to one of $m$ values, that we call “states”. Let $S$ be the set of possible states. The objective of the sink is to learn the state of each source. That is, evaluating the function $f$ results in one of the $m^N$ possible outputs. The $m$-state captures several interesting functions as special cases, that include:

- **The membership function:** A membership criterion is specified, for example, whether the observed value belongs to a specific subset of the alphabet $A$. Each node determines whether it is a member, and accordingly sends a message from the set $\{\text{member, not member}\}$. In this case $m = 2$.

- **The identity function:** The sink wants to learn the message $u_i$ that each source $\sigma_i$ observes. In this case $m = K$.

The range of the $m$-state function is $(m + 1)^N$ and therefore from Corollary 3.3.2, for $A = (1 \ldots 1)$ it is necessary that $q \geq (m + 1)^N$. This lower bound on $q$ is also sufficient, and is achieved by the following scheme.

Encoding  $x_i = m^{i-1} \cdot C(u_i)$

Decoding  $f(u) = (C^{-1}(\lfloor \frac{u}{m^0} \rfloor \mod m), \ldots, C^{-1}(\lfloor \frac{u}{m^{N-1}} \rfloor \mod m))$.

where $C : S \cup \{\phi\} \to \{0, 1, \ldots, m\} \in \mathbb{F}_q$ is an injective function from the state values to the elements in $\mathbb{F}_q$, with $C(\phi) = 0$.

To understand why this scheme works, first note that if we express the symbol sent by every source $\sigma_i$ in base $m$, it is a number of the form $C(u_i) 0 \ldots 0$, i.e., a digit equal to value $C(u_i)$ (which is less than $m$) followed by $i - 1$ zeros. Now if we compute the sum of the input symbols in base $m$, note that no carry ever occurs and that only one input symbol is influencing the value of the $i$-th digit of the sum. This means that

\footnote{This map can potentially be different for every source node.}
by expressing the network output in base \( m \), we can reconstruct all the \( C(u_i) \) that were sent by the sources. By inverting the mapping \( C \), we can therefore reconstruct the states in which the sources are.

Note that, for \( m = K \), we get an upper bound on the required field size to calculate an arbitrary function \( f \), since we can first calculate the identity function at the sink and then, having collected all the measurements, the function value.

**Threshold function** Assume that \( \mathcal{A} = \{0, 1\} \), the threshold function is defined as

\[
f(u) = \begin{cases} 
1 & \text{if } |\{u_i = 1\}| \geq l \\
0 & \text{otherwise.}
\end{cases}
\]

Note that the \( N \)-input OR and AND binary operations as well as the majority function are special cases of the threshold function. A scheme to compute the threshold function when \( q > N \) is as follows:

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Decoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i = u_i )</td>
<td>( f(u) = 1_{{y \geq l}} )</td>
</tr>
</tbody>
</table>

We now show that any scheme to compute the threshold function over the \((1 \ldots 1)\) channel requires \( q > N \). Assume that all sources are active. Let \( x_i^j \in \mathbb{F}_q \) be the symbol sent by source \( \sigma_i \) when it observes \( j \in \mathcal{A} \). Then for any collection of distinct indices \( I = \{i_1, i_2, \ldots, i_k\} \) (with each \( i_j \in \{1, 2, \ldots, N\} \)), we have

\[
\sum_{i \in I} x_i^0 \neq \sum_{i \in I} x_i^1.
\]

(3.2)

To see why this is true, consider two distinct input vectors \( u, v \in \mathcal{A}^N \). Vector \( u \) has 1 as input for all source indices in \( I \). Additionally, it has 1 as input for a collection of \( \max\{0, l - |I|\} \) other sources indices, say \( J \), each of which is not in \( I \). All the other inputs are set to 0. On the other hand, \( v \) has 1 as input for only the indices in \( J \) and all the other inputs are set to 0. Note that the number of inputs in \( J \) is always less than \( l \).
As a result we have $1 = f(u) \neq f(v) = 0$ and hence we require

$$\sum_{i \in I} x_i^1 + \sum_{i \in J} x_i^1 + \sum_{i \notin I \cup J} x_i^0 \neq \sum_{i \in I} x_i^0 + \sum_{i \in J} x_i^1 + \sum_{i \notin I \cup J} x_i^0 \implies \sum_{i \in I} x_i^1 \neq \sum_{i \in I} x_i^0.$$ 

From (3.2), the sink should receive a distinct symbol (in $\mathbb{F}_q$) from the network for each of the following input vectors:

$$(0 \ 0 \ldots 0), (0 \ 0 \ldots 1), (0 \ 0 \ldots 1 1), \ldots, (1 \ 1 \ldots 1).$$

Since the field size is $q$, this implies that $q > N$.

**Histogram function** Let $A = \{1, \ldots, K\}$. The histogram function $f(u) : A^N \rightarrow \{0, \ldots, N\}^K$ is defined as follows:

$$f(u) = v \text{ where } (v)_j = \sum_{i=1}^{N} 1_{\{u_i = j\}}$$

The histogram function is very useful, since it can be used to compute all symmetric functions of the node observations. For instance, all statistical functions fall in this category, such as the median, the mode and the maximum function.

When $q > N(N + 1)^{K-1}$, this function can be computed with the following scheme:

**Encoding**

$$x_i = \begin{cases} 
(N + 1)^{u_i - 1} & \text{if } u_i \neq \phi \\
0 & \text{otherwise}
\end{cases}$$

**Decoding**

$$f(u) = \left( \left\lfloor \frac{y}{N} \right\rfloor \mod (N + 1), \ldots, \left\lfloor \frac{y}{KN-K\sigma} \right\rfloor \mod (N + 1) \right).$$

The above scheme is similar to the scheme used for the identity function. In this case, every node sends a number of the form $10 \ldots 0$ in base $N + 1$. If we compute the output of the network in base $N + 1$, we will never incur any carry bits, since for each digit the number of 1’s summed is at most $N$. This means that if we represent the output of the network in base $N + 1$, the $i$-th digit is the number of sources that sent the message
10...0, a 1 followed by $i-1$ zeros. This gives the number of sources which observed
the value $i$ and thus the $i$-th coordinate of $f(u)$.

Using the total number of possible histograms [32] and Corollary 3.3.2, for com-
puting the histogram function we need

$$q \geq \binom{K + N - 1}{K}.$$

When $K$ is fixed and $N$ grows, the above bound implies that $q = \Omega(N(N + 1)^{K-1})$, which shows that the proposed scheme is within a constant factor from the
minimum required $q$.

**Maximum function** In this section, we are going to study the maximum function
which can be derived from the histogram. We will show that despite the fact that this
function has a much smaller range than the histogram, computing the maximum over the
sum channel requires $q$ to be as big (up to a constant factor) as that for the histogram.
Let $\mathcal{A} = \{1, \ldots, K\}$. The maximum function is defined as

$$f(u) = \begin{cases} \max_{u_i \neq \phi} u_i \quad &\exists u_i \neq \phi \\ 0 &\text{otherwise} \end{cases}$$

A scheme to compute the maximum function when $q \geq N^{K+1-1}N^{-1}$ is as follows:

- **Encoding** $x_i = c(u_i) = \sum_{j=0}^{u_i-1} N^j = \frac{N^{u_i}-1}{N-1}$
- **Decoding** $f(u) = \sum_{i=1}^{K} i \cdot 1_{\{c(k) \leq y \leq c(k+1)\}}$.

To understand why the above scheme works, first observe that

$$c(i+1) - N \cdot c(i) = \sum_{j=0}^{(i+1)-1} N^j - N \cdot \sum_{j=0}^{i-1} N^j = 1.$$ 

This implies that $N \cdot c(i) \leq c(i+1)$. Let $u_{\max} = \max_i u_i$ and $S = \{i \mid u_i = u_{\max}\}$. Then

$$y = \sum_{i=1}^{N} c(u_i) = |S| \cdot c(u_{\max}) + \sum_{i \not\in S} c(u_i).$$
This implies that

\[ y \leq |S| \cdot c(u_{\max}) + (N - |S|) \cdot c(u_{\max}) \]
\[ \leq N \cdot c(u_{\max}) \leq c(u_{\max} + 1). \]

Further,

\[ y \geq |S| \cdot c(u_{\max}) \geq c(u_{\max}). \]

We will now prove a lower bound on the field size \( q \) required to compute the maximum function over the sum channel. Assume that all sources are active. Let \( x_i^j \) be the symbol sent by source \( \sigma_i \) when it observes \( j \in \mathcal{A} \).

Let \( S \subseteq \mathcal{A}^N \) be the collection of all vectors \( w \) such that

\[ w_i \geq w_{i+1} \text{ for every } i \in \{1, 2, \ldots, N - 1\}. \quad (3.3) \]

Let \( u, v \in S \) be two distinct vectors. Consider the maximal set of indices \( I \subseteq \{1, \ldots, N\} \) such that for each \( i \in I \), we have \( u_i = v_i \). Let \( \hat{u} \) (\( \hat{v} \) respectively) be the input vector generated from \( u \) (\( v \) respectively) by setting all components in \( I \) to 0 and retaining all others.

From (3.3), there exists \( j \geq 1 \) such that

\[ u_i = v_i \text{ for every } i \in \{1, 2, \ldots, j - 1\}, u_j \neq v_j, \]
\[ f(\hat{u}) = u_j, f(\hat{v}) = v_j \implies f(\hat{u}) \neq f(\hat{v}). \]

Thus the sink should receive different symbols for the input vectors \( \hat{u}, \hat{v} \), and hence we have

\[ \sum_{i \in I} x_i^0 + \sum_{j \notin I} x_j^{u_j} \neq \sum_{i \in I} x_i^0 + \sum_{j \notin I} x_j^{v_j} \implies \sum_{j \notin I} x_j^{u_j} \neq \sum_{j \notin I} x_j^{v_j} \]
\[ \implies \sum_{i \in I} x_i^{u_i} + \sum_{j \notin I} x_j^{u_j} \neq \sum_{i \in I} x_i^{v_i} + \sum_{j \notin I} x_j^{v_j} \]

where the last inequality follows since for each \( i \in I \), \( u_i = v_i \). Since \( u, v \in S \) are arbitrary, this implies that the sink should receive a distinct symbol for every input vector.
The number of vectors in $S$ is equal to the number of binary sequences of length $N + K - 1$ with $N$ zeroes and the rest all one. Thus we have $|S| = \binom{N+K-1}{K-1}$. Since the sink should receive a different symbol for every distinct input vector in $S$ and the size of the field is $q$, to successfully compute the maximum function we need $q \geq \binom{N+K-1}{K-1}$ which is same as the lower bound for the histogram function.

### 3.3.4 Computation over multiple timeslots

In the previous section, we showed that some functions cannot be implemented with only one channel use if the field size $q$ is not large enough. In this section, we will allow multiple uses of the network.

Our main observation is that computing a function using $T$ timeslots in a network with underlying field $\mathbb{F}_q$ is equivalent to computing it in just one timeslot over a network that operates with symbols of an extension field $\mathbb{F}_{q^T}$. In the previous section, we devised schemes and proved bounds for computation over networks operating over a prime field. To study computation over multiple timeslots, we need to extend those results to general finite fields. In the following we illustrate this via the example of threshold functions. We then discuss a generic strategy.

**Threshold function** Let $q = p^T$ for some prime $p$ and let $f$ be the threshold function. Then equation (3.2) in Section 3.3.3 provides a necessary condition for computation and implies that if we create a vector $v \in \mathbb{F}_q^N$ such that the $i$-th component $v_i = x_i^0 - x_i^1$, then we have that for any $I \subseteq \{1, 2, \ldots, N\}$,

$$\sum_{i \in I} v_i \neq 0 \quad (3.4)$$

where the sum is over the field $\mathbb{F}_q$. Since $q = p^T$, each element $v_i$ can be viewed as a vector of length $T$ over $\mathbb{F}_p$. Let $v_i = (v_{i1}, v_{i2}, \ldots, v_{iT})$. Given characteristic $p$, we want to find the minimum $T$ such that (3.4) holds. It can be easily checked that $T = \lceil N/(p - 1) \rceil$ suffices for any $p$. It remains to find if this is optimal. For any $p$, consider the following system of $T$ polynomial equations in $N$ variables, with each
polynomial in $\mathbb{F}_p[x_1, x_2, \ldots, x_N]$:

$$\sum_{i=1}^{N} v_{ij} \cdot x_i^{p-1} = 0 \quad \forall j \in \{1, ..., T\}.$$  

It can be verified that if there exists any non-trivial solution to the above system of polynomial equations, then it will imply that the vectors $v_1, v_2, \ldots, v_N$ violate (3.4). From the Chevalley-Warning theorem [33, Theorem 3, Pg. 5], there exists a non-trivial solution if $T < N/(p-1)$. Thus, the minimum $T$ such that the threshold target function $f$ can be computed in a network operating over the finite field of size $q = p^T$ is equal to $\lceil N/(p-1) \rceil$.

**Generic upper bound** Let $B$ be the sum channel. We can implement the channel $y' = C \cdot x'$, where $C$ is any desired $MC \times N$ matrix, and $y'$ is the vector that the sink collects after $T = MC$ timeslots, with the following scheme:

- **Encoding** $x_i[t] = (C)_{t,i} \cdot x'_i$
- **Decoding** $f(u) = (y[1] \ldots y[MC])$.

If we select $C$ to be any $N \times N$ invertible matrix (i.e., $T = N$), we obtain a “virtual“ rate-preserving network. This means that,

$$T_c(f, B) \leq N \cdot T_c(f, C)$$

where $T_c(f, C)$ can be upper bounded as described in Proposition 3.3.4.

Moreover, if we have any information in advance about the number of active sources, we can select an appropriate matrix $C$ that allows to reconstruct the symbols $x'$, potentially using $T \ll N$. This is the case in compressed sensing: if we know for example that the vector $x'$ is sparse (few sources are active), we can select a “good“ matrix $C$, so that we can efficiently recover $x'$ [34]. Thus our formulation provides a novel application of compressed sensing methods.
3.4 Non-coherent communication

We now discuss $T_{nc}(f, S)$, the minimal number of transmissions that are necessary to compute function $f$ when the channel matrix can be any of the matrices contained in $S$ and is not known by the sink and the sources. We will set $S$ to contain channel matrices of either rate-preserving or constant-rate network. We will call the minimum number of timeslots necessary to compute over rate-preserving (resp. constant-rate) networks $T^r_{nc}(f)$ (resp. $T^c_{nc}(f)$).

3.4.1 Rate preserving networks

A possible strategy for non-coherent communication over rate-preserving networks is to learn the channel $A$ by sending $d \cdot N$ probe symbols into the network from each source before the actual codewords. Assume sources send $X' = (P|X)$ where $P$ is a $N \times N$ invertible matrix formed by the probe symbols and $X$ is the codeword from a code for a fixed rate-preserving network $A'$. Let $Y = (Y_P|Y_X)$ the output of the network where $Y_P = A \cdot P$ and $Y_X = A \cdot X$. If $P$ is chosen such that it is invertible then the sink can learn $A$ by solving:

$$A = Y_P \cdot P^{-1}.$$ 

If the network is rate-preserving then the sink can reconstruct what would have been the output of the network if the channel matrix was $A'$ by computing $A' \cdot A^{-1} \cdot Y_X$. This implies that given a coherent code of length $T$ to compute a function $f$ in a rate-preserving network, we can construct a non-coherent code of length at most $T + N$.

This directly implies the following lemma:

**Lemma 3.4.1.** For every function $f$ there exists a code for computing $f$ that is $(N + \lceil \log_q |A| \rceil)$-feasible for any rate-preserving network.

As we will show later, one can often do better by designing more sophisticated schemes for the non-coherent channel.

