CONVERGENCE TO AND QUALITY OF EQUILIBRIA IN DISTRIBUTED SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Damon Mosk-Aoyama
December 2008

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Tim Roughgarden) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Ashish Goel)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Amin Saberi)

Approved for the Stanford University Committee on Graduate Studies.

Abstract

A natural property of a distributed system to study is its set of stable points, or equilibria. In a system comprising individual agents where the communication between the agents is local, determining whether the system converges over time to an equilibrium is important for understanding whether a global consensus is reached by the agents. A proof that such a system converges to a fixed point shows that the dynamic process executed by the agents can be considered a local computation that produces a global output. In a distributed system with selfish users, an equilibrium in which no user can benefit via a unilateral deviation is an expected outcome of the system. From the point of view of a system designer, a comparison between the aggregate welfare of all the users at an equilibrium and that at an optimal state provides a useful measure of system performance.

In this work, we study the equilibria of several distributed systems. One setting that we consider is a network in which nodes obtain global information by repeatedly communicating with their neighbors. We study a gossip algorithm of Deb and Médard for information dissemination in the network. For a general network topology, we provide an upper bound on the time required for the network to converge to a state in which every node has every message to be disseminated. We also show that if each node begins with a positive number and executes a simple gossip algorithm, the system converges to a state in which every node has an accurate estimate of the sum of the numbers. Using this approximate summation algorithm as a subroutine, we develop a simple distributed algorithm for a class of convex optimization problems. In this algorithm, the nodes repeatedly execute a gradient ascent procedure that converges to an approximate solution to the convex optimization problem.

Another setting that we consider involves the production of a good that is to be consumed by multiple users. Each user requests a quantity of the good. The system

produces enough of the good to satisfy all the requests, and recovers the cost of producing the total amount requested from the users by assigning a cost share to each user. The utility of a user is a function of the user's requested quantity and assigned cost share. When users act to maximize their individual utilities, a strategic game is a natural model of the system. Under standard assumptions on the utility functions and a quadratic cost function, and two well-known cost sharing methods, average cost pricing and serial cost sharing, this game is guaranteed to have a Nash equilibrium in which no user can benefit by requesting a different quantity. We show how to determine the worst-case ratio between the aggregate welfare of the users under a Nash equilibrium and the optimal aggregate welfare for each cost sharing method in a class that interpolates between these two methods.

Acknowledgments

I would first like to thank Tim Roughgarden. His suggestions of problems to work on and ideas about how to approach problems have been very helpful to me. He agreed to serve as my dissertation adviser at a time when I did not see a clear path towards a dissertation, and as such I am indebted to him.

Devavrat Shah was a de facto adviser to me through much of the work in this dissertation. I greatly enjoyed working with him, starting from when we shared an office. I would like to thank him for all of the invaluable help he has given me.

Earlier in my time at Stanford, Mihalis Yannakakis and Balaji Prabhakar served as advisers to me. I would like to thank both of them for their help and the opportunities to learn from them. I thank Ashish Goel and Amin Saberi, who offered me useful advice over the years, and who served on my reading committee.

Going back to my time at MIT, I would like to thank David Karger and Lynn Andrea Stein, who introduced me to research. They were very generous with their time, and were instrumental in the development of my interest in research.

I would like to thank many fellow students and group members: Zoë Abrams, Mohsen Bayati, Aaron Bradley, Peerapong Dhangwatnotai, Shahar Dobzinski, Shaddin Dughmi, Vivek Farias, Phil Fong, Roberta Fracchia, Martin Hoefer, Wathanyoo Khaisongkram, Huan Liu, Yi Lu, Athina Markopoulou, Chandra Nair, Mayank Sharma, Anthony Man-Cho So, Mukund Sundararajan, Mei Wang, and Qiqi Yan. I am grateful for their companionship and support over the years.

Finally, I would like to thank my family, especially my parents and my brother. Their support throughout my time at Stanford was invaluable to me.

Contents

List of Figures

Chapter 1

Introduction

A common contemporary model of computation is a distributed system. In a distributed system, there is a collection of independent entities, each of which has its own computational and storage resources. When these entities communicate, they can collaborate to achieve some global objective, or compete to further their own self interests.

Recent interest in distributed systems is motivated by several technological developments. One is the emergence of modern networks such as sensor, ad-hoc wireless, and peer-to-peer networks. The nodes in these networks often operate under constraints imposed by limited power, computation, and communication. They may be unreliable, and as such may fail during a computational process. Nodes may be added to the network while it is in operation, thus changing the topology of the network. These conditions make it difficult to establish infrastructure for coordinated centralized computation. A natural approach to computation in these networks, therefore, is to design distributed algorithms that the nodes can execute to solve a problem through cooperation.

A second development that has stimulated interest in distributed systems is the increasing importance of the Internet. The Internet is a decentralized system in which the various components are controlled by different users. These users all have their own goals, and so it is reasonable to expect that they will act independently in their own best interests. In the absence of any mechanism for dictating the behavior of the users, a system designer must accept the competition that arises when the goals of different users conflict. To cope with this competitive environment, it is useful to understand the outcomes that could result from the selfish behavior of the users, which in turn can inform system design.

This thesis studies several problems that arise in distributed systems. Three of these problems involve the cooperative perspective, in which we assume that we can specify the behavior of nodes in a network, and our goal is to design distributed algorithms that solve computational problems by local communication among nodes. In the fourth problem, we take the competitive perspective, and try to understand the loss in efficiency that occurs in the production of a good due to the selfish behavior of users under different methods of sharing the cost of production among the users.

1.1 Distributed Systems

We model a distributed system by a network for the algorithmic problems that we study. Each node in the network has some computational resources, and can communicate with other nodes. The problem inputs are distributed among the nodes in some way at the outset of the computation. Through repeated communication and computation at the individual nodes, the nodes cooperate to compute a function of the inputs. The output of the function is stored at the nodes at the end of the computation.

The communication links in the network are represented by an undirected graph $G = (V, E)$ with $|V| = n$ vertices. Each node in the network corresponds to a vertex in the graph. We assume that the nodes are numbered arbitrarily so that $V = \{1, \ldots, n\}$. Two nodes i and j can communicate with each other if and only if $(i, j) \in E$. Thus, the edges in the graph correspond to communication links in the network. As the graph is undirected, we are assuming that node i can send a message to node j if and only if j can send a message to i.

We assume that a node in the network does not have complete knowledge of the communication graph G; this is inspired by technological considerations in modern networks. Each node in the network knows only its neighbors in the graph, and can contact any neighbor to initiate a communication. As a result, each node has a local view of the network as a whole, and information from nodes not in this local view can only reach the node through its neighbors.

In this setting, a natural approach to distributed computation is a *gossip* algorithm. Gossip algorithms are useful for achieving fault-tolerant and scalable distributed computations in large networks. In a gossip algorithm, each node repeatedly initiates communication with a small number of neighbors in the network, typically chosen at random,

and exchanges information with those neighbors. Over the course of the computation, the state at each node evolves so that, eventually, the outputs of the function computed can be found at the nodes of the network.

The distributed algorithms that we consider all converge to an equilibrium state. If the nodes execute one of the algorithms for a sufficiently long time, then each node eventually reaches the same state. In this equilibrium state, every node has the output of the computation.

As previously mentioned, we also study a distributed system in which the users act independently and selfishly. In this setting, we assume that there are n users who are interested in a good. Each user requests a quantity of the good. All of the requests are satisfied, so that the good is produced in an amount equal to the sum of the requested quantities at some cost. The production cost is then collected from the users by an assignment of a cost share to each user.

The utility of an individual user is a function of the amount of the good the user requests and the cost share of the user. As the cost share of a user may depend on the quantities requested by the other users, this is a competitive environment. We assume that all the users choose quantities to maximize their own utilities, subject to the actions of the other users. We are interested in the equilibrium outcomes of the system: collections of requested quantities under which all the users are simultaneously maximizing their utilities.

1.2 Problems Studied

In this thesis, we study three algorithmic problems in a distributed setting. The first is the *information dissemination* or *information spreading* problem, which involves the transmission of messages through the network. Initially, each node $i \in V$ has a message m_i. Through communication across edges in E, nodes can transmit messages to each other. The goal is for every node in the network to obtain all of the n messages. An algorithm for this problem specifies the mechanism by which nodes choose communication partners, and the protocol that determines the data transmitted between communicating nodes.