Notice that this approach can be used to learn the channel matrix at the sink even in constant-rate networks however in that case it is not clear which code the sources should use to choose $X$. 
Justified by the discussion in Section 3.2.1, in the sequel we will only consider codes which use one-dimensional subspaces. We will denote the dimension of any subspace $\pi$ by $\dim(\pi)$. Also, for any vector $u$, the $j$-th component will be denoted by $(u)_j$. Consider a set of indices $I = (i_1, i_2, \ldots, i_{|I|}) \subseteq \{1, \ldots, N\}$. For any $a = (a_1, a_2, \ldots, a_{|I|}) \in \mathcal{A}^{|I|}$ and any vector $u \in \mathcal{A}^N$, let $u(I, a) = (u_1, u_2, \ldots, u_N)$ denote a vector which is obtained from $u$ by substituting the components corresponding to the index set $I$ with values from the vector $a$ and retaining all the other components. That is, for each $j \in \{1, \ldots, |I|\}$, $(u(I, a))_j = (a)_j$ and for each $k \notin I$, $(u)_k = (u(I, a))_k$.

We conclude this introduction with two lemmas that are often used in the subsequent sections.

**Lemma 3.4.2.** If there exist one-dimensional subspaces $\pi_1, \pi_2, \ldots, \pi_K \subseteq \mathbb{F}_q^T$ such that

$$\pi_i \not\subseteq \sum_{j<i} \pi_j \quad \forall \ i \in \{1, \ldots, K\} \quad (3.5)$$

then $T \geq K$.

**Proof.** (3.5) implies that the basis vectors for the $K$ subspaces are linearly independent. The result then follows.

**Lemma 3.4.3.** The number of subspaces of dimension $d$ in $\mathbb{F}_q^T$ is at most $4q^{d(T-d)}$ [29, Lemma 4].

**General lower bounds**

We now present two lower bounds on $T_{\text{rp}}^{\text{nc}}(f)$. We first present a lower bound for functions that satisfy a given property and then we generalize the idea to all functions.

Consider the following function property. **Function property $P$** : For each source $\sigma_k$ and any $a, b \in \mathcal{A}$, there exists $u$ such that

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

**Examples** : The identity function and arithmetic sum function satisfy property $P$. We have the following simple lower bound.
Lemma 3.4.4. Given a network $A$ for any target function $f$ which satisfies property $P$,\[ T_{nc}^{rp}(f) \geq \log_q \frac{|A|}{4}. \]

Proof. For any $T$ feasible code for computing $f$, each source must assign a distinct $d$-dimensional subspace to each $a \in A$. From Lemma 3.4.3, we have\[ 4q^{d(T-d)} \geq |A| \Rightarrow d \cdot T \geq \log_q \frac{|A|}{4}. \]

Now consider the following general lemma.

Lemma 3.4.5. Let $\pi \subseteq \mathbb{F}_q^T$ be a subspace of dimension $d_1$. Let $\pi_1, \pi_2, \ldots, \pi_K \subseteq \mathbb{F}_q^T$ be $d_2$-dimensional subspaces such that for every $i \neq j$, $\pi + \pi_i \neq \pi + \pi_j$. Then,\[ T \geq \max \left\{ \sqrt[3]{\log_q (K-1)}, \frac{\log_q (K-1)}{3d_2} \right\}. \]

Proof. Denote the complement subspace of $\pi$ by $\overline{\pi}$ ($\pi \cap \overline{\pi} = \emptyset$, $\pi + \overline{\pi} = \mathbb{F}_q^T$). Let $< b_1, \ldots, b_{d_1} >$ be a basis of $\pi$ and $< b_{d_1+1}, \ldots, b_T >$ be a basis of $\overline{\pi}$ so that together they span $\mathbb{F}_q^T$. Now let $< c_1, \ldots, c_{d_2} >$ denote the basis for any subspace $\pi_i$. Then each $c_i$ can be expressed as a linear combination of the $b_i$’s, that is, $c_i = \alpha_{i,1} b_1 + \ldots + \alpha_{i,d_1} b_1$. Thus, $\pi + \pi_i$ is a subspace spanned by $< b_1, \ldots, b_{d_1}, \sum_{i=1}^{T} \alpha_{i,1} b_i, \ldots, \sum_{i=1}^{T} \alpha_{i,d_1} b_1 >$. This is identical to the subspace spanned by $< b_1, \ldots, b_{d_1}, \sum_{i=d_1+1}^{T} \alpha_{i,1} b_i, \ldots, \sum_{i=d_1+1}^{T} \alpha_{i,d_1} b_1 >$, where the last $d$ vectors are a linear combination of vectors in $\overline{\pi}$. Therefore for each subspace $\pi_i$, there exists a subspace $\overline{\pi}_i \subseteq \pi$ such that $\pi + \pi_i = \pi + \overline{\pi}_i$ and $\overline{\pi}_i \cap \pi = \emptyset$. Then for every $i \neq j$, $\overline{\pi}_i \neq \overline{\pi}_j$ since $\pi + \overline{\pi}_i \neq \pi + \overline{\pi}_j$. Further, each $\overline{\pi}_i$ has dimension at most $d_2$. Note that the dimension of $\pi$ is $T - d_1$ and each subspace $\overline{\pi}_i$ is a subspace of
Π. Since there are $K$ distinct $\pi_i$'s, we have from Lemma 3.4.3 that

$$1 + 4 \cdot \min\{T-d_1, d_2\} \sum_{j=1}^q q_j(T-d_1-j) \geq K.$$  \hspace{1cm} (3.6)

Then, we have

$$4 \cdot \sum_{j=1}^{T-d_1} q_j(T-d_1-j) \geq K - 1$$

$$\Rightarrow 4(T - d_1) q(T-d_1)^2 \geq K - 1$$

$$\Rightarrow \log_q(4(T - d_1)) + \left(\frac{T - d_1}{2}\right) \geq \log_q(K - 1).$$

Since $\log_q(4(T - d_1)) \leq 2(T - d_1)^2$, we have

$$3(T - d_1)^2 \geq \log_q(K - 1)$$

$$\Rightarrow T \geq \frac{\sqrt{\log_q(K - 1)}}{3}.$$  

From (3.6), we also have

$$4 \cdot \sum_{j=1}^{d_2} q_j(T-d_1-j) \geq K - 1$$

$$\Rightarrow 4d_2 \cdot q(T-d_1-\hat{d}) \geq K - 1 \text{ with } \hat{d} = \arg\max_{j \in \{1, d_2\}} q_j(T-d_1-j)$$

$$\Rightarrow \log_q(4d_2) + \hat{d}(T - d_1 - \hat{d}) + \geq \log_q(K - 1).$$

Since $\log_q(4d_2) \leq 2d_2$ and $\hat{d} \leq d_2$, we have

$$2d_2T + d_2T \geq \log_q(K - 1)$$

$$\Rightarrow T \geq \frac{\log_q(K - 1)}{3d_2}.$$
For any \( u \in \tilde{A}^N \) and \( I \subseteq \{1, 2, \ldots, N\} \), let

\[
R^u_I(f) = \left| \left\{ f(u(I), a) : a \in \tilde{A}^{|I|} \right\} \right|
\]

(3.7)
denote the number of distinct values that the function takes when only the arguments corresponding to \( I \) are varied and all the others are held fixed according to \( u \). Also, for any \( T \) code, any input vector \( u \in \tilde{A}^N \) and \( I \subseteq \{1, 2, \ldots, N\} \), let

\[
\Pi^L_u = \sum_{i \in I} \pi_i^u.
\]

**Lemma 3.4.6.** Given a network \( A \) for any target function \( f \) we have,

\[
T_{nc}^{wp}(f) \geq \max_{I, u, \pi} \max_{R^u_I(f) > 1} \left\{ \frac{\sqrt{\log_q (R^u_I(f) - 1)}}{3}, \frac{\log_q (R^u_I(f) - 1)}{3 |I|} \right\}.
\]

**Proof.** Consider any \( I \subseteq \{1, 2, \ldots, N\} \) and any input vector \( u \). For any \( a, b \in (A \cup \{\phi\})^{|I|} \), if \( f(u(I), a) \neq f(u(I), b) \), then any \( T \) feasible code should satisfy the following condition.

\[
\sum_{j \in \{1, \ldots, |I|\}} \pi_{a_j} + \sum_{i \in I^c} \pi_i^u = \sum_{j \in \{1, \ldots, |I|\}} \pi_{b_j} + \sum_{i \in I^c} \pi_i^u \Rightarrow \Pi^L_{u(I,a)} + \Pi^L_{u(I,b)} \neq \Pi^L_{u(I,a)} + \Pi^L_{u(I,b)}.
\]

(3.8)

Note that for any \( I \) and \( a \in (A \cup \{\phi\})^{|I|} \), \( \dim \left( \Pi^L_{u(I,a)} \right) \leq d \cdot |I| \) since it is composed of the union of at most \( |I| \) \( d \)-dimensional subspaces. Then, (3.8) and (3.7) imply that there exist \( R^u_I(f) \) subspaces, each with dimension at most \( d \cdot |I| \), such that the union of any one of them with \( \Pi^L_{u(I,a)} \) is unique. Since \( I, u \) were arbitrary, the result follows from Lemma 3.4.5.

**Example 3.4.7.**
• For the identity target function $f$, the above bound gives

$$T_{nc}^{rp}(f) \geq \frac{\log_q |A|}{3}.$$ 

• For the arithmetic sum target function $f$, we get

$$T_{nc}^{rp}(f) \geq \sqrt{\frac{\log_q N |A|}{3}}.$$ 

**Functions which are maximally hard to compute**

Consider the case $N \geq \log_q |A|$. Next, we present a class of functions for which the cost is required to grow linearly with respect to the number of sources $N$. Thus, the number of transmissions that each source makes for the computation of such functions is almost the same (in the order sense) as the upper bound given at the beginning of the this section. For any vector $u \in \tilde{A}^N$, let $I_u$ denote the index set corresponding to the components which are not $\phi$. Then, consider a target function $f$ which satisfies the following property with some constant $\alpha \in (0, 1]$.

**Function property $P(\alpha)$:** There exists a vector $u^* = (u^*_1, u^*_2, \ldots, u^*_N)$ with $|I_{u^*}| \geq \alpha N$ such that for each $k \in I_{u^*}$,

$$f(u^*(\{k\}, \phi)) \neq f(u^*).$$

(3.9)

Recall that $u^*(\{k\}, \phi)$ denotes the vector obtained from $u^*$ by setting $u^*_k$ equal to $\phi$ and retaining all the other components. This implies that the function value is sensitive to whether any specific source $\sigma_k$ is active or not.

**Example 3.4.8.**

• The identity function satisfies property $P(1)$ by choosing each $u^*_i$ equal to any element of the alphabet $A$.

• The arithmetic sum function satisfies property $P(1)$ by choosing each $u^*_i$ equal to some non-zero element of the alphabet $A$. 
• The parity function \((A = \{0, 1\})\) satisfies property \(P(1)\) by choosing each \(u_i^*\) equal to 1.

**Lemma 3.4.9.** Let \(f\) be a function which satisfies the property \(P(\alpha)\). Then, \(T_{nc}^{rp}(f) \geq \alpha N\).

**Proof.** From (3.9), any feasible code for computing \(f\) must satisfy the following condition. For each \(k \in I_{u^*},\)

\[
\pi_k u_k^* + \sum_{j \neq k} \pi_j u_j^* \neq \sum_{j \neq k} \pi_j u_j^* \implies \pi_k u_k^* \not\subseteq \sum_{j \neq k} \pi_j u_j^* .
\]

Since \(|I_{u^*}| \geq \alpha N\), the result follows from\(^4\) Lemma 3.4.2. \(\blacksquare\)

**Comment :** Lemma 3.4.6 provides a general lower bound on \(T_{nc}^{rp}(f)\) for arbitrary functions. Functions for which the lower bound is of the same order as \(N + \lceil \log_q |A| \rceil\) are also maximally hard to compute.

**Bounds for specific functions**

**k-threshold Function** Let \(A = \{1\}\). The \(k\)-threshold function is defined as\(^5\)

\[
f(u_1, u_2, \ldots, u_N) = \begin{cases} 
1 & \text{if } u_1 + u_2 + \ldots + u_N \geq k \\
0 & \text{otherwise.}
\end{cases}
\]

**Lemma 3.4.10.** Given a rate-preserving network \(A\) there exists a \(T\) feasible code for computing the \(k\)-threshold function with \(k < N/2\), such that

\[
T \leq NH_q \left( \frac{k}{2N} \right)
\]

\(^4\)From Lemma 3.4.2, the result can be proven for a relaxed albeit more complicated version of the property \(P(\alpha)\).

\(^5\)For any integer \(a\), we set \(a + \phi = a\). Thus, the function computes whether the number of active sources is at least \(k\) or not.
Let $H$ be the $l \times N$ parity check matrix of a binary code with minimum distance $d_{\text{min}} = k + 1$.

- Source $\sigma_i$ uses $C_i = \{h_i\}$, where $h_i$ is a column of $H$.
- If the dimension of the subspace that the sink receives is less than $k$, it outputs 0. Otherwise, it outputs 1.

**Figure 3.2:** A $T$-admissible code for the $k$-threshold function for non-coherent rate-preserving networks.

where $H_q$ is the $q$-ary entropy function defined as

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

**Proof.** Consider the scheme in Figure 3.2. The scheme uses a $T \times N$ parity check matrix of a binary code with minimum distance $d_{\text{min}} = k + 1$. From [35, 36], there exists such a matrix with

$$T \leq NH_q \left( \frac{k}{2N} \right).$$

$\blacksquare$

**Comment:** For a constant $k$, $O \left( NH_q \left( \frac{k}{2N} \right) \right) = O \left( k \log_q N \right)$. Thus, while computing the identity function requires the cost to grow linearly with the number of sources $N$, the $k$-threshold function requires only logarithmic growth.

**Lemma 3.4.11.** For the $k$-threshold function $f$ with $k < N/2$,

$$T_{\text{nc}}^p(f) \geq \frac{N}{2} H_q \left( \frac{k}{2N} \right).$$

**Proof.** Consider two possible input vectors $(u_1, u_2, \ldots, u_N)$ and $(v_1, v_2, \ldots, v_N)$ such that

- $u_i = 1 \ \forall \ i \in \{1, 2, \ldots, k\}$ and $u_i = \phi$ otherwise
- $v_i = 1 \ \forall \ i \in \{2, 3, \ldots, k\}$ and $v_i = \phi$ otherwise.
Note that
\[ 1 = f(u_1, u_2, \ldots, u_N) \neq f(v_1, v_2, \ldots, v_N) = 0 \]
and hence it is necessary for any feasible code for computing \( f \) that
\[ \pi_1^1 + \sum_{i=2}^{k} \pi_i^1 \neq \sum_{i=2}^{k} \pi_i^1 \implies \pi_1^1 \not\subseteq \sum_{i=2}^{k} \pi_i^1. \]

The same argument can be extended to get the following necessary condition. For any subset \((i_1, i_2, \ldots, i_k)\) of \(\{1, \ldots, N\}\),
\[ \pi_{i,j}^1 \not\subseteq \sum_{i \neq j} \pi_{i,l}^1 \text{ for every } j \in \{1, \ldots, k\}. \]

Denote a basis vector for \(\pi_i^1\) by \(v_i\). From the necessary condition on the subspaces \(\pi_1^1, \pi_2^1, \ldots, \pi_N^1\), any collection of \(k\) vectors from \(v_1, v_2, \ldots, v_N\) are linearly independent. The \(T \times N\) matrix with the vectors \(v_1, v_2, \ldots, v_N\) as columns corresponds to the parity check matrix for a linear code of length \(N\) and minimum distance at least \(k + 1\). Using the bounds in [35, 36], for \(k < N/2\) we have
\[ T \geq N H_q \left( \frac{k}{2N} \right) - \frac{1}{2} \log_q \left( 4k \left( 1 - \frac{k}{2N} \right) \right). \]
The result then follows since
\[ \frac{1}{2} \log_q \left( 4k \left( 1 - \frac{k}{2N} \right) \right) \leq \frac{N}{2} H_q \left( \frac{k}{2N} \right). \]
Proof of the above inequality can be found in [37].