The second problem we study is that of *computing separable functions*. We assume that there is a function $f_i(x_i)$ associated with each node $i \in V$, and we consider the function

$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} f_i(x_i)$. This function f is called a separable function because it can be decomposed into a sum of functions of individual variables. The inputs to the algorithm are the values $x_1, \ldots, x_n$, with each node i having the value x_i at the start of the computation. An algorithm's goal is to compute the value $f(x_1, \ldots, x_n)$ of the function f for the input values x_i.

The third problem that we study is *convex optimization*. We associate with each node $i \in V$ a variable x_i, so that an n-dimensional vector x contains the variables for all the nodes. The goal is to find a vector x^* of variable values that minimizes a convex separable function $f(x_1, \ldots, x_n)$ subject to the equality constraints $Ax = b$, where A is a matrix with m rows, corresponding to the linear constraints, and b is a m-dimensional vector.

In the competitive environment involving the production of the good, we consider the *price of anarchy* of a cost sharing method. The cost function of the system specifies the cost of producing each amount of the good. A cost sharing method is a function that takes as inputs the cost function and the requested quantities of the users, and assigns a cost share to each user. For any cost sharing method, the selfish behavior of the users leads to a strategic game. Informally, the price of anarchy of a cost sharing method is the worst-case ratio, over a class of utility functions of the users, between the aggregate surplus of the system under a Nash equilibrium and the maximum aggregate surplus achievable by any collection of requested quantities. We study the price of anarchy of any cost sharing method in a class that interpolates between two well-known cost sharing methods, average cost pricing and serial cost sharing, under a quadratic cost function.

1.3 Our Contributions

All of the gossip algorithms that we consider in this thesis are randomized. As such, the performance guarantees for these algorithms are probabilistic in nature. We first analyze a gossip algorithm for information dissemination based on network coding that was originally proposed by Deb and Médard [11, 12]. We provide an upper bound on the amount of time required for every node in a network of arbitrary topology to receive every message under the algorithm. The bound is stated in terms of spectral properties of the communication network.

Next, we consider the problem of computing separable functions. We present a randomized algorithm that approximately computes the value $f(x_1, \ldots, x_n)$. The algorithm

takes as input an error parameter ϵ, and produces an estimate of $f(x_1, \ldots, x_n)$ that is accurate up to a factor of $1 \pm \epsilon$.

This algorithm is based on a reduction of the problem of computing separable functions to the information dissemination problem. To obtain a complete algorithm for approximately computing separable functions, we employ a simple gossip algorithm for information dissemination. We provide an analysis of this algorithm that again yields an upper bound on the time required for every node to receive every message in terms of spectral properties of the communication network. The upper bound on the amount of time required for the algorithm for computing separable functions to obtain an estimate of the function value with a multiplicative error of $1 \pm \epsilon$ depends on this bound on the information dissemination time.

Building on the algorithm for approximately computing separable functions, we then turn to the convex optimization problem. We show how to use dual gradient ascent to reduce the class of convex optimization problems described above to the problem of computing separable functions. This leads to an iterative algorithm for obtaining an approximate solution to the convex program. The solution has additive error with respect to the optimal objective function value and multiplicative error with respect to the linear equality constraints. As the algorithm for convex optimization uses the algorithm for computing separable functions as a subroutine, its running time depends on that of the algorithm for computing separable functions, as well as additional parameters that measure the variation in curvature of the objective function.

For the price of anarchy problem concerning the production game, we study a class of cost sharing methods that interpolates between the two most well-known cost sharing methods, average cost pricing and serial cost sharing. We develop a method of analysis that applies to any cost sharing method in this class when the cost function is quadratic. Using this analysis, we determine the price of anarchy of every cost sharing method in the class.

1.4 Related Work

In this section, we describe prior work on the problems we study. The presentation is organized according to the different problems.

Information Dissemination

Gossip algorithms for disseminating a message to all nodes in a complete graph of communication links in which communication partners are chosen uniformly at random have been studied for some time [16, 40, 14]. Karp, Schindelhauer, Shenker, and Vöcking presented a gossip algorithm that disseminates a single message to all n nodes in a graph in $O(\log n)$ time with high probability [21]. For other related results, we refer the reader to [42, 24, 25], and to the recent work on the spread of epidemics in a network of Ganesh, Massoulié, and Towsley [18], and of Berger, Borgs, Chayes, and Saberi [5].

Chapter 2 provides an analysis of a gossip algorithm for information dissemination based on network coding that was originally proposed by Deb and Médard [11, 12]. Deb and Médard showed that, on a complete graph of communication links, the algorithm simultaneously transmits n messages to all the nodes in $O(n)$ time with probability $1 - O(1/n)$. This improves upon the $O(n \log n)$ bound that is achieved by a sequential dissemination, one message at a time, of the n messages. The algorithm can be executed on an arbitrary communication graph, but the method of analysis does not extend beyond complete graphs. In Chapter 2, we analyze this algorithm for an arbitrary connected communication graph, and obtain an upper bound on the time required to disseminate the n messages in terms of spectral properties of the graph.

The algorithm for approximately computing separable functions that we present in Chapter 3 uses as a subroutine a simple gossip algorithm for information dissemination. To obtain an upper bound on the running time of the algorithm for computing separable functions, we analyze the time required for the information dissemination algorithm to transmit n messages to n nodes for arbitrary connected communication graphs and non-uniform random choices of communication partners.

Computation of Separable Functions

Algorithms for computing the number of distinct elements in a multiset or data stream [15, 3] can be adapted to compute separable functions using information spreading [10]. We are not aware, however, of a previous analysis of the amount of time required for these algorithms to achieve a certain accuracy in the estimates of the function value in the gossip setting that we study.

A related problem to that of computing separable functions is that of computing

averages. There are a number of existing approaches to the distributed computation of averages, which is a special case of the problem of reaching agreement or consensus among processors via a distributed computation. Because the average of a collection of numbers at the n nodes in a network is the ratio between the sum of those numbers and n, averaging algorithms cannot be extended in general to compute arbitrary separable functions when the nodes do not know the exact number of nodes in the network. On the other hand, an algorithm for computing separable functions can be used to compute averages by separately computing the sum of the input numbers, and the number of nodes in the graph (using one as the input at each node). Thus, distributed summation seems to be a strictly more general problem than distributed averaging.

Distributed algorithms for reaching consensus under appropriate conditions have been known since the classical work of Tsitsiklis [49] and Tsitsiklis, Bertsekas, and Athans [50] (see also the book by Bertsekas and Tsitsiklis [6]). Iterative load-balancing schemes [6, 41], in which processors send jobs to each other in order to balance the amount of work at the different processors, can be considered distributed algorithms that average the load in a system.

Recently, Kempe, Dobra, and Gehrke gave a randomized iterative gossip algorithm for averaging with optimal averaging time [23]. This result was restricted to complete graphs. The algorithm requires that the nodes begin the computation in an asymmetric initial state in order to compute separable functions, a requirement that may not be convenient for large networks that do not have centralized agents for global coordination. Subsequently, Boyd, Ghosh, Prabhakar, and Shah presented a simpler iterative gossip algorithm for averaging that addresses some of the limitations of the Kempe et al. algorithm [7]. Specifically, the algorithm and its analysis are applicable to arbitrary graph topologies. Boyd et al. showed a connection between the averaging time of the algorithm and the mixing time of an appropriate random walk on the graph representing the network. They also found an optimal averaging algorithm as a solution to a semi-definite program.

Convex Optimization

The design of distributed algorithms for convex minimization with linear constraints has been of interest since the early 1960s. The essence of the work before the mid-1980s is well documented in the book by Rockafellar [44]. Rockafellar describes distributed algorithms

for *monotropic programs*, which are separable convex minimization problems with linear constraints. These algorithms leverage the decomposable structure of the Lagrange dual problem arising from the separable primal objective. This structure has also been used to design parallel and asynchronous algorithms for monotropic programs; see the book by Bertsekas and Tsitsiklis [6] for further details.

There are several network resource allocation problems studied in the literature that are special cases of monotropic programs. Gallager [17] developed a distributed algorithm for the special case of a network routing problem. Kelly, Maulloo, and Tan [22] used the known distributed algorithmic solutions for monotropic programs to explain the congestion control protocols for a resource allocation problem. See also Garg and Young [19] for similar results that emphasize the rate of convergence to an optimal solution, and the book by Srikant [48] for further work on congestion control.