**Maximum Function**

**Lemma 3.4.12.** Given a rate-preserving network \(A\) there exists a \(T\) feasible code for computing the maximum function such that
\[ T \leq \min \left\{ |A|, N + \lceil \log_q |A| \rceil \right\}. \]
Proof. Consider the following two schemes for computing the maximum function$^6$.

- **$A(1, |\mathcal{A}|)$ scheme**: Let $v_1, v_2, \ldots, v_{|\mathcal{A}|}$ be linearly independent vectors of length $|\mathcal{A}|$ each. For every source $\sigma_i$, let $C_i = (v_1, v_2, \ldots, v_{|\mathcal{A}|})$. This scheme has $T = |\mathcal{A}|$.

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

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

**Lemma 3.4.13.** For the maximum target function $f$,

$$T_{\text{nc}}^{\text{rp}}(f) \geq \min\{|\mathcal{A}|, N\}.$$ 

Proof. Let $\mathcal{A} = (a_1, a_2, \ldots, a_{|\mathcal{A}|})$ be an ordered set (in increasing order) and let $M = \min\{N, |\mathcal{A}|\}$. Consider two possible input vectors $(u_1, u_2, \ldots, u_N)$ and $(v_1, v_2, \ldots, v_N)$ such that

$$u_i = a_i \forall i \in \{1, \ldots, M\} \text{ and } u_i = \phi \text{ otherwise}$$

$$v_i = a_i \forall i \in \{1, \ldots, M - 1\} \text{ and } v_i = \phi \text{ otherwise}.$$ 

Note that

$$M = f(u_1, u_2, \ldots, u_N) \neq f(v_1, v_2, \ldots, v_N) = M - 1$$

and hence any feasible code for computing $f$ must satisfy

$$\sum_{i=1}^{M-1} \pi_i^a_i + \pi_M^a_M \neq \sum_{i=1}^{M-1} \pi_i^a_i \implies \pi_M^a_M \not\subseteq \sum_{i=1}^{M-1} \pi_i^a_i.$$ 

The same argument can be extended to get the following necessary condition. For any subset $(i_1, i_2, \ldots, i_M)$ of $\{1, \ldots, N\}$ and any ordered subset (in increasing order) $(a_{i_1}, a_{i_2}, \ldots, a_{i_M})$ of $\mathcal{A}$,

$$\pi_{i_k}^{a_{i_k}} \not\subseteq \sum_{m < k} \pi_{i_m}^{a_{i_m}}.$$ 

$^6$For any $a \in \mathcal{A}$, we set $\max\{a, \phi\} = a.$
Let $H$ be the $(T/|\mathcal{A}|) \times N$ parity check matrix of a binary code with minimum distance $K + 1$. 

- If source $\sigma_i$ takes value $a_j$ from the alphabet $\mathcal{A}$, then it transmits a vector which is all zero except the $(j - 1) \times (T/|\mathcal{A}|) + 1$ to $j \times (T/|\mathcal{A}|)$ elements, which take values from the $i$-th column of $H$.
- Each vector in the union subspace $\Pi$ that the sink receives is parsed into $|\mathcal{A}|$ sub-vectors of length $T/|\mathcal{A}|$.
- Let $\Pi_j \subseteq \mathbb{F}_q^{T/|\mathcal{A}|}$ denote the subspace spanned by collecting the $j$-th sub-vector of each vector in $\Pi$.
- Thus by calculating $\dim(\Pi_{|\mathcal{A}|})$, $\dim(\Pi_{|\mathcal{A}|-1})$ . . . , the sink can compute the $K$ largest values.

**Figure 3.3:** A $T$-admissible code for the $K$-largest values function for non-coherent rate-preserving networks.

Then the result follows from Lemma 3.4.2.

**Lemma 3.4.14.** Given a rate-preserving network $A$ there exists a $T$ feasible code for computing the $K$-largest values function with $K < N/2$, such that

$$T \leq |\mathcal{A}| \cdot NH_q \left( \frac{K}{2N} \right).$$

**Proof.** Consider the scheme in Figure 3.3.
Again from [35, 36], there exists a parity check matrix such that
\[
\frac{l}{|\mathcal{A}|} \leq NH_q \left( \frac{K}{2N} \right) .
\]

\[\blacksquare\]

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

\textbf{Lemma 3.4.15.} For the $K$-largest values target function $f$ with $K < N/2$,
\[
T_{\text{nc}}^{\text{rp}}(f) \geq \frac{N}{2} H_q \left( \frac{K}{2N} \right).
\]

\textit{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 3.4.11. \[\blacksquare\]

\textbf{Arbitrary Functions when all sources are active}

We now present a general method to compute functions over a non-coherent rate-preserving network that works if all sources are always active. We will illustrate the method for boolean functions of the form $f : \mathcal{A}^N \to \{0, 1\}$. 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
\[
f(u_1, u_2, \ldots, u_N) = \sum_{s=1}^{s} \prod_{j=1}^{N} B_{ij}
\]
where $s$ is some integer such that $1 \leq s \leq |\mathcal{A}|^N$; $\{B_{ij}\}$ are boolean variables such that the value of $B_{ij}$ depends only on $u_j$; and the sum and product represent boolean OR and
AND. By taking the complement, we have

\[ f(u_1, u_2, \ldots, u_N) = \prod_{i=1}^{s} \sum_{j=1}^{N} B_{ij}. \]

Given any input \( u_j \), source \( j \) creates a vector \( v_j \) of length \( s \) such that \( i \)-th component is \( B_{ij} \). 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

\[ A_i = \sum_{j=1}^{N} B_{ij} \]

and hence,

\[ f(u_1, u_2, \ldots, u_N) = \prod_{i=1}^{s} A_i. \]

Thus, we have a \( s \)-feasible scheme to compute any function \( f \) with binary output.

This scheme always work if all the sources are active. If some of the sources are not active this scheme works if for every \( i \) the condition \( B_{ij} = 1 \) when \( u_j = \phi \).

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 3.4.16. Let \( \mathcal{B} = \{1, \ldots, K\} \) 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

\[ f(u_1, u_2, \ldots, u_N) = \begin{cases} 1 & \text{if } \mathcal{B} \nsubseteq \bigcup_{i=1}^{N} u_i \\ 0 & \text{otherwise.} \end{cases} \]

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

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

Then the function $f$ can be rewritten as

$$f(u_1, u_2, \ldots, u_N) = \sum_{i=1}^{K} \prod_{j=1}^{N} 1_{\{i \notin u_j\}}.$$  

Then using the scheme described in this section, the set cover function can be computed using a $K$-feasible code. This scheme is in-fact optimal in terms of the smallest possible cost for any feasible code.

### 3.4.2 Constant rate networks

It is easy to obtain an upper bound for $T_{nc}(f)$ for constant-rate network. Let $\{c_i\}$ a coloring of the source messages as described in Section 3.3.2. We can immediately derive the following upper bound:

**Proposition 3.4.17.** Given a constant-rate network $A$ there exists a $T$ feasible code for computing a function $f$ such that

$$T = N + \sum_{i=1}^{N} \lceil \log_q \left( |\{c_i(a) : a \in A\}| \right) \rceil.$$  

Such bound corresponds to a code where each source transmits one by one the color of its message and a pilot symbol to let the sink learn the channel matrix.

We now show that provided that the field size $q$ is sufficiently large the lower bounds derived in Section 3.4.1 holds also in constant-rate networks. To do so it is sufficient to prove that with an appropriate choice of $q$ it is not possible to distinguish what was sent if the sources send two sets of vectors that span the same subspace. In rate-preserving networks this was trivial to prove, in rate-preserving networks the sent vectors can only be combined with positive coefficients and therefore not all vectors in
the span of the input vectors can be received.

For general $q$ this condition does not hold. For instance assume that $q = 2$, $d = 1$ and $N = 3$. If the sources send $A = \{(100), (010), (001)\}$ the only output of the network is $(111)$. If the sources send $B = \{(110), (010), (001)\}$, which spans the same subspace as $A$, the only possible network output is $(101)$ so the sink can distinguish between the two set of vectors.

We now show that the if $q$ is sufficiently large this does not happen:

**Proposition 3.4.18.** Given two sets $A$ and $B$ of vectors from $\mathbb{F}_q^T$, such that $\text{span}(A) = \text{span}(B) = \Pi$ with $K = \dim(\Pi)$. Let $C_S$ (for $S = A, B$) the set of all the linear combinations of vectors of the form $\sum_{x \in S} c_x \cdot x$ such that $c_x \neq 0$ for every $x \in S$. If $q > K/\log(2) + 1$ then $C_A \cap C_B \neq 0$.

**Proof.** We know that $C_A \subset \Pi$ and $C_B \subset \Pi$. If we can prove that $|C_A| + |C_B| > |\Pi|$ we will have proved that $C_A \cap C_B \neq \emptyset$. To prove it we observe that $|C_S| \geq (q - 1)^K$ for $S = A, B$. This is the case because there are at least $K$ linearly independent vectors in $S$ and those can be combined with at least $q - 1$ coefficients to obtain distinct vectors (the other $|S| - K$ vectors can be combined with fixed coefficients). Therefore we know that:

$$|C_A| + |C_B| \geq 2(q - 1)^K$$

If we can prove that this is bigger than $|\Pi| = q^K$ we are done. This can be done as follows:

$$2(q - 1)^K > q^K \iff q - 1 > 2^{-1/K}q \iff q > \frac{1}{1 - 2^{-1/K}} = \frac{\sqrt[4]{2}}{\sqrt[4]{2} - 1}$$

We can now prove that the previous condition holds as follows:

$$\frac{\sqrt[4]{2}}{\sqrt[4]{2} - 1} \leq K/\ln(2) + 1 \iff \frac{1}{\sqrt[4]{2} - 1} \leq K/\ln(2) \iff \ln(2)/K \leq \sqrt[4]{2} - 1$$

observe that for $K = 1$ the inequality holds. To prove that it always hold we can take the derivative of both sides of the inequality and observe that the lhs is always decreasing faster than the rhs.

$\blacksquare$
Since we know that any set of sent vectors has maximal cardinality $d \cdot N$ we can easily obtain a condition on $q$ that makes the above condition hold. Under such condition all derivations in 3.4.1 hold.

### 3.5 Conclusions

In this chapter we investigated function computation in a network where the intermediate node operation result in a fixed linear transformation of the source data. We considered the question whether we can calculate different functions of the source data at the sink efficiently without altering the relays operation. We focused our attention on two classes of linear transformations that model widely used network protocols.

We both considered the case in which the linear transformation is not known, where we proposed appropriate subspace codes for the source nodes and the case in which the linear transformation is known, where we proposed vector codes. For both cases we calculated upper and lower bounds on the required number of channel uses $T$ and we derived optimal code designs for some common functions.

### 3.6 Appendix

**Proposition 3.6.1.** Consider two rate-preserving networks: $\mathcal{N}$ with source degree $d$ and channel matrix $A$, and $\mathcal{N}'$ with source degree 1 and channel matrix $A'$. If the network $\mathcal{N}$ admits a coherent code $\{C_i\}$ of length $T$, then the network $\mathcal{N}'$ admits a coherent code $\{C'_i\}$ of length $dT$.

*Proof.* This follows simply from the observation that under coherent communication in a rate-preserving network, the sink can reconstruct all symbols sent by the sources by multiplying the network output by the inverse of the channel matrix. Thus the code of length $T$ for the network $\mathcal{N}$ can be simulated over the network $\mathcal{N}'$ by a code of length $dT$. ■

**Proposition 3.6.2.** Consider two rate-preserving networks: $\mathcal{N}$ with source degree $d$ and channel matrix $A$, and $\mathcal{N}'$ with source degree 1 and channel matrix $A'$. If the network
\( \mathcal{N} \) admits a non-coherent code \( \{C_i\} \) of length \( T \), then the network \( \mathcal{N}' \) admits a non-coherent code \( \{C'_i\} \) of length \(dT\).

**Proof.** For any given source message vector \((u_1, u_2, \ldots, u_N)\), let the matrix \( X \) of size \(dN \times T\) denote the input into the network \( \mathcal{N} \) using the code \( \{C_i\} \). Let \( X_i \) be the \( N \times T \) matrix of the symbols sent by the sources on their \( i \)-th out-going edge, i.e. \((X_i)_{j,k}\) is the symbol sent by source \( \sigma_j \) at time \( k \) on its \( i \)-th edge.

Then the code \( \{C'_i\} \) is constructed such that for the same source message vector, the input into the network \( \mathcal{N}' \) is given by \( X' = (X_1, \ldots, X_d) \) of size \( N \times dT \).

Recall that for non-coherent communication, the sink relies solely on the received subspace for decoding. Thus, we need to show that the subspace spanned by the rows of the matrix \( Y = AX \) received in \( \mathcal{N} \) can be reconstructed from the rows of \( Y' = A'X' \) received in \( \mathcal{N}' \). Let \( Y'_1, \ldots, Y'_d \) be the \( N \times T \) submatrices of \( Y' \), such that each \( Y'_i = A'X_i \). Since \( A \) and \( A' \) are full rank we have:

\[
\text{span}(Y) = \text{span}(A \cdot X) = \text{span}(X) = \sum_{i=1}^{d} \text{span}(X_i) = \sum_{i=1}^{d} \text{span}(A'X_i) = \sum_{i=1}^{d} \text{span}(Y'_i)
\]

\[\blacksquare\]

**Proposition 3.6.3.** Consider two constant-rate networks: \( \mathcal{N} \) with source degree \( d \) and channel matrix \( A \), and \( \mathcal{N}' \) with source degree \( 1 \) and channel matrix \( A' \). If the network \( \mathcal{N} \) admits a coherent code \( \{C_i\} \) of length \( T \), then the network \( \mathcal{N}' \) admits a coherent code \( \{C'_i\} \) of length \( T \).

**Proof.** For any given source message vector \((u_1, u_2, \ldots, u_N)\), let the matrix \( X \) of size \(dN \times T\) denote the input into the network \( \mathcal{N} \) using the code \( \{C_i\} \). Let the rows of \( X \) be arranged such that rows \( d(i-1) + 1 \) to \( di \) correspond to the symbols transmitted by
source $\sigma_i$. Then the output $Y$ received by the sink is given by

$$(Y)_{1,i} = \sum_{j=1}^{dN} (A)_{1,j} \cdot (X)_{j,i}, \text{ for each } i \in \{1, 2, \ldots, T\}.$$  

The code $\{C'_i\}$ is constructed such that for the same source message vector $(u_1, \ldots, u_N)$, the input matrix $X'$ into the network $\mathcal{N}'$ is given by

$$(X')_{j,i} = (A')_{1,j}^{-1} \cdot \sum_{k=d(j-1)+1}^{dj} (A)_{i,k} \cdot (X)_{k,i}, \text{ for each } i \in \{1, 2, \ldots, T\}, j \in \{1, 2, \ldots, N\}.$$  

It is then easy to see that for any given source message vector,

$$Y' = A'X' = AX = Y$$  

and hence the result follows. $\blacksquare$

**Proposition 3.6.4.** Consider two constant-rate networks: $\mathcal{N}$ with source degree $d$ and channel matrix $A$, and $\mathcal{N}'$ with source degree 1 and channel matrix $A'$. If the network $\mathcal{N}$ admits a non-coherent code $\{C_i\}$ of length $T$, then the network $\mathcal{N}'$ admits a non-coherent code $\{C'_i\}$ of length $T$.

**Proof.** For any given source message vector $(u_1, u_2, \ldots, u_N)$, let the matrix $X$ of size $dN \times T$ denote the input into the network $\mathcal{N}$ using the code $\{C_i\}$. Let the rows of $X$ be arranged such that rows $d(i-1)+1$ to $di$ correspond to the symbols transmitted by source $\sigma_i$.