Flow control also serves as the motivation for the work of Bartal, Byers, and Raz [4] on distributed algorithms for positive linear programming (building on earlier work by Papadimitriou and Yannakakis [39] and Luby and Nisan [29]). In this model, there is a primal agent for each primal variable and a dual agent for each dual variable (or primal constraint). In [4], direct communication is required between a dual agent and all of the primal agents appearing in the corresponding constraint.

Recently, Neely developed an algorithm for convex minimization [38] that solves a more general convex optimization problem than the one we study in Chapter 4. It requires the nodes in the network to construct a shortest path tree, which pushes it outside our computational model, and to solve convex programs involving local variables.

Price of Anarchy in Cost Sharing

The production game that we study in Chapter 5 is a fundamental cost-sharing problem; see the paper by Moulin [35] for many references to other work related to this game. Moulin determines the price of anarchy of average cost pricing and of serial cost sharing when the cost function is quadratic, but uses different analyses for these two cost sharing methods. In Chapter 5, we provide a unified analysis for quadratic cost functions that applies to both of these cost sharing methods, as well as other cost sharing methods in a class that interpolates between these two. This analysis recovers the results of Moulin for average cost pricing and serial cost sharing, and also determines the price of anarchy of

any cost sharing method in the broader class.

1.5 Organization

The subsequent chapters are organized as follows. Chapter 2 presents the analysis of the gossip algorithm for information dissemination based on network coding. In Chapter 3, we describe and analyze the algorithm for approximately computing separable functions. Next, we use this algorithm to develop the alogrithm for approximate convex optimization in Chapter 4. Chapter 5 covers the analysis that determines the price of anarchy of a cost sharing method in the production game. We provide concluding remarks in Chapter 6.

1.6 Notation

We use the following notation throughout this thesis. We write $\mathbf{R}$ to denote the real numbers, $\mathbf{R}_+$ to denote the non-negative real numbers, and $\mathbf{R}_{++}$ to denote the positive real numbers. Similarly, $\mathbf{Z}$ denotes the integers and $\mathbf{Z}_+$ denotes the non-negative integers. For two functions $f, g : \mathbf{Z}_+ \to \mathbf{R}_+$, we write $f(n) \sim g(n)$ if $\lim_{n\to\infty} f(n)/g(n) = 1$.

For a matrix A, A^T denotes the transpose of A. We consider vectors to be column vectors, so that v^T denotes the row vector corresponding to a vector v.

1.7 Bibliographic Notes

Chapter 2 is joint work with Devavrat Shah and appeared in [33]. Chapter 3 is also joint work with Devavrat Shah, and appeared in preliminary form in [32], and in complete form in [34]. An announcement of the results of Chapter 4, which is joint work with Tim Roughgarden and Devavrat Shah, appeared in [31]. Chapter 5 is joint work with Tim Roughgarden.

Chapter 2

Information Dissemination via Network Coding

In this chapter, we study the problem of information dissemination (or information spreading) through the use of network coding in the gossip setting for arbitrary graphs. Network coding is a technique that has been shown to provide significant performance improvements in the transmission of messages in the context of multicasting. It is based on the idea that when a set of messages must be transmitted from a collection of sources to a collection of sinks in a network, it may not be optimal for the intermediate nodes on a path from a source to a sink to forward each message in its original form. Instead, it may be beneficial to combine the different messages in some way, thus creating new messages. The network code must be designed so that the sinks can recover the original messages from the coded messages that they receive.

Network coding has been studied in a number of recent papers, such as [1, 28, 27]. We study here a gossip algorithm, originally proposed by Deb and Médard [11, 12], for information dissemination that uses network coding. This algorithm operates on messages that are vectors over a finite field. When a node sends a coded message to another node, the coded message is a random linear combination of the messages the sending node possesses, along with a code vector. As the algorithm proceeds, the linear subspace spanned by the code vectors at any node grows. Once the dimension of this subspace is sufficiently large, the node can decode the coded messages to recover the original messages.

Consider a complete communication graph on n nodes, in which any node can communicate with any other node. Suppose that, whenever a node chooses another node to contact for the purpose of communicating messages, it selects every other node with equal probability. In this setting, the algorithm of Deb and Médard transmits n messages to all the nodes in $O(n)$ time with probability $1 - O(1/n)$, improving upon the $\Theta(n \log n)$ time required for a sequential dissemination of n messages using the randomized gossip algorithm of [21].