The code $\{C'_i\}$ is constructed such that for the same source message vector $(u_1, \ldots, u_N)$, the input matrix $X'$ into the network $\mathcal{N}'$ is given by

$$(X')_{j,i} = \sum_{k=d(j-1)+1}^{dj} (X)_{k,i}, \text{ for each } i \in \{1, 2, \ldots, T\}, j \in \{1, 2, \ldots, N\}.$$
It is easy to verify that $X' = BX$ where the matrix $B$ is defined by

$$(B)_{i,j} = \begin{cases} 
1 & \text{if } d(i-1) + 1 \leq j \leq d_i \\
0 & \text{otherwise}
\end{cases} \quad \text{for every } i \in \{1, 2, \ldots, N\}.$$

Then the output matrix $Y'$ received by the sink in $N'$ is given by

$$Y' = A' \cdot X' = A'B \cdot X.$$ 

Note that the matrix $A'B$ is a possible channel matrix of the network $N'$ and since $\{C_i\}$ is an admissible non-coherent code, the sink must be able to reconstruct the function from $Y'$. This completes the proof. 

Chapter 3, in part, has been submitted for publication of the material. The dissertation author was a primary investigator and author of this paper. The material also appears in [38].
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 [39, 40]. 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 [39, 41, 42]. 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 [43–48] 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, [43] 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 [49–51]. 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, 52–55]. 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 [56] have been proposed as practical schemes for information dissemination in large unreliable networks and were studied in the context of distributed computation in [56–61].

In the present chapter, our approach is somewhat (tangentially) related to the field of communication complexity [1, 62] 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 [63–66] on computing the sum (over a finite field) of source messages in networks. We note that in independent work, Kowshik and Kumar [67] 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 $G$ contains a directed path from every node in $\mathcal{V}$ to the receiver $\rho$. 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

$$f : A^s \longrightarrow B$$

(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 $v \in V - \rho$ and any out-edge $e \in E_o(v)$, an encoding function:

$$h^{(e)} : \left\{ \begin{array}{l}
\left( \prod_{\hat{e} \in \hat{E}(v)} A^n \right) \times A^k \rightarrow A^n \quad \text{if } v \text{ is a source node} \\
\prod_{\hat{e} \in \hat{E}(v)} A^n \rightarrow A^n \quad \text{otherwise}.
\end{array} \right.$$ 

(ii) A decoding function:

$$\psi : \prod_{j=1}^{|E_i(\rho)|} A^n \rightarrow B^k.$$ 

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 $\alpha$. 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 $\alpha$. Let $e_1, e_2, \ldots, e_{|E_i(\rho)|}$ 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 $\psi$ is such that for each $j \in \{1, 2, \ldots, k\}$ and for every message generator $\alpha$, we have

$$\psi\left(z_{e_1}, \ldots, z_{e_{|E_i(\rho)|}}\right)_j = f\left(\alpha(\sigma_1)_j, \ldots, \alpha(\sigma_s)_j\right).$$ 

(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 $[68, 69]$. 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.

<table>
<thead>
<tr>
<th>Target function</th>
<th>Alphabet $\mathcal{A}$</th>
<th>$f(x_1, \ldots, x_s)$</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>arbitrary</td>
<td>$(x_1, \ldots, x_s)$</td>
<td></td>
</tr>
<tr>
<td>arithmetic sum</td>
<td>${0, \ldots, q - 1}$</td>
<td>$x_1 + \cdots + x_s$</td>
<td>‘+’ is ordinary integer addition</td>
</tr>
<tr>
<td>mod $r$ sum</td>
<td>${0, \ldots, q - 1}$</td>
<td>$x_1 \oplus \cdots \oplus x_s$</td>
<td>$\oplus$ is mod $r$ addition</td>
</tr>
<tr>
<td>histogram</td>
<td>${0, \ldots, q - 1}$</td>
<td>$(c_0, \ldots, c_{q-1})$</td>
<td>$c_i = {</td>
</tr>
<tr>
<td>linear</td>
<td>finite field</td>
<td>$a_1 x_1 + \ldots + a_s x_s$</td>
<td>arithmetic performed in the field</td>
</tr>
<tr>
<td>maximum</td>
<td>ordered set</td>
<td>max ${x_1, \ldots, x_s}$</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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 $f : \mathcal{A}^s \rightarrow \mathcal{B}$, 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 $j \not\in I$.

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

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

\(^4\)Witsenhausen [70] 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 [44–46]. 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

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

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 $a \in \mathcal{A}^t$, $a_I = x$, 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 $|g(\mathcal{A}^{|I|})| \geq R_{I,f}$, 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 $C_{\text{cod}}(\mathcal{N}, f)$ 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 $\rho$, if for each $i \in \{1, 2, \ldots, d\}$, every directed path from $\sigma_{m_i}$ to $\rho$ 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

$$I_C = \{ i : C \text{ separates } \sigma_i \text{ from the receiver} \}$$

$$R_{C,f} = R_{I_C,f}. \quad (4.2)$$

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 $v \in V$, there exists a node $u$ such that all the out-edges of $v$ are in-edges to $u$, i.e., $E_o(v) \subseteq E_i(u)$ (e.g., see Figure 4.1).

![Figure 4.1: An example of a multi-edge tree.](image)

### 4.1.2 Classes of target functions

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

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

1. $f^{\{1,\ldots,s\}} = f$
2. $|f^I (A^{|I|})| \leq |f (A^s)|$
3. For every partition $\{I_1, \ldots, I_\gamma\}$ of $I$, there exists a function $g : B_{I_1} \times \cdots \times B_{I_\gamma} \rightarrow B_I$ such that for every $x \in A^{|I|}$, we have $f^I(x) = g\left(f^{I_1}(x_{I_1}), \ldots, f^{I_\gamma}(x_{I_\gamma})\right)$.

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

Divisible functions have been studied previously 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

---

5The definitions in [3, 4] are similar to ours but slightly more restrictive.
\{I_1, \ldots, I_s\} of the source indices \{1, \ldots, s\}, the receiver \(\rho\) can evaluate the target function \(f\) by combining evaluations of \(f^{I_1}, \ldots, f^{I_s}\). 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 \(f : \mathcal{A}^s \rightarrow \mathcal{B}\) is *symmetric* if for any permutation \(\pi\) 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 \(\text{mod } r\) 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 [52].

**Definition 4.1.10.** Let \(\lambda \in (0, 1]\). A target function \(f : \mathcal{A}^s \rightarrow \mathcal{B}\) is said to be \(\lambda\)-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 \(f : \mathcal{A}^s \rightarrow \mathcal{B}\) is said to be \(\lambda\)-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 \(\text{mod } r\) 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 [71], 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 [71, 72]. 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 [41] and Vazirani [72].

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

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

  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

  $$\text{min-cut}(\mathcal{N}, f) = \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{q} ((q - 1) |I_C| + 1)}. \tag{4.5}$$
Let $A$ be an ordered set. If $f$ is the maximum target function, then

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

### 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 $\min\text{-}\text{cut}(\mathcal{N}, f)$ is an upper bound on the computing capacity $c_{\text{cod}}(\mathcal{N}, f)$. 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), $\lambda$-exponential (Theorem 4.4.4), and $\lambda$-bounded (Theorem 4.4.5) target functions. For $\lambda$-exponential target functions, the computing capacity is at least $\lambda$ times the min-cut. If every non-receiver node in a network is a source, then for $\lambda$-bounded target functions the computing capacity is at least a constant times the min-cut divided by $\lambda$. 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 $\min\text{-}\text{cut}(\mathcal{N}, f)$. 
Theorem 4.2.1. If $\mathcal{N}$ is a network with target function $f$, then

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

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 $a^{(j)} \neq b^{(j)}$, where $\equiv$ is the equivalence relation from Definition 4.1.4. Then there exist $x, y \in \mathcal{A}^n$ satisfying: $f(x) \neq f(y)$, $x_{I_C} = a^{(j)}$, $y_{I_C} = b^{(j)}$, and $x_i = y_i$ for every $i \notin I_C$.

The receiver $\rho$ can compute the target function $f$ only if, for every such pair $\{a^{(1)}, \ldots, a^{(k)}\}$ and $\{b^{(1)}, \ldots, b^{(k)}\}$ 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 $(R_{C,f})^k$ distinct vectors. Thus, we have

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

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

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

Proof. Rasala Lehman and Lehman [73, 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 $\min\text{-cut}(\mathcal{N}, f) = \min\text{-cut}(\mathcal{N})$, 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

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

Proof. The proof of this result is relegated to Section 5.5. It also follows from [65, 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

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

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

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

(4.6)
Consider any positive integers \( k, n \) such that
\[
\frac{k}{n} \leq \min_{v \in V - \rho} \frac{|E_o(v)|}{\log_{|A|} R_{I_{E_o(v)}, f}}.
\] (4.7)

Then we have
\[
|A|^{|E_o(v)|n} \geq R_{I_{E_o(v)}, f}^k \quad \text{for every node } v \in V - \rho.
\] (4.8)

We outline a \((k, n)\) solution for computing \( f \) in the multi-edge tree network \( N \). Each source \( \sigma_i \in S \) generates a message vector \( \alpha(\sigma_i) \in A^k \). Denote the vector of \( i \)-th components of the source messages by
\[
x^{(i)} = (\alpha(\sigma_1)_i, \ldots, \alpha(\sigma_s)_i).
\]

Every node \( v \in V - \{\rho\} \) sends out a unique index (as guaranteed by (4.8)) over \( A^{|E_o(v)|n} \) corresponding to the set of equivalence classes
\[
\Phi_{I_{E_o(v)}, f}(x^{(l)}_{I_{E_o(v)}}) \quad \text{for } l \in \{1, \ldots, k\}. \]
\] (4.9)

If \( v \) has no in-edges, then by assumption, it is a source node, say \( \sigma_j \). 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 \( u_1, u_2, \ldots, u_j \) be the nodes with out-edges to \( v \). For each \( i \in \{1, 2, \ldots, j\} \), using the uniqueness of the index received from \( u_i \), node \( v \) recovers the equivalence classes
\[
\Phi_{I_{E_o(u_i)}, f}(x^{(l)}_{I_{E_o(u_i)}}) \quad \text{for } l \in \{1, \ldots, k\}. \]
\] (4.10)

Furthermore, the equivalence classes in (4.9) can be identified by \( v \) from the equivalence classes in (4.10) (and \( \alpha(v) \) if \( v \) is a source node) using the fact that for a multi-edge tree network \( N \), we have a disjoint union
\[
I_{E_o(v)} = \bigcup_{i=1}^{j} I_{E_o(u_i)}.
\]

If each node \( v \) follows the above steps, then the receiver \( \rho \) can identify the equiv-
alence classes $\Phi_{I_e(v), f}(x^{(i)})$ 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

$$C_{\text{cod}}(\mathcal{N}, f) \geq \min_{v \in \mathcal{V} - \rho} \frac{|E_o(v)|}{\log_{|\mathcal{A}|} R_{I_e(v), f}}.$$  (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 of $\mathcal{N}$ to be a minimal (with respect to nodes and edges) subgraph of $G$ containing $S$ and $\rho$ such that every source in $S$ has a directed path to the receiver $\rho$. Note that every non-receiver node in a Steiner tree has exactly one out-edge. Let $T(\mathcal{N})$ denote the collection of all Steiner trees in $\mathcal{N}$. For each edge $e \in \mathcal{E}(G)$, let $J_e = \{i : t_i \in T(\mathcal{N}) \text{ and } e \in \mathcal{E}(t_i)\}$. The fractional Steiner tree packing number $\Pi(\mathcal{N})$ is defined as the linear program

$$\Pi(\mathcal{N}) = \max \sum_{t_i \in T(\mathcal{N})} u_i \text{ subject to } \begin{cases} u_i \geq 0 & \forall t_i \in T(\mathcal{N}), \\ \sum_{i \in J_e} u_i \leq 1 & \forall e \in \mathcal{E}(G). \end{cases}$$  (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 [74]).

\(^7\)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 $\sum_{i \in J} u_i \leq 1$.

**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

$$C = \bigcup_{i' \in I_{C'}} E_o(\sigma_{i'}).$$

(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 $\sigma_i$ from $\rho$ in $\mathcal{N}$, and thus $I_{C'} \subseteq I_C$.

Let $\sigma_i$ be a source such that $i \in I_C$ and let $P$ be a path from $\sigma_i$ to $\rho$ in $\mathcal{N}$. From (4.13), it follows that there exists $i' \in I_{C'}$ such that $P$ contains at least one edge in $E_o(\sigma_{i'})$. If $P$ also lies in $\mathcal{N}'$ and does not contain any edge in $C'$, then $\sigma_{i'}$ has a path to $\rho$ 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., $I_C \subseteq I_{C'}$. \[\Box\]

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

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

**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 $E_o(v)$ in $\mathcal{N}'$), we have

$$\forall v \in \mathcal{V}' - \rho, \exists C' \in \Lambda(\mathcal{N}) \text{ such that } I_{E_o(v)} = I_{C'}.$$ 

(4.14)

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

$$C_{\text{cod}}(\mathcal{N}', f) = \min \text{-cut}(\mathcal{N}', f) \quad \text{[from Theorem 4.3.3]} \quad (4.15)$$

$$= \min_{v \in V' - \rho} \frac{1}{\log |A| R_{\epsilon_a(v) \cdot f}} \quad \text{[from Theorem 4.2.1, (4.6), (4.11)]}$$

$$\geq \min_{C \in \Lambda(\mathcal{N})} \frac{1}{\log |A| R_{IC \cdot f}} \quad \text{[from (4.14)].} \quad (4.16)$$

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

$$\sum_{t_i \in T(\mathcal{N})} u_i r_i \quad \text{subject to} \quad \left\{ \begin{array}{l} u_i \geq 0 \quad \forall \ t_i \in T(\mathcal{N}) , \\ \sum_{i \in I_e} u_i \leq 1 \quad \forall \ e \in E(G) \end{array} \right. \quad (4.17)$$

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

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

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 $(ck, \lceil cn \log_q p \rceil)$ code (e.g. see [75]). 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

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

Since $\epsilon$ 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

$$C_{\text{cod}}(\mathcal{N}, f) \geq \frac{\min_{C \in \Lambda(\mathcal{N})} |C|}{(q - 1) \cdot \log_q P(s)}$$
where $P(s)$ is the smallest prime number\footnote{From Bertrand’s Postulate [76, p.343], we have $P(s) \leq 2s$.} 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 $\sigma_j$ 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

$$\frac{k}{n} \geq \frac{\min_{C \in \Lambda(N)} |C|}{\log_2 P(s)} - \epsilon.$$
puting the histogram target function \( \hat{f} \) with an achievable computing rate\(^{11}\) of at least

\[
\frac{1}{(q - 1)} \cdot \frac{1}{\log_q 2} \cdot \min_{C \in \Lambda(N)} |C| \cdot \log_2 P(s) - 2\epsilon.
\]

Since \( \epsilon \) is arbitrary, the result follows.

---

**Figure 4.2:** The Reverse Butterfly Network \( N_1 \) has two binary sources \( \{\sigma_1, \sigma_2\} \) and network \( N_2 \) has three binary sources \( \{\sigma_1, \sigma_2, \sigma_3\} \), each with \( A = \{0, 1\} \). Each network’s receiver \( \rho \) computes the arithmetic sum of the source messages.

**Example 4.3.8.** Consider networks \( N_1 \) and \( N_2 \) in Figure 4.2, each with alphabet \( A = \{0, 1\} \) and the (symmetric) arithmetic sum target function \( f \). Theorem 4.3.7 provides a larger lower bound on the computing capacity \( C_{\text{cod}}(N_1, f) \) than Theorem 4.3.5, but a smaller lower bound on \( C_{\text{cod}}(N_2, f) \).

- For network \( N_1 \) (in Figure 4.2), we have \( \max_{C \in \Lambda(N_1)} R_{C,f} = 3 \) and \( \min_{C \in \Lambda(N_1)} |C| = 2 \), both of which occur, for example, when \( C \) consists of the two in-edges to the receiver \( \rho \). Also, \((q - 1) \log_q P(s, q) = \log_2 3 \) and \( \Pi(N_1) = 3/2 \), so

\[
C_{\text{cod}}(N_1, f) \geq (3/2)/\log_2 3 \quad \text{[from Theorem 4.3.5]}
\]

\[
C_{\text{cod}}(N_1, f) \geq 2/\log_2 3 \quad \text{[from Theorem 4.3.7].}
\]

---

\(^{11}\)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 $\max_{C \in \Lambda(\mathcal{N}_2)} R_{C,f} = 4$ and $\min_{C \in \Lambda(\mathcal{N}_2)} |C| = 1$, 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

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

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

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 [68], 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 $f : \mathcal{A}^s \rightarrow \mathcal{B}$ 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 \( R_{I, f} \leq |f^I(\mathcal{A}^{|I|})| \). We need \(|f^I(\mathcal{A}^{|I|})| \leq |f(\mathcal{A}^s)|\) to complete the proof which follows from Definition 4.1.8(2).

\[\blacksquare\]

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

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

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

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

\[
\mathcal{C}_{\text{cod}}(\mathcal{N}, f) \geq \frac{\Pi(\mathcal{N})}{|\mathcal{A}|} \cdot \min_{C \in \Lambda(\mathcal{N})} \frac{1}{\log_{|\mathcal{A}|} R_{C, f}} \geq \frac{\Pi(\mathcal{N})}{|\mathcal{A}|} \cdot \frac{1}{\log_{|\mathcal{A}|} |f(\mathcal{A}^s)|} \quad \text{[from Lemma 4.4.1].} \tag{4.19}
\]

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 \( \rho \). Thus,

\[
\text{min-cut}(\mathcal{N}, f) \leq \frac{|\mathcal{E}_i(\rho)|}{\log_{|\mathcal{A}|} R_{\mathcal{E}_i(\rho), f}} \quad \text{[from (4.4)]}
\]

\[
= \frac{|\mathcal{E}_i(\rho)|}{\log_{|\mathcal{A}|} |f(\mathcal{A}^s)|} \quad \text{[from } I_{\mathcal{E}_i(\rho)} = S \text{ and Definition 4.1.5].} \tag{4.20}
\]

Combining (4.19) and (4.20) completes the proof. \[\blacksquare\]

**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

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

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 $f$,

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

$\blacksquare$

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

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

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

Proof. We have

$$\min\text{-cut}(\mathcal{N}, f) = \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{|A|} R_{C, f}} \leq \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\lambda |I_C|} \quad [\text{from } f \text{ being } \lambda\text{-exponential}]$$

$$= \frac{1}{\lambda} \cdot \min\text{-cut}(\mathcal{N}) \quad [\text{from (4.3)}].$$

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

\[
\frac{\min\text{-cut}(\mathcal{N}, f)}{C_{\text{cod}}(\mathcal{N}, f)} \leq \frac{1}{\lambda} \cdot \frac{\min\text{-cut}(\mathcal{N})}{C_{\text{cod}}(\mathcal{N}, f)} \leq \frac{1}{\lambda}
\]

where the last inequality follows because a computing rate of \(\min\text{-cut}(\mathcal{N})\) 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., \(C_{\text{cod}}(\mathcal{N}, f) \geq \min\text{-cut}(\mathcal{N})\).

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

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

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 [77, p.405, Theorem 8.4.20] that

\[
\Pi(\mathcal{N}) = \min_{C \in \Lambda(\mathcal{N})} |C|.
\]

Then,

\[
C_{\text{cod}}(\mathcal{N}, f)
\geq \min_{C \in \Lambda(\mathcal{N})} |C| \cdot \min_{C \in \Lambda(\mathcal{N})} \frac{1}{\log_{|A|} \hat{R}_C,f}
\geq \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\lambda} \quad \text{[from \(f\) being \(\lambda\)-bounded].} \tag{4.21}
\]
On the other hand,

\[
\text{min-cut}(\mathcal{N}, f) = \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{|\mathcal{A}|} R_{C,f}} \\
\leq \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{|\mathcal{A}|} \hat{R}_f} \tag{from the definition of } \hat{R}_f \tag{4.22}
\]

Combining (4.21) and (4.22) gives

\[
\frac{\text{min-cut}(\mathcal{N}, f)}{C_{\text{cod}}(\mathcal{N}, f)} \leq \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{|\mathcal{A}|} \hat{R}_f} \cdot \frac{1}{\min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\lambda}} = \frac{\lambda}{\log_{|\mathcal{A}|} \hat{R}_f}.
\]

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

\[
C_{\text{cod}}(\mathcal{N}, f) = \text{min-cut}(\mathcal{N}, f).
\]

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 \( \lambda \)-exponential (respectively, \( \lambda \)-bounded) target functions, the computing capacity \( C_{\text{cod}}(\mathcal{N}, f) \) is at least a constant fraction of the min-cut(\( \mathcal{N}, f \)) for any constant \( \lambda \) 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 \( C_{\text{cod}}(\mathcal{N}, f) \) 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 $A = \{0, 1\}$, connected to the receiver node $\rho$ via a relay $\sigma_0$. Each bold edge denotes $L$ parallel capacity-one edges. $\rho$ computes the arithmetic sum of the source messages.

multicommodity flow. It was shown in [71] 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 [78] 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 $f$,

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

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

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

Let $m$ be the number of sources disconnected from the receiver $\rho$ by a cut $C$ in the
network $\mathcal{N}_{M,L}$. For each such source $\sigma$, the cut $C$ must contain the edge $(\sigma, \rho)$ as well as either the $L$ parallel edges $(\sigma, \sigma_0)$ or the $L$ parallel edges $(\sigma_0, \rho)$. Thus,

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

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

$$c^*/\ln 2 \geq \min_{1 \leq m \leq M} \left\{ \frac{m + 1}{\ln(m + 1)} \right\} \quad (4.24)$$

$$\geq \min_{x \geq 2} \left\{ \frac{x}{\ln x} \right\} > \min_{x \geq 2} \left\{ \frac{x}{x - 1} \right\} > 1,$$

$$L = c^* \log_2 (m^* + 1) - m^* \quad \text{[from (4.23)]}$$

$$\leq c^* \log_2 \left( \frac{c^*}{\ln 2} \right) - \left( \frac{c^*}{\ln 2} - 1 \right) \quad (4.25)$$

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

$$L =$$

$$O\left( \min\text{-cut}(\mathcal{N}_{M,L}, f) \log_2(\min\text{-cut}(\mathcal{N}_{M,L}, f)) \right) \quad \text{[from (4.25)],} \quad (4.26)$$

$$\min\text{-cut}(\mathcal{N}_{M,L}, f) = \Omega((\log M)^{1-\epsilon}) \quad \text{[from (4.26)],} \quad (4.27)$$

$$C_{\text{cod}}(\mathcal{N}_{M,L}, f)$$

$$= O(1) \quad \text{[from Lemma 4.7.1]}$$

$$= O\left( \frac{1}{(\log M)^{1-\epsilon}} \right) \cdot \min\text{-cut}(\mathcal{N}_{M,L}, f) \quad \text{[from (4.27)].}$$

**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 $N_3$ has three binary sources, $\sigma_1$, $\sigma_2$, and $\sigma_3$ with $A = \{0, 1\}$ and the receiver $\rho$ 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 $N_3$ with respect to the arithmetic sum target function $f$ is

$$C_{\text{cod}}(N_3, f) = \frac{2}{1 + \log_2 3}.$$

**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 $\sigma_1$, $\sigma_2$, $\sigma_3$, respectively, and let $z_1, z_2 \in \{0, 1\}^n$ be the vectors carried by edges $(\sigma_1, \rho)$ and $(\sigma_2, \rho)$, respectively.

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

$$\frac{k}{n} \leq \frac{2}{1 + \log_2 3}. \quad (4.28)$$

Then we have

$$2^n \geq 3^{k/2} 2^{k/2}. \quad (4.29)$$

We will describe a $(k, n)$ network code for computing $f$ in the network $N_3$. Define
vectors $y^{(1)}, y^{(2)} \in \{0, 1\}^k$ by:

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

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

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 $\sigma_3$ sends its full message vector $w^{(3)}$ ($k < n$) to each of the two nodes it is connected to. Source $\sigma_1$ (respectively, $\sigma_2$) 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 $\rho$ 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 $w^{(1)}_i + w^{(2)}_i + w^{(3)}_i$, i.e., the arithmetic sum target function $f$. The above solution achieves a computing rate of $k/n$. From (4.28), it follows that

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

We now prove a matching upper bound on the computing capacity $C_{\text{cod}}(\mathcal{N}_3, f)$. 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 $\bar{w}^{(3)} \in \{0, 1\}^k$.

$$A_p = \{(z_1, z_2) : w^{(1)} + w^{(2)} + w^{(3)} = p\}.$$
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
\[ \{ (z_1, z_2) : (w^{(1)}, w^{(2)}) \in T, w^{(3)} = \tilde{w}^{(3)} \} \subseteq A_p. \] (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
\[ |T| = 2^{k-j}. \] (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,
\[ \left| \{ (z_1, z_2) : (w^{(1)}, w^{(2)}) \in T, w^{(3)} = \tilde{w}^{(3)} \} \right| \geq |T| \] (4.33)
and therefore

\[ |A_p| \]

\[ \geq \left| \{ (z_1, z_2) : (w^{(1)}, w^{(2)}) \in T, w^{(3)} = \tilde{w}^{(3)} \} \right| \quad \text{[from (4.31)]} \]

\[ \geq |T| \quad \text{[from (4.33)]} \]

\[ = 2^{k-j}. \quad \text{[from (4.32)]} \quad (4.34) \]

We have the following inequalities:

\[ 4^n \geq \left| \{ (z_1, z_2) : w^{(1)}, w^{(2)}, w^{(3)} \in \{0, 1\}^k \} \right| \]

\[ = \sum_{p \in \{0, 1, 2, 3\}^k} |A_p| \quad (4.35) \]

\[ = \sum_{j=0}^{k} \sum_{p \in \{0, 1, 2, 3\}^k, |\{i : p_i \in \{0, 3\}\}| = j} |A_p| \]

\[ \geq \sum_{j=0}^{k} \sum_{p \in \{0, 1, 2, 3\}^k, |\{i : p_i \in \{0, 3\}\}| = j} 2^{k-j} \quad \text{[from (4.34)]} \]

\[ = \sum_{j=0}^{k} \binom{k}{j} 2^k 2^{k-j} \]

\[ = 6^k \quad (4.36) \]

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

\[ C_{\text{cod}}(N_3, f) \leq \frac{2}{1 + \log_2 3}. \quad (4.37) \]

Combining (4.30) and (4.37) concludes the proof. \[\square\]
Corollary 4.5.2. For the network $N_3$ with the arithmetic sum target function $f$,

$$C_{\text{cod}}(N_3, f) < \text{min-cut}(N_3, f).$$

Proof. Consider the network $N_3$ 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

$$\text{min-cut}(N_3, f) = 1 \quad [\text{from (4.5)}].$$

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 $N_3$ derived in Section 4.3 with the exact computing capacity. It can be shown that $\Pi(N_3) = 1$. If $f$ is the arithmetic sum target function, then

$$C_{\text{cod}}(N_3, f) \geq 1/2 \quad [\text{from Theorem 4.3.5}]$$
$$C_{\text{cod}}(N_3, f) \geq 1/\log_2 5 \quad [\text{from Theorem 4.3.7}]$$
$$C_{\text{cod}}(N_3, f) \geq 1/2 \quad [\text{from Theorem 4.4.2}].$$

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

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

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

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

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

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

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 $$\lim_{M \to \infty} C_{\text{cod}}(N_{M,L}, f) = 1$$.

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

$$g_i : \{0, 1\}^k \rightarrow \{0, 1\}^n$$
be the corresponding encoding function on the edge \((\sigma_i, \rho)\). Then for any \(A_1, A_2, \ldots, A_M \subseteq \{0, 1\}^k\), we have
\[
\left( \prod_{i=1}^{M} |g_i(A_i)| \right) \cdot 2^{nL} \geq \left| Q\left( \prod_{i=1}^{M} A_i \right) \right| .
\] (4.40)

Each \(A_i\) represents a set of possible message vectors of source \(\sigma_i\). 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 \(\rho\) (i.e., \(|g_i(A_i)|\) possible vectors on each edge \((\sigma_i, \rho)\) and \(2^{nL}\) possible vectors on the \(L\) parallel edges \((\sigma_0, \rho)\)). 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 \(|g_i^{-1}(z_i)| \geq 2^{k-n}\) and choose \(A_i = g_i^{-1}(z_i)\) for each \(i\). Also, let \(U^{(M)} = \prod_{i=1}^{M} A_i\). Then we have
\[
|Q(U^{(M)})| \leq 2^{nL} \quad \text{[from } |g_i(A_i)| = 1 \text{ and (4.40)].}
\] (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\(^{13}\)
\[
|Q(U^{(M)})| \geq (M + 1)^{\gamma(k/n)k}
\] (4.42)

where the function \(\gamma\) 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
\[
r \gamma(r) \log_2(M + 1) \leq \frac{1}{n} \log_2 |Q(U^{(M)})| \leq L.
\] (4.43)

From (4.43), we have
\[
r \gamma(r) \leq \frac{L}{\log_2(M + 1)}.
\] (4.44)

\(^{13}\) One can compare this lower bound to the upper bound \(|Q(U^{(M)})| \leq (M + 1)^k\) which follows from (4.38).
The quantity \( r \gamma(r) \) is monotonic increasing from 0 to \( \infty \) 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. \( C_{\text{cod}}(N_{M,L}, f) \leq 1 \). In summary,

\[
\lim_{M \to \infty} C_{\text{cod}}(N_{M,L}, f) = 1.
\]

\[\blacksquare\]

**Lemma 4.7.2.** Let \( k, n, M \) be positive integers, with \( k > n \). For each \( i \in \{1, \ldots, M\} \), 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,

\[
|Q(U^{(M)})| \geq (M + 1)^{\gamma(k/n)k}.
\]

**Proof.** The result follows from Lemmas 4.7.4 and 4.7.7. \[\blacksquare\]

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 \( j \in \{1, 2, \ldots, k\} \), define the map

\[
h^{(j)} : \{0,1,\ldots, M\}^k \longrightarrow \{0,1,\ldots, M\}^k
\]

by

\[
(h^{(j)}(p))_i = \begin{cases} 
\max\{0, p_i - 1\} & \text{if } i = j \\
p_i & \text{otherwise.}
\end{cases}
\]  

(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 \( j \in \{1, 2, \ldots, k\} \), define the map

\[
\tilde{\phi}^{(j)} : 2^{\{0,1\}^k} \times \{0,1\}^k \longrightarrow \{0,1\}^k
\]
by

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

(4.46)

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

\[ \phi^{(j)} : 2^{\{0, 1\}^k} \rightarrow 2^{\{0, 1\}^k} \]

by

\[ \phi^{(j)}(A) = \left\{ \hat{\phi}^{(j)}(A, a) : a \in A \right\}. \]  

(4.47)

Note that

\[ |\hat{\phi}^{(j)}(A)| = |A|. \]  

(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 \( a \in A \),

\[ h^{(j)}(a) \in A. \]  

(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

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

(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.,

\[ \phi^{(m)}(B) = B. \]  

(4.51)
Consider any vector \( c \in \phi^{(r)}(B) \). From (4.49), we need to show that \( h^{(m)}(c) \in \phi^{(r)}(B) \).