While the algorithm of Deb and Médard can be executed in any communication graph, the analysis of it does not extend to an arbitrary graph. We analyze this algorithm for an arbitrary connected communication graph, and obtain an upper bound on the running time in terms of spectral properties (or sparse cuts) of the graph. This result provides insight into how the graph topology affects the performance of the algorithm.

2.1 Model and Overview

Consider an undirected communication graph $G = (V, E)$, with $V = \{1, \ldots, n\}$ and the edges in E representing communication links between nodes. Each node $i \in V$ has a message m_i. We seek a communication protocol that can be used to disseminate all of the messages to each of the n nodes.

To model some of the resource constraints on the nodes, we impose a *transmitter gossip* constraint on node communication. Each node is allowed to contact at most one other node at a particular time for communication. However, a node can be contacted by multiple nodes simultaneously.

A time model determines when nodes in the network communicate with each other. For the gossip algorithms in this chapter and in Chapter 3, we consider both a synchronous and an asynchronous time model. These models are defined as follows.

Definition 2.1 (Synchronous time model). Time is measured in time slots or rounds, which are common to all nodes in the network. In any time slot, each node contacts one neighbor to initiate a communication. The choice by a node i of which node to contact can be made randomly, but any random choice must be independent of the choices made by all other nodes $j \neq i$. The gossip constraint governs the simultaneous communication among the nodes.

Definition 2.2 (Asynchronous time model). In this model, time is discretized according to the ticks of various clocks. Each node has an independent clock that ticks according to a Poisson process of rate 1. When a node's clock ticks, it chooses one neighbor (possibly at random), and contacts that neighbor.

Equivalently, there is a global clock that ticks according to a Poisson process of rate n. Let R_k denote the time corresponding to the kth clock tick. Then, the inter-clock-tick times $\{R_{k+1} - R_k\}$ are independent and identically distributed exponential random variables of rate n. On each tick of the global clock, a node a_k in the network is chosen uniformly at random, and we consider the global clock tick to be a tick of the clock at the node a_k.

We measure the running times of gossip algorithms in absolute time, which is the number of time slots in the synchronous time model, and is (on average) the number of global clock ticks divided by n in the asynchronous time model. The relationship between clock ticks in the asynchronous model and absolute time is further characterized by the following lemma and corollary.

Lemma 2.1. *For any $k \geq 1$, let $W_1, \ldots, W_k$ be independent and identically distributed exponential random variables with rate λ. Let*

$$A_k = \frac{1}{k} \sum_{i=1}^{k} W_i.$$

Then, $E[A_k] = \lambda^{-1}$, and for any $\epsilon \in (0, 1/2)$,

$$\Pr\left(\left|A_k - \frac{1}{\lambda}\right| \geq \frac{\epsilon}{\lambda}\right) \leq 2\exp\left(-\frac{\epsilon^2 k}{3}\right). \tag{2.1}$$

Proof. By definition,

$$\begin{aligned} E[A_k] &= \frac{1}{k} \sum_{j=1}^{k} \lambda^{-1} \\ &= \lambda^{-1}. \end{aligned}$$

The inequality in (2.1) follows directly from Cramér's Theorem (see [13], pp. 30, 35) and properties of exponential random variables. □

A direct implication of Lemma 2.1 is the following corollary.

Corollary 2.2. *For $k \geq 1$,*

$$E[R_k] = \frac{k}{n}.$$

Further, for any $\epsilon \in (0, 1/2)$,

$$\Pr\left(\left|R_k - \frac{k}{n}\right| \geq \frac{\epsilon k}{n}\right) \leq 2\exp\left(-\frac{\epsilon^2 k}{3}\right).$$

To measure the performance of a gossip algorithm for information dissemination, we define a quantity known as the information spreading time. For any node $i \in V$, and any time t, let $M_i(t)$ be the set of messages that node i can decode using the information that it has at time t.