We have the following cases.

\[ c_\tau = 1 : \]

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

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

\[ h^{(r)}(h^{(m)}(c)) = h^{(m)}(h^{(r)}(c)) \in B \quad \text{[from (4.51), (4.52)]} \quad (4.54) \]

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

\[ c_\tau = 0 : \]

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

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

\[ h^{(m)}(h^{(r)}(b)) = h^{(r)}(h^{(m)}(b)) \in \phi^{(r)}(B) \quad \text{[from (4.56)]} \quad (4.57) \]

\[ h^{(m)}(c) \in \phi^{(r)}(B) \quad \text{[from (4.55), (4.57)].} \]

Thus, the lemma is true for \( t = \tau \) and the induction argument is complete. \( \blacksquare \)

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 verified 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
\[ |B_i| = |A_i| \geq 2^{k-n}. \]  
(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.**
\[ |Q(U^{(M)})| \geq |Q(V^{(M)})|. \]

**Proof.** We begin by showing that
\[ |Q(U^{(M)})| \geq |Q(\phi^{(1)}(U^{(M)}))|. \]  
(4.59)

For every \( p \in \{0, 1, \ldots, M\}^{k-1} \), let
\[
\begin{align*}
\varphi(p) &= \{ r \in Q(U^{(M)}): (r_2, \cdots, r_k) = p \} \\
\varphi_1(p) &= \{ s \in Q(\phi^{(1)}(U^{(M)})): (s_2, \cdots, s_k) = p \}
\end{align*}
\]
and note that
\[
\begin{align*}
Q(U^{(M)}) &= \bigcup_{p \in \{0, 1, \ldots, M\}^{k-1}} \varphi(p) \quad \text{(4.60)} \\
Q(\phi^{(1)}(U^{(M)})) &= \bigcup_{p \in \{0, 1, \ldots, M\}^{k-1}} \varphi_1(p) \quad \text{(4.61)}
\end{align*}
\]
where the unions are disjoint. We show that for every \( p \in \{0, 1, \ldots, M\}^{k-1} \),
\[ |\varphi(p)| \geq |\varphi_1(p)| \]  
(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
\[ |\varphi_1(p)| \leq K_p + 1. \] (4.63)

Since \((K_p, p_1, \ldots, p_{k-1}) \in \varphi_1(p)\), there exists \((a^{(1)}, a^{(2)}, \ldots, a^{(M)}) \in U^{(M)}\) such that
\[ \sum_{i=1}^{M} \hat{\phi}^{(i)}(A_i, a^{(i)}) = (K_p, p_1, \ldots, p_{k-1}). \] (4.64)

Then from the definition of \(\hat{\phi}^{(i)}\) in (4.46), there are \(K_p\) of the \(a^{(i)}\)'s from among \(\{a^{(1)}, \ldots, a^{(M)}\}\) such that \(a^{(i)} = 1\) and \(\hat{\phi}^{(i)}(A_i, a^{(i)}) = a^{(i)}\). Let \(I = \{i_1, \ldots, i_{K_p}\}\) \(\subseteq \{1, 2, \ldots, M\}\) be the index set for these vectors and let \(\hat{a}^{(i)} = h^{(i)}(a^{(i)})\) for each \(i \in I\). Then for each \(i \in I\), we have
\[ a^{(i)} = (1, a^{(i)}_2, \ldots, a^{(i)}_k) \in A_i \]
\[ \hat{a}^{(i)} = (0, a^{(i)}_2, \ldots, a^{(i)}_k) \in A_i \] [from \(\hat{\phi}^{(i)}(A_i, a^{(i)}) = a^{(i)}\), (4.46)].

Let
\[ R = \left\{ \sum_{i=1}^{M} b^{(i)} : b^{(i)} \in \{a^{(i)}, \hat{a}^{(i)}\} \text{ for } i \in I, \right. \]
\[ \left. b^{(i)} = a^{(i)} \text{ for } i \notin I \right\} \subseteq \varphi(p). \] (4.65)

From (4.64) and (4.65), for every \(r \in R\) we have
\[ r_1 \in \{0, 1, \ldots, |I|\}, \]
\[ r_i = p_i \ \forall \ i \in \{2, 3, \ldots, k\} \]
and thus
\[ |R| = |I| + 1 = K_p + 1. \] (4.66)

Hence, we have
\[ |\varphi(p)| \geq |R| \] \quad [from (4.65)]
\[ = K_p + 1 \] \quad [from (4.66)]
\[ \geq |\varphi_1(p)| \] \quad [from (4.63)]
and then from (4.60) and (4.61), it follows that

\[ |Q(U^{(M)})| \geq |Q(\phi(1)(U^{(M)}))|. \]

For any \( A \subseteq \{0,1\}^k \) and any \( j \in \{1,2,\ldots,k\} \), we know that \( |\phi^{(j)}(A)| \subseteq \{0,1\}^k \).

Thus, the same arguments as above can be repeated to show that

\[ |Q(\phi(1)(U^{(M)}))| \geq |Q(\phi^{(2)}(\phi(1)(U^{(M)})))| \]
\[ \geq |Q(\phi^{(3)}(\phi^{(2)}(\phi(1)(U^{(M)}))))| \]
\[ \vdots \]
\[ \geq |Q(\phi^{(k)}(\phi^{(k-1)}(\ldots \phi^{(1)}(U^{(M)}))))| \]
\[ = |Q(V^{(M)})|. \]

\[ \blacksquare \]

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

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

**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 \( i_1,i_2,\ldots,i_t \in \{1,2,\ldots,k\} \) such that

\[ q = h^{(i_1)}(h^{(i_2)}(\ldots(h^{(i_t)}(p)))) \).

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

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

and \( b^{(m)}_i = 1 \) for some \( m \in \{1,2,\ldots,M\} \). 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 \( Q(V^{(M)}) \).

The lemma below is presented in [53] without proof, as the proof is straightforward.

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

\[
\min_{0 \leq m_i \leq M, \sum_{i=1}^{k} m_i \geq \delta M k} \prod_{i=1}^{k} (1 + m_i) \geq (M + 1)^{\delta k}.
\]

(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 \( \gamma \) defined in (4.39).

**Lemma 4.7.7.**

\[
|Q(V^{(M)})| \geq (M + 1)^{\gamma(k/n)}.
\]

**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

\[
\sum_{j=0}^{\lfloor \delta k \rfloor} \binom{k}{j} \leq 2^{kH(\delta)} \quad \text{[from [79, p.15, Theorem 1]]}
\]

\[
= 2^{(k-n)/2} \quad \text{[from (4.39)].}
\]

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

\[
p = \sum_{i=1}^{M} b^{(i)} \in Q(V^{(M)}).
\]
It follows that $p_j \in \{0, 1, 2, \ldots, M\}$ for every $j \in \{1, 2, \ldots, k\}$, and
\[
\sum_{j=1}^{k} p_j = \sum_{i=1}^{M} |b^{(i)}|_H \geq \delta M k.
\]

(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 $Q(V^{(M)})$. Therefore,
\[
|Q(V^{(M)})| \geq \prod_{j=1}^{k} (1 + p_j) \\
\geq (M + 1)^{\delta k} \quad \text{[from (4.68) and Lemma 4.7.6].}
\]

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 [39, 41, 78], whether linear codes are sufficient to achieve the capacity [80–83], and cut-set upper bounds on the capacity and the tightness of such bounds [41, 73, 78].

Network computing, on the other hand, considers a more general problem in which each receiver node demands a target function of the source messages [54, 63, 65, 67, 84, 85]. 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 [85], 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. [86]).

In network coding, it is known that linear codes are sufficient to achieve the coding capacity for multicast networks [39], but they are not sufficient in general to achieve the coding capacity for non-multicast networks [81]. 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 [87]. However, it has been shown that for single-receiver networks, linear coding is sufficient for solvability when computing a scalar linear target function [65, 75]. 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 [84] 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 $C_{\text{cod}}(N, f)$, $C_{\text{lin}}(N, f)$, and $C_{\text{rout}}(N, f)$ denote the computing capacity, linear computing capacity, and routing computing capacity, respectively, for a network $N$ with $s$ sources and target function $f$. The columns labeled $f$ and $A$ indicate constraints on the target function $f$ and the source alphabet $A$, respectively.

<table>
<thead>
<tr>
<th>Result</th>
<th>$f$</th>
<th>$A$</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall f \ \forall N \ C_{\text{lin}}(N, f) = C_{\text{rout}}(N, f)$</td>
<td>non-reducible</td>
<td>field</td>
<td>Theorem 5.4.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>semi-injective</td>
<td>ring</td>
</tr>
<tr>
<td>$\forall f \ \forall N \ C_{\text{cod}}(N, f) = C_{\text{rout}}(N, f)$</td>
<td>injective</td>
<td></td>
<td>Theorem 5.4.9</td>
</tr>
<tr>
<td>$\forall f \ \exists N \ C_{\text{cod}}(N, f) &gt; C_{\text{lin}}(N, f)$</td>
<td>non-injective &amp; non-reducible</td>
<td>field</td>
<td>Theorem 5.4.11</td>
</tr>
<tr>
<td>$\forall f \ \exists N \ C_{\text{lin}}(N, f) &gt; C_{\text{rout}}(N, f)$</td>
<td>reducible</td>
<td>ring</td>
<td>Theorem 5.4.12</td>
</tr>
<tr>
<td>$\exists f \ \exists N \ C_{\text{cod}}(N, f) &gt; C_{\text{lin}}(N, f) &gt; C_{\text{rout}}(N, f)$</td>
<td>reducible</td>
<td></td>
<td>Theorem 5.4.14</td>
</tr>
<tr>
<td>$\forall f \ \forall N \ C_{\text{cod}}(N, f) = C_{\text{lin}}(N, f) \leq s C_{\text{rout}}(N, f)$</td>
<td>linear</td>
<td>field</td>
<td>Theorem 5.5.7</td>
</tr>
<tr>
<td>$\forall f \ \exists N \ C_{\text{lin}}(N, f) = s C_{\text{rout}}(N, f)$</td>
<td>linear</td>
<td>field</td>
<td>Theorem 5.5.8</td>
</tr>
<tr>
<td>$\exists f \ \exists N \ C_{\text{cod}}(N, f)$ is irrational</td>
<td>arithmetic sum</td>
<td></td>
<td>Theorem 5.6.3</td>
</tr>
</tbody>
</table>

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 [39] 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 [68].

Our main results are summarized in Table 5.1.
5.2 Network model and definitions

In this chapter, a network \( N = (G, S, \rho) \) consists of a finite, directed acyclic multigraph \( G = (V, E) \), a set \( S = \{\sigma_1, \ldots, \sigma_s\} \subseteq V \) of \( s \) distinct source nodes and a single receiver \( \rho \in V \). We assume that \( \rho \notin S \), and that the graph\(^1 \) \( G \) contains a directed path from every node in \( V \) to the receiver \( \rho \). For each node \( u \in V \), let \( E_i(u) \) and \( E_o(u) \) 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 \( e = (u, v) \in E \), 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, \( A \) will denote a source alphabet and \( B \) will denote a receiver alphabet. For any positive integer \( m \), any vector \( x \in A^m \), 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 \( (x_{i_1}, x_{i_2}, \ldots, x_{i_q}) \in A^{|I|} \). Sometimes we view \( A \) 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 \( N = (G, S, \rho) \), we use \( s \) throughout the chapter to denote the number \( |S| \) of receivers in \( N \). For given network \( N \), a target function is a mapping

\[
f : A^s \longrightarrow B.
\]

The goal in network computing is to compute \( f \) at the receiver \( \rho \), 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 \( A \) be a ring. A target function \( f : A^s \longrightarrow B \) is said to be reducible if there exists an integer \( \lambda \) satisfying \( \lambda < s \), an \( s \times \lambda \) matrix \( T \) with elements...
Table 5.2: Definitions of some target functions.

<table>
<thead>
<tr>
<th>Target function  $f$</th>
<th>Alphabet $\mathcal{A}$</th>
<th>$f(x_1, \ldots, x_s)$</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>arbitrary</td>
<td>$(x_1, \ldots, x_s)$</td>
<td>$\mathcal{B} = \mathcal{A}^s$</td>
</tr>
<tr>
<td>arithmetic sum</td>
<td>${0, \ldots, q-1}$</td>
<td>$x_1 + \cdots + x_s$</td>
<td>‘+’ is ordinary integer addition, $\mathcal{B} = {0, 1, \ldots, s(q-1)}$</td>
</tr>
<tr>
<td>mod $r$ sum</td>
<td>${0, \ldots, q-1}$</td>
<td>$x_1 \oplus \cdots \oplus x_s$</td>
<td>$\oplus$ is mod $r$ addition, $\mathcal{B} = \mathcal{A}$</td>
</tr>
<tr>
<td>linear</td>
<td>ring</td>
<td>$a_1 x_1 + \cdots + a_s x_s$</td>
<td>arithmetic in the ring, $\mathcal{B} = \mathcal{A}$</td>
</tr>
<tr>
<td>maximum</td>
<td>ordered set</td>
<td>$\max {x_1, \ldots, x_s}$</td>
<td>$\mathcal{B} = \mathcal{A}$</td>
</tr>
</tbody>
</table>

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

$$g(xT) = f(x).$$ \hfill (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

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

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 \( g(y_1, y_2) = y_1y_2 \), we get

\[
g(xT) = g(x_1 + x_2, x_3)
= (x_1 + x_2)x_3
= f(x).
\]

Thus the target function \( f \) is reducible.

**Example 5.2.3.** The notion of reducibility requires that for a target function \( f : \mathcal{A}^s \rightarrow \mathcal{B} \), 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 \( N \) with source set \( S \) and alphabet \( \mathcal{A} \), a message generator is any mapping

\[
\alpha : S \rightarrow \mathcal{A}^k.
\]

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 \( N \) consists of the following:

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

\(^2\)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:

\[
\begin{align*}
  h^{(e)} : \left( \prod_{\hat{e} \in \mathcal{E}(v)} \mathcal{A}^n \right) & \times \mathcal{A}^k \rightarrow \mathcal{A}^n & \text{if } v \text{ is a source node} \\
  h^{(e)} : \prod_{\hat{e} \in \mathcal{E}(v)} \mathcal{A}^n & \rightarrow \mathcal{A}^n & \text{otherwise}.
\end{align*}
\]

(ii) A decoding function \( \psi \) of the form:

\[
\psi : \prod_{\hat{e} \in \mathcal{E}(v)} \mathcal{A}^n \rightarrow \mathcal{B}^k.
\]

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 \( \alpha \). 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 \( \alpha \).

**Definition 5.2.5.** Suppose in a network \( \mathcal{N} \), the in-edges of \( \rho \) are \( e_1, e_2, \ldots, e_{|\mathcal{E}(\rho)|} \). 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 \( \alpha \), the decoding function satisfies

\[
\psi \left( z_{e_1}, \cdots, z_{e_{|\mathcal{E}(\rho)|}} \right)_j = f \left( \left( \alpha(\sigma_1)_j, \cdots, \alpha(\sigma_\rho)_j \right) \right), \tag{5.2}
\]

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 [68]. 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

$$C_{\text{cod}}(\mathcal{N}, f) = \sup \left\{ \frac{k}{n} : \exists (k, n) \text{ network code that computes } f \text{ in } \mathcal{N} \right\}.$$

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

$$C_{\text{lin}}(\mathcal{N}, f) = \sup \left\{ \frac{k}{n} : \exists (k, n) \text{ linear network code that computes } f \text{ in } \mathcal{N} \right\}.$$

The *routing computing capacity* $C_{\text{rout}}(\mathcal{N}, f)$ is defined similarly by restricting the encoding functions to routing. We call the quantity $C_{\text{cod}}(\mathcal{N}, f) - C_{\text{lin}}(\mathcal{N}, f)$ the *computing capacity gain* of using nonlinear coding over linear coding. Similar “gains”, such as, $C_{\text{cod}}(\mathcal{N}, f) - C_{\text{rout}}(\mathcal{N}, f)$ and $C_{\text{lin}}(\mathcal{N}, f) - C_{\text{rout}}(\mathcal{N}, f)$ 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 $C \subseteq E$ in network $\mathcal{N}$ is said to separate sources $\sigma_{m_1}, \ldots, \sigma_{m_d}$ from the receiver $\rho$, if for each $i \in \{1, 2, \ldots, d\}$, every directed path from $\sigma_{m_i}$ to $\rho$ 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 [73]. Let $C_{\text{rout}}(\mathcal{N})$ 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 [73, Theorem 4.2] that for any single-receiver network $\mathcal{N}$,

$$C_{\text{rout}}(\mathcal{N}) = \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{|I_C|}. \tag{5.4}$$

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

$$C_{\text{rout}}(\mathcal{N}, f) = C_{\text{rout}}(\mathcal{N}).$$

**Proof.** Since any routing code that computes the identity target function can be used to compute any target function $f$, we have

$$C_{\text{rout}}(\mathcal{N}, f) \geq C_{\text{rout}}(\mathcal{N}).$$

Conversely, it is easy to see that every component of every source message must be received by $\rho$ in order to compute $f$, so

$$C_{\text{rout}}(\mathcal{N}, f) \leq C_{\text{rout}}(\mathcal{N}).$$
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 $f : A^s \rightarrow B$, then

$$C_{cod}(\mathcal{N}, f) \leq (\log_2 |A|) \min_{C \in \Lambda(\mathcal{N})} |C|.$$ 

**Proof.** Using [85, Theorem II.1], one finds the term $\text{min-cut}(\mathcal{N}, f)$ defined in [85, Equation (3)] in terms of a quantity $R_{IC, f}$, which in turn is defined in [85, Definition 1.5]. Since target functions are restricted to not being constant functions of any of their arguments, we have $R_{IC, f} \geq 2$, from which the result follows. ■

**Theorem 5.2.12.** If $\mathcal{N}$ is network with a target function $f : A^s \rightarrow B$, then

$$C_{cod}(\mathcal{N}, f) \leq s (\log_2 |A|) C_{rout}(\mathcal{N}, f).$$

**Proof.**

$$C_{cod}(\mathcal{N}, f) \leq (\log_2 |A|) \min_{C \in \Lambda(\mathcal{N})} |C| \quad \text{[from Lemma 5.2.11]}$$

$$\leq s (\log_2 |A|) C_{rout}(\mathcal{N}, f). \quad \text{[from (5.3), (5.4), and Theorem 5.2.10]}$$

■
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 $f : \mathcal{A}^2 \rightarrow \{0, 1, 2\}$ 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

$$C_{\text{lin}}(\mathcal{N}_4)^R = \sup \left\{ \frac{k}{n} : \exists (k, n) \text{-linear code that computes } f \text{ in } \mathcal{N} \right\}.$$

The received vector $z$ at $\rho$ can be viewed as a function of the source vectors generated at $\sigma_1$ and $\sigma_2$. 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

$$z(\alpha(\sigma_1), \alpha(\sigma_2)) = \alpha(\sigma_1) M_1 + \alpha(\sigma_2) M_2. \quad (5.5)$$

<table>
<thead>
<tr>
<th>$f$</th>
<th>$a_0$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$a_1$</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$a_2$</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a_3$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Let \( m_{i,1}, \ldots, m_{i,k} \) 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 \( f: \mathcal{A}^2 \to \{0, 1, 2\} \) be the target function shown in Table 5.3, where \( \alpha_i = i \), for each \( i \). If a \((k, n)\) linear code over \( \mathcal{A} \) computes \( f \) in \( \mathcal{N}_4 \) and \( \rho \) 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 \( \rho \) receives a 0 by (5.5) and must decode a 0 since \( f((0, 0)) = 0 \) (from Table 5.3). Thus, \( \rho \) 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 \( \rho \) 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

\[
m_{1,j} = \sum_{i=1 \atop i \neq j}^k -\alpha(\sigma_1)_j^{-1} \alpha(\sigma_1)_i m_{1,i} + \sum_{i=1}^k -\alpha(\sigma_1)_j^{-1} \alpha(\sigma_2)_i m_{2,i} \tag{5.6}
\]

\[
= y^{(1)} M_1 + y^{(2)} M_2 \tag{5.7}
\]

where \( y^{(1)} \) and \( y^{(2)} \) are \( k \)-dimensional vectors defined by

\[
y^{(1)}_i = \begin{cases} 
-\alpha(\sigma_1)_j^{-1} \alpha(\sigma_1)_i & \text{if } i \neq j \\
0 & \text{if } i = j 
\end{cases}
\]

\[
y^{(2)}_i = -\alpha(\sigma_1)_j^{-1} \alpha(\sigma_2)_i. \tag{5.8}
\]

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 \( z(y^{(1)}_j, y^{(2)}_j) = m_{1,j} \). Thus, in order for the code to compute \( f \), we must have \( f(x_j, 0) = f(y^{(1)}_j, y^{(2)}_j) \). But
\[ f(x_j, 0) = f(1, 0) = 1 \text{ and} \]
\[ f(y_j^{(1)}, y_j^{(2)}) = f(0, -\alpha(\sigma_1)^{-1} \alpha(\sigma_2)_{j}) \]
\[ = f(0, -\alpha(\sigma_1)^{-1} \alpha(\sigma_1)_{j}) \quad \text{[from } \alpha(\sigma_1) = \alpha(\sigma_2)\text{]} \]
\[ = f(0, -1) \]
\[ = f(0, 3) \quad \text{[from } 3 = -1 \text{ in } \mathbb{Z}_4\text{]} \]
\[ = 2 \quad \text{[from Table 5.3],} \]

a contradiction. Thus, \( \alpha(\sigma_1) \in \{0, 2\}^k \).

\section*{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 \( f : \mathcal{A}^2 \rightarrow \{0, 1, 2\} \) shown in Table 5.3, satisfies
\[
C_{\text{lin}}(\mathcal{N}_4, f)_{\mathbb{Z}_4^2} \leq \frac{2}{3} \\
C_{\text{lin}}(\mathcal{N}_4, f)_{\mathbb{Z}_2^2 \times \mathbb{Z}_2} = 1.
\]

(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 \( i \).)

\textbf{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,
\[
C_{\text{lin}}(\mathcal{N}, f)_{\mathbb{Z}_2^2 \times \mathbb{Z}_2} \leq 1. \tag{5.10}
\]

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 \( 1 \times 1 \) identity matrix), then \( \rho \) 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
\[
C_{\text{lin}}(\mathcal{N}, f)^{\mathbb{Z}_2 \times \mathbb{Z}_2} = 1.
\]

We now prove that $C_{\text{lin}}(\mathcal{N}, f)^{\mathbb{Z}_4} \leq 2/3$. 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 $\rho$ 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,
\[
4^n = |\mathcal{A}|^n \\
\geq |\{ yM : y \in \mathcal{A}^{2k} \}| \quad \text{[from } y \in \mathcal{A}^{2k} \implies yM \in \mathcal{A}^n]\]
\[
\geq \frac{|\mathcal{A}|^{2k}}{|\mathcal{K}|} \quad \text{[from } y^{(1)}, y^{(2)} \in \mathcal{A}^{2k}, y^{(1)}M = y^{(2)}M \implies y^{(1)} - y^{(2)} \in \mathcal{K}]}
\[
\geq \frac{|\mathcal{A}|^{2k}}{2^k} \quad \text{[from Lemma 5.3.1]}
\[
= 4^{3k/2}. \quad \text{[from } |\mathcal{A}| = 4]\]

Thus, $k/n \leq 2/3$, so $C_{\text{lin}}(\mathcal{N}_4, f)^{\mathbb{Z}_4} \leq \frac{2}{3}$. \qed
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 \( f : \mathcal{A}^s \rightarrow \mathcal{B} \) 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 \( \lambda \) satisfying \( \lambda < s \), matrix \( T \in \mathcal{A}^{s \times \lambda} \), and map \( g : \mathcal{A}^\lambda \rightarrow \mathcal{B} \) such that

\[
g(xT) = f(x) \quad \text{for each } x \in \mathcal{A}^s.
\]
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) \quad (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

$$\mathcal{M}^\perp = \{ y \in \mathcal{A}^s : xy^t = 0 \text{ for all } x \in \mathcal{M} \}$$

and let $\dim(\mathcal{M})$ denote the dimension of $\mathcal{M}$ over $\mathcal{A}$.

**Lemma 5.4.5.** 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 $f : \mathcal{A}^s \to \mathcal{B}$ 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 $d \in \mathcal{A}^s$,

$$f(d + x) \neq f(x).$$

---

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. [88, Theorem 7.5] or [89, Corollary 3.2.3]). An analogous result for orthogonal complements over inner product spaces is well known in linear algebra (e.g. [90, Theorem 5 on pg. 286]).
(The arithmetic in $ad + x$ and $d + x$ is performed component-wise over the corresponding $A$.)

Proof. (i) If $f$ is reducible, then there exists an integer $\lambda$ satisfying $\lambda < s$, matrix $T \in A^{s \times \lambda}$, and map $g : A^\lambda \rightarrow B$ such that

$$g(xT) = f(x) \text{ for each } x \in A^s. \quad (5.13)$$

Since $\lambda < s$, there exists a non-zero $d \in A^s$ such that $dT = 0$. Then for each $a \in A$ and each $x \in A^s$,

$$f(ad + x) = g((ad + x)T) = g(xT) = f(x). \quad (5.14)$$

Conversely, suppose that there exists a non-zero $d$ such that (5.14) holds for every $a \in A$ and every $x \in A^s$ and let $\mathcal{M}$ be the one-dimensional subspace of $A^s$ spanned by $d$. Then

$$f(t + x) = f(x) \text{ for every } t \in \mathcal{M}, x \in A^s. \quad (5.15)$$

Note that $\dim(\mathcal{M}^\perp) = s - 1$. Let $\lambda = s - 1$, let $T \in A^{s \times \lambda}$ 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

$$g : \mathcal{R}_T \rightarrow f(A^s)$$

as follows. For any $y \in \mathcal{R}_T$ such that $y = xT$ for $x \in A^s$, let

$$g(y) = g(xT) = f(x). \quad (5.16)$$
Note that if \( y = x^{(1)}T = x^{(2)}T \) for \( x^{(1)} \neq x^{(2)} \), then

\[
(x^{(1)} - x^{(2)})T = 0
\]

\[
x^{(1)} - x^{(2)} \in (\mathcal{M}^\perp)^\perp \quad \text{[from construction of } T]\]

\[
x^{(1)} - x^{(2)} \in \mathcal{M} \quad \text{[from Lemma 5.4.5]}
\]

\[
f(x^{(1)}) = f((x^{(1)} - x^{(2)}) + x^{(2)})
\]

\[
= f(x^{(2)}). \quad \text{[from (5.15)]}
\]

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 \( f(d + x) \neq f(x) \).

\[\blacksquare\]

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 \( a \in \mathcal{A} \),

\[
f(2a + x) = 2(2a + x)
\]

\[
= 2x \quad \text{[from } 4 = 0 \text{ in } \mathbb{Z}_4]\]

\[
= f(x)
\]

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 [73]. For
network computing, the next theorem shows that with non-reducible target functions there is no advantage to using linear coding over routing.\footnote{As a reminder, “network” here refers to single-receiver networks in the context of computing.}

**Theorem 5.4.8.** Let $\mathcal{N}$ be a network with target function $f : \mathcal{A}^s \rightarrow \mathcal{B}$ 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 
\[
C_{\text{lin}}(\mathcal{N}, f) = C_{\text{rout}}(\mathcal{N}, f).
\]

**Proof.** Since any routing code is in particular a linear code,
\[
C_{\text{lin}}(\mathcal{N}, f) \geq C_{\text{rout}}(\mathcal{N}, f).
\]

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 $\rho$ 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 $\rho$ 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 $\rho$ 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
where $A$ is a field and $f$ is not reducible, by Lemma 5.4.6(i), there exist $a \in A$ and $x \in A^s$ such that $ad \neq 0$ and

$$f(ad + x) \neq f(x). \quad (5.18)$$

In the case where $A$ 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 $A$ 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 $\rho$ must differ between two cases: (1) when the source messages are the columns of $u$, and (2) when the source 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

$$\sum_{i=1}^{s} u^{(i)} M(e)_i = \sum_{i=1}^{s} u^{(i)} M(e)_i + a \sum_{i=1}^{s} (w^{(i)} - y^{(i)}) M(e)_i \quad \text{[from (5.17)]}$$

$$= \sum_{i=1}^{s} v^{(i)} M(e)_i \quad (5.19)$$

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

$$|A|^{|r| - |C|} \geq |A|^{|k| - |I_C|}$$
and thus
\[ \frac{k}{n} \leq \frac{|C|}{|I_C|}. \]

Since the cut \( C \) is arbitrary, we conclude (using (5.4)) that
\[ \frac{k}{n} \leq \min_{C \in \lambda(N)} \frac{|C|}{|I_C|} = \mathcal{C}_{rout}(N, f). \]

Taking the supremum over all \((k, n)\) linear network codes that compute \( f \) in \( N \), we get
\[ \mathcal{C}_{lin}(N, f) \leq \mathcal{C}_{rout}(N, f). \]

\[ \blacksquare \]

**Figure 5.3:** Network \( N_{5,s} \) 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

\[ C_{\text{cod}}(\mathcal{N}, f) = C_{\text{rout}}(\mathcal{N}, f). \]

**Proof.** It follows from [73, Theorem 4.2] that for any single-receiver network $\mathcal{N}$ and the identity target function $f$, we have $C_{\text{cod}}(\mathcal{N}, f) = C_{\text{rout}}(\mathcal{N}, f)$. 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 $f : \mathcal{A}^s \rightarrow \mathcal{B}$, satisfies

\[ C_{\text{cod}}(\mathcal{N}_{5,s}, f) \geq \min \left\{ 1, \frac{1}{\log_{|\mathcal{A}|} |f(\mathcal{A}^s)|} \right\}. \]

**Proof.** Suppose

\[ \log_{|\mathcal{A}|} |f(\mathcal{A}^s)| < 1. \quad (5.20) \]

Let $k = n = 1$ and assume that each source node sends its message to node $v$. Let

\[ g : f(\mathcal{A}^s) \rightarrow \mathcal{A} \]
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 \( C_{\text{cod}}(\mathcal{N}_{5,s}, f) \geq 1 \).