Definition 2.3. For an information dissemination algorithm $\mathcal{D}$, and any $\delta > 0$, the δ-information-spreading time of $\mathcal{D}$, denoted $T_{\mathcal{D}}^{\text{spr}}(\delta)$, is defined as

$$T_{\mathcal{D}}^{\text{spr}}(\delta) = \inf\left\{t \;\middle|\; \Pr\left(\bigcup_{i=1}^{n}\{|M_i(t)| < n\}\right) \leq \delta\right\}.$$

In words, if the information dissemination algorithm $\mathcal{D}$ runs for $T_{\mathcal{D}}^{\text{spr}}(\delta)$ time, then the probability that any node cannot decode all of the original n messages is at most δ.

2.1.1 Organization

Section 2.2 provides a description of the information dissemination algorithm based on network coding that was originally proposed by Deb and Médard. In Section 2.3, we analyze this algorithm in the synchronous and asynchronous models, and obtain an upper bound on the δ-information-spreading time of the algorithm.

2.2 Coding-Based Gossip Algorithm

The coding-based gossip algorithm for information dissemination consists of two components: the gossip *mechanism*, which determines how a node chooses a neighbor to contact when it initiates a communication; and the gossip *protocol*, which specifies the message transmitted by a node to its communication partner during a communication. Recall that

each node starts the communication protocol with its unique message, and the goal is to spread all of the messages to all of the nodes. We now describe the gossip mechanism and the random coding-based gossip protocol.

2.2.1 Gossip Mechanism

The gossip mechanism does not require the nodes to have knowledge of the complete graph topology; each node only must know about its neighbors. The analysis of Deb and Médard [11, 12] assumes that the communication graph G is complete, and that whenever a node contacts another node, it chooses the communication partner uniformly at random. In this chapter and in Chapter 3, we study a simple gossip mechanism that generalizes this setting. Recall that, in the asynchronous time model, a node becomes active when its clock ticks, while in the synchronous time model a node becomes active once every round or time step. An active node contacts a neighbor in the communication graph. To describe how the neighbor to contact is chosen, we define a matrix as follows.

Definition 2.4. For a communication graph $G = (V, E)$ with $V = \{1, \ldots, n\}$, an $n \times n$ matrix P is said to be a *communication matrix* for G if it satisfies the following conditions:

- $\forall i, j \in V$, $P_{ij} \geq 0$;
- $\forall i \in V$, $\sum_{j=1}^{n} P_{ij} = 1$;
- $\forall i, j \in V$, $i \neq j$, $(i, j) \notin E \Rightarrow P_{ij} = 0$.

Given a communication matrix P for the communication graph G, nodes choose communication partners according to the probability distributions defined by the matrix. When node i initiates a communication, it contacts node j with probability P_{ij}, independently of all other communication events. The constraint that $P_{ij} = 0$ if $(i, j) \notin E$ on a communication matrix prevents a node i from contacting a node j when there is no communication link between i and j. In this chapter, we will restrict our attention to symmetric matrices P; that is, $P_{ij} = P_{ji}$ for all nodes i and j.

We assume in this chapter that nodes transmit data according to the pull mechanism. That is, when node i contacts node j, it receives data from node j, but does not send data to node j. Another popular gossip mechanism is the push mechanism, in which node i sends data to node j when node i contacts node j. In Chapter 3, we study a gossip algorithm that uses both the push and pull mechanisms.

The data transmitted from one node to another during a communication are determined by the random linear coding protocol explained below. As the gossip algorithm executes, a node receives coded messages from its communication partners. Eventually, the node can decode the coded messages it has, as described below, to obtain all n original messages.

2.2.2 Random Linear Coding Gossip Protocol

We study here the same gossip protocol that was originally proposed by Deb and Médard [11, 12]. Each message is a vector over a finite field $\mathbf{F}_q$ of size $q \geq n$. Let each message be a vector of size $r \in \mathbf{Z}$. In particular, let the initial message at node i be $m_i \in \mathbf{F}_q^r$, for $i = 1, \ldots, n$, and let $M = \{m_1, \ldots, m_n\}$ denote the set of the n message vectors. During the execution of the gossip algorithm, each node collects linear combinations of message vectors in M. When each node has n linearly independent code vectors, it can recover all the messages in M successfully.