Now suppose

\[
\log_{|\mathcal{A}|} |f(A^s)| \geq 1. \tag{5.21}
\]

Choose integers \( k \) and \( n \) such that

\[
\frac{1}{\log_{|\mathcal{A}|} |f(A^s)|} - \epsilon \leq \frac{k}{n} \leq \frac{1}{\log_{|\mathcal{A}|} |f(A^s)|}. \tag{5.22}
\]

Now choose an arbitrary injective map (which exists by (5.22))

\[
g : (f(A^s))^k \rightarrow \mathcal{A}^n.
\]

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

\[
\frac{k}{n} \geq \frac{1}{\log_{|\mathcal{A}|} |f(A^s)|} - \epsilon.
\]

Since \( \epsilon \) was arbitrary, it follows that the computing capacity \( C_{\text{cod}}(\mathcal{N}_{5,s}, f) \) is at least \( 1/\log_{|\mathcal{A}|} |f(A^s)| \).

\[\blacksquare\]

**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

\[
C_{\text{cod}}(\mathcal{N}, f) > C_{\text{lin}}(\mathcal{N}, f).
\]
Proof. If $\mathcal{N}$ is the network $\mathcal{N}_{5,s}$ shown in Figure 5.3 with alphabet $\mathcal{A}$, then

$$C_{\text{lin}}(\mathcal{N}, f) = \frac{1}{s} \quad \text{[from Theorem 5.4.8 and (5.4)]}$$

$$< \min \left\{ 1, \frac{1}{\log |\mathcal{A}| |f(\mathcal{A}^s)|} \right\} \quad \text{[from } s \geq 2 \text{ and } |f(\mathcal{A}^s)| < |\mathcal{A}|^s \text{]}$$

$$\leq C_{\text{cod}}(\mathcal{N}, f). \quad \text{[from Lemma 5.4.10]}$$

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 $f : \mathcal{A}^s \rightarrow \mathcal{B}$ is reducible, then there exists a network $\mathcal{N}$ such that

$$C_{\text{lin}}(\mathcal{N}, f) > C_{\text{rout}}(\mathcal{N}, f).$$

**Proof.** Since $f$ is reducible, there exist $\lambda < s$, a matrix $T \in \mathcal{A}^{s \times \lambda}$, and a map $g : \mathcal{A}^\lambda \rightarrow f(\mathcal{A}^s)$ such that

$$g(xT) = f(x) \quad \text{for every } x \in \mathcal{A}^s. \quad \text{[from Definition 5.2.1]} \quad (5.23)$$

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 $x^T$ and sends it to the receiver. Having received the $n$-dimensional vector $x^T$, the receiver computes

$$\psi(x^T) = g(x^T) \quad [\text{from } \psi = g]$$

$$= f(x). \quad [\text{from (5.23)}]$$

Thus there exists a linear code that computes $f$ in $\mathcal{N}$ with an achievable computing rate of

$$\frac{k}{n} = \frac{1}{\lambda}$$

$$> \frac{1}{s} \quad [\text{from } \lambda \leq s - 1]$$

$$= C_{\text{rout}}(\mathcal{N}) \quad [\text{from (5.4)}]$$

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 $C_{\text{lin}}(\mathcal{N}, f) = C_{\text{rout}}(\mathcal{N}, f)$ 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

$$C_{\text{lin}}(\mathcal{N}, f) = C_{\text{rout}}(\mathcal{N}, f) = 1.$$
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:

$$C_{\text{cod}}(\mathcal{N}, f) > C_{\text{lin}}(\mathcal{N}, f) > C_{\text{rout}}(\mathcal{N}, f).$$

**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

$$C_{\text{rout}}(\mathcal{N}, f) = 1/3. \quad \text{[from (5.4)]} \quad (5.24)$$

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

$$C_{\text{cod}}(\mathcal{N}, f) \geq 1. \quad (5.25)$$

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 $g(x_1, x_2) = x_1 x_2$ 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),

$$
\frac{k}{n} \leq C_{\text{lin}}(\mathcal{N}_{5,2}, g)
$$

$$
= C_{\text{rout}}(\mathcal{N}_{5,2}, g) \quad \text{[from Theorem 5.4.8]}
$$

$$
= 1/2. \quad \text{[from (5.4)]}
$$

Consequently,

$$
C_{\text{lin}}(\mathcal{N}, f) \leq 1/2. \quad (5.26)
$$

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

$$C_{\text{lin}}(\mathcal{N}, f) = 1/2.$$  \hspace{1cm} (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 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

$$\alpha(\sigma) = (\alpha(\sigma)_1, \alpha(\sigma)_2, \ldots, \alpha(\sigma)_c) \in \mathbb{F}_q^k$$

where $\alpha(\sigma)_i \in \mathbb{F}_q^n$ for each $i$. Likewise, the decoder $\psi$ 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:

$$\begin{cases} 
\sum_{\hat{e} \in \mathcal{E}_{i}(u)} \gamma^{(e)}_{\hat{e}} z_{\hat{e}} + \sum_{j=1}^{c} \beta^{(e)}_{j} \alpha(u)_j & \text{if } u \in S \\
\sum_{\hat{e} \in \mathcal{E}_{i}(u)} \gamma^{(e)}_{\hat{e}} z_{\hat{e}} & \text{otherwise}
\end{cases} \quad (5.28)$$

for some $\gamma^{(e)}_{\hat{e}}, \beta^{(e)}_{j} \in \mathbb{F}_q^n$ and we use a decoding function $\psi$ whose $j$-th component output $\psi_j$ is:

$$\sum_{e \in \mathcal{E}_{i}(\rho)} \delta^{(e)}_{j} z_e \quad \text{for all } j \in \{1, 2, \ldots, c\} \quad (5.29)$$

for certain $\delta^{(e)}_{j} \in \mathbb{F}_q^n$. Here we view each $h^{(e)}$ as a function of the in-edges to $e$ and

---

6The definition of “linear target function” was given in Table 5.2.
the source messages generated by \( u \) and we view \( \psi \) 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 \( \sigma \) 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 \( y^{(e_j)}_{e_i} \) 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 \( \tau \in \{1, 2, \ldots, s\} \), let \( A(\tau)(x) \) be a \( c \times |\mathcal{E}| \) matrix \( A(\tau)(x) \), given by

\[
(A(\tau)(x))_{i,j} = \begin{cases} 
  x_i^{(e_j)} & \text{if } e_j \in \mathcal{E}_o(\sigma_{\tau}) \\
  0 & \text{otherwise}
\end{cases}
\]  

(ii) Let \( F(y) \) be a \( |\mathcal{E}| \times |\mathcal{E}| \) matrix, given by

\[
(F(y))_{i,j} = \begin{cases} 
  y_{e_i}^{(e_j)} & \text{if } e_i, e_j \in \mathcal{E} \text{ and head}(e_i) = \text{tail}(e_j) \\
  0 & \text{otherwise}
\end{cases}
\]

(iii) Let \( B(w) \) be a \( c \times |\mathcal{E}| \) matrix, given by

\[
(B(w))_{i,j} = \begin{cases} 
  w_i^{(e_j)} & \text{if } e_j \in \mathcal{E}_i(\rho) \\
  0 & \text{otherwise}
\end{cases}
\]

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 [83, Lemma 2]) The matrix \( I - F(y) \) is invertible over the ring \( \mathbb{F}_q[y] \).
Lemma 5.5.2. (Koetter-Médard [83, 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 [91, Theorem 1.2]) Let $F$ be an arbitrary field, and let $g = g(x_1, \ldots, x_m)$ be a polynomial in $F[x_1, \ldots, x_m]$. 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 $F$ 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

$$M_\tau(x, y, w) = A_\tau(x)(I - F(y))^{-1}B(w)^t \quad (5.33)$$

where the components of $M_\tau(x, y, w)$ are viewed as lying in $F_q[x, y, w]$.

Lemma 5.5.4. If for all $\tau \in \{1, 2, \ldots, s\}$,

$$\det(M_\tau(x, y, w)) \neq 0$$

in the ring $F_q[x, y, w]$, then there exists an integer $n > 0$ and vectors $\beta, \gamma, \delta$ over $F_q^n$ 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 $F_q^n$.

Proof. The quantity

$$\det \left( \prod_{\tau=1}^{s} M_\tau(x, y, w) \right)$$

is a nonzero polynomial in $F_q[x, y, w]$ and therefore also in $F_q^n[x, y, w]$ 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 $\beta, \gamma, \delta$ (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 \( \tau \in \{1, 2, \ldots, s\} \)

\[
\det (M_{\tau}(\beta, \gamma, \delta)) \neq 0.
\]

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

\[
C_{\text{cod}}(\mathcal{N}, f) \leq \min_{C \in \Lambda(\mathcal{N})} |C|.
\]

**Proof.** The same argument is used as in the proof of Lemma 5.2.11, except instead of using \( R_{IC,f} \geq 2 \), we use the fact that \( R_{IC,f} = |\mathcal{A}| \) for linear target functions.

**Theorem 5.5.6.** If \( \mathcal{N} \) is a network with a linear target function \( f \) over finite field \( \mathbb{F}_{q^n} \), then

\[
C_{\text{lin}}(\mathcal{N}, f) = \min_{C \in \Lambda(\mathcal{N})} |C|.
\]

**Proof.** We have

\[
C_{\text{lin}}(\mathcal{N}, f) \leq C_{\text{cod}}(\mathcal{N}, f)
\]

\[
\leq \min_{C \in \Lambda(\mathcal{N})} |C|. \tag{from Lemma 5.5.5}
\]

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 \( c = \min_{C \in \Lambda(\mathcal{N})} |C| \).

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 \( \sigma_x \) and the
receiver $\rho$ has at least $c$ edges, it follows from [83, Theorem 2]\(^7\) that $\det(M_\tau(x, y, w)) \neq 0$ for every $\tau \in \{1, 2, \ldots, s\}$. From Lemma 5.5.4, there exists an integer $n > 0$ and vectors $\beta, \gamma, \delta$ 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

$$\hat{A}_\tau(\beta) = a_\tau (M_\tau(\beta, \gamma, \delta))^{-1}A_\tau(\beta). \quad (5.35)$$

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 $\rho$ is

$$\psi = \sum_{\tau=1}^{s}\alpha(\sigma_\tau) \hat{A}_\tau(\beta)(I - F(\gamma))^{-1}B(\delta)^t$$  

[from Lemma 5.5.2]

$$= \sum_{\tau=1}^{s}\alpha(\sigma_\tau) a_\tau (M_\tau(\beta, \gamma, \delta))^{-1}A_\tau(\beta)(I - F(\gamma))^{-1}B(\delta)^t$$  

[from (5.35)]

$$= \sum_{\tau=1}^{s}a_\tau \alpha(\sigma_\tau)$$  

[from (5.33)]

$$= (f(\alpha(\sigma_1)_1, \ldots, \alpha(\sigma_s)_1), \ldots, f(\alpha(\sigma_1)_c, \ldots, \alpha(\sigma_s)_c))$$

which proves that the linear code achieves a computing rate of $c$. \hfill \blacksquare

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 $(1) \implies (3)$ in [83, Theorem 2].
field \( \mathbb{F}_q \), then

\[
C_{\text{lin}}(\mathcal{N}, f) = C_{\text{cod}}(\mathcal{N}, f) \leq s \cdot C_{\text{rout}}(\mathcal{N}, f).
\]

**Proof.**

\[
s \cdot C_{\text{rout}}(\mathcal{N}, f) \geq \min_{C \in \Lambda(\mathcal{N})} |C| \quad \text{[from (5.4) and Theorem 5.2.10]}
\]

\[
\geq C_{\text{cod}}(\mathcal{N}, f) \quad \text{[from Lemma 5.5.5]}
\]

\[
\geq C_{\text{lin}}(\mathcal{N}, f)
\]

\[
= \min_{C \in \Lambda(\mathcal{N})} |C|. \quad \text{[from Theorem 5.5.6]}
\]

\( \square \)

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 \( f : \mathcal{A}^s \rightarrow \mathcal{A} \) is linear over finite field \( \mathbb{F}_q \), then there exists a network \( \mathcal{N} \) with \( s \) sources, such that

\[
C_{\text{lin}}(\mathcal{N}, f) = s \cdot C_{\text{rout}}(\mathcal{N}, f).
\]

**Proof.** Let \( \mathcal{N} \) denote the network \( \mathcal{N}_{5,s} \) shown in Figure 5.3. Then

\[
C_{\text{lin}}(\mathcal{N}, f) = 1 \quad \text{[from Theorem 5.5.6]}
\]

\[
C_{\text{rout}}(\mathcal{N}, f) = C_{\text{rout}}(\mathcal{N}) \quad \text{[from Theorem 5.2.10]}
\]

\[
= 1/s. \quad \text{[from (5.4)]}
\]

\( \square \)
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.

![Diagram](image)

Figure 5.5: The butterfly network and its reverse $N_6$.

The network $N_6$ shown in Figure 5.5(b) is called the reverse butterfly network. It has $S = \{\sigma_1, \sigma_2\}$, receiver node $\rho$, 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 $N_6$ with alphabet $A = \{0, 1, \ldots, q - 1\}$ and arithmetic sum target function $f : A^2 \rightarrow \{0, 1, \ldots, 2(q - 1)\}$ are

$$C_{\text{rout}}(N_6, f) = C_{\text{lin}}(N_6, f) = 1.$$ 

**Proof.** We have

$$C_{\text{lin}}(N_6, f) = C_{\text{rout}}(N_6) \quad \text{[from Theorem 5.4.8]}$$

$$= 1. \quad \text{[from (5.4)]}$$
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 $\sigma_1$ down the left side and $\sigma_2$ 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 $N_6$ with alphabet $A = \{0, 1, \ldots, q-1\}$ and arithmetic sum target function $f : A^2 \rightarrow \{0, 1, \ldots, 2(q-1)\}$ is

$$C_{\text{cod}}(N_6, f) = \frac{2}{\log_q (2q - 1)}.$$  

Remark 5.6.4. The computing capacity $C_{\text{cod}}(N_6, f)$ obtained in Theorem 5.6.3 is a function of the coding alphabet $A$ (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 [68]. For the reverse butterfly network, if, for example, $q = 2$, then $C_{\text{cod}}(N_6, f)$ is approximately equal to 1.26 and increases asymptotically to 2 as $q \rightarrow \infty$.

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 [68] 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 [68] 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 $N_6$ with $A = \{0, 1, \ldots, q - 1\}$ and the mod $q$ sum target function $f$ is

$$C_{\text{cod}}(N_6, f) = 2.$$

**Proof.** The upper bound of 2 on $C_{\text{cod}}(N_6, f)$ follows from [85, Theorem II.1]. To establish the achievability part, let $k = 2$ and $n = 1$. Consider the code shown in Figure 5.6, where ‘⊕’ indicates the mod $q$ sum. The receiver node $\rho$ 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

$$C_{\text{cod}}(N_6, f) \leq 2 / \log_q(2q - 1).$$

[from [85, Theorem II.1]]
To establish the lower bound, we use the fact that the arithmetic sum of two elements from \( A = \{0, 1, \ldots, q - 1\} \) is equal to their mod 2\( q - 1 \) sum. Let the reverse butterfly network have alphabet \( \hat{A} = \{0, 1, \ldots, 2(q - 1)\} \). From Lemma 5.6.7 (with alphabet \( \hat{A} \)), the mod 2\( q - 1 \) sum target function can be computed in \( N_6 \) at rate 2. Indeed for every \( n \geq 1 \), there exists a \((2n, n)\) network code that computes the mod 2\( q - 1 \) sum target function at rate 2. So for the remainder of this proof, let \( k = 2n \). Furthermore, every such code using \( \hat{A} \) can be “simulated” using \( A \) by a corresponding \((2n, \lceil n \log_q (2q - 1) \rceil)\) code for computing the mod 2\( q - 1 \) 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 \( g : \hat{A}^n \rightarrow A^{n'} \) be an injection (which exists since \( q^{n'} \geq (2q - 1)^n \)) and let the function \( g^{-1} \) denote the inverse of \( g \) on its image \( g(\hat{A}) \). Let \( x^{(1)}, x^{(2)} \) denote the first and last, respectively, halves of the message vector \( \alpha(\sigma_1) \in A^{2n} \), where we view \( x^{(1)} \) and \( x^{(2)} \) as lying in \( \hat{A}^n \) (since \( A \subseteq \hat{A} \)). The corresponding vectors \( y^{(1)}, y^{(2)} \) for the source \( \sigma_2 \) are similarly defined.

Figure 5.7 illustrates a \((2n, n')\) code for network \( N_6 \) using alphabet \( A \) where \( \oplus \) denotes the mod 2\( q - 1 \) sum. Each of the nodes in \( N_6 \) converts each of the received vectors over \( A \) into a vector over \( \hat{A} \) using the function \( g^{-1} \), then performs coding in Figure 5.6 over \( \hat{A} \), and finally converts the result back to \( A \). Similarly, the receiver node \( T \) computes the component-wise arithmetic sum of the source message vectors \( \alpha(\sigma_1) \) and \( \alpha(\sigma_2) \) using

\[
\alpha(\sigma_1) + \alpha(\sigma_2) = (g\left( g(x^{(1)} \oplus x^{(2)} \oplus y^{(1)} \oplus y^{(2)}) \right) \oplus g\left( g(x^{(2)} \oplus y^{(2)}) \right)),
\]

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
Figure 5.7: The reverse butterfly network with a code that computes the arithmetic sum target function. ‘⊕’ denotes mod \(2q - 1\) 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