Now, consider a certain instant t, during the execution of the gossip algorithm, when node i becomes active and contacts node j. Let $C_i(t) = \{f_1, \ldots, f_{|C_i(t)|}\}$ and $C_j(t) = \{g_1, \ldots, g_{|C_j(t)|}\}$ be the sets of all the coded messages at nodes i and j, respectively, at time t. By definition, for $g_\ell \in C_j(t)$, $\ell = 1, \ldots, |C_j(t)|$, $g_\ell \in \mathbf{F}_q^r$ and

$$g_\ell = \sum_{u=1}^{n} \alpha_{\ell_u} m_u,$$

where $\alpha_{\ell_u} \in \mathbf{F}_q$ for each $u = 1, \ldots, n$. The protocol ensures that node j knows the coefficients α_{ℓ_u} (see [12] for details). An analogous situation holds for $C_i(t)$.

When a node i contacts node j, it receives a message from node j. This message is a random coded message with payload $e_{ji} \in \mathbf{F}_q^r$, where

$$e_{ji} = \sum_{g_\ell \in C_j(t)} \beta_\ell g_\ell.$$

The coefficients $\beta_\ell \in \mathbf{F}_q$ for all ℓ are chosen randomly according to the uniform probability distribution

$$\Pr(\beta_\ell = \beta) = \frac{1}{q}$$

for all $\beta \in \mathbf{F}_q$. The message e_{ji} can be re-written as

$$\begin{aligned} e_{ji} &= \sum_{g_\ell \in C_j(t)} \beta_\ell g_\ell \\ &= \sum_{g_\ell \in C_j(t)} \beta_\ell \sum_{u=1}^{n} \alpha_{\ell_u} m_u \\ &= \sum_{u=1}^{n} \left(\sum_{\ell=1}^{|C_j(t)|} \beta_\ell \alpha_{\ell_u} \right) m_u \\ &= \sum_{u=1}^{n} \theta_u m_u, \end{aligned} \tag{2.2}$$

where

$$\begin{aligned} \theta_u &= \sum_{\ell=1}^{|C_j(t)|} \beta_\ell a_{\ell_u} \\ &\in \mathbf{F}_q. \end{aligned}$$

For the purpose of decoding, along with the coded message e_{ji}, node j transmits the coefficients $(\theta_1, \ldots, \theta_n)$ to node i. This vector of coefficients is called a code vector. Once the subspace spanned by the code vectors at a node has dimension n, the node can recover the original messages $m_1, \ldots, m_n$ by solving a system of linear equations.

We now recall the following key result.

Lemma 2.3 (Lemma 2.1, [12]). *Let $U_i(t)^-$ and $U_j(t)^-$ denote the subspaces spanned by the code vectors at nodes i and j, respectively. Let $U_i(t)^+$ be the subspace spanned by the code vectors at i after j sends a coded message and a code vector to i at time t. For a subspace U, let $\dim(U)$ denote the dimension of U. Then,*

$$\Pr\left(\dim\left(U_i(t)^+\right) > \dim\left(U_i(t)^-\right) \mid U_j(t)^- \not\subseteq U_i(t)^-\right) \geq 1 - \frac{1}{q}.$$

When node i receives a message from node j, the dimension of the subspace spanned by the code vectors at i can increase only if the subspace spanned by the code vectors at j is not a subset of the subspace spanned by the code vectors at i. Lemma 2.3 provides a lower bound on the probability that the dimension of the subspace of node i will increase

whenever this is the case.

2.3 Analysis of Gossip Algorithm

We bound from above the running time of the coding-based information dissemination algorithm in an arbitrary connected graph in terms of properties of cuts in the graph. Given the communication graph $G = (V, E)$ with n nodes, let P be a communication matrix for G. We define the s-conductance of P as follows.

Definition 2.5. The *s-conductance* of a communication matrix P for G, denoted Φ_P^s, is defined as

$$\Phi_P^s = \min_{S \subset V,\, 0 < |S| \leq s} \frac{\sum_{i \in S, j \notin S} P_{ij}}{|S|}.$$

Note that the s-conductance of P is monotonically non-increasing in s. We assume that $\Phi_P^{n-1} > 0$, which implies that $\Phi_P^s > 0$ as well for all $s = 1, \ldots, n-2$. This assumption is satisfied, for example, whenever the communication graph G is connected and every node i contacts each of its neighbors j with positive probability P_{ij}. Without this assumption, it is possible for the information dissemination algorithm to reach a state in which the nodes in some set S cannot obtain all of the messages because they do not contact nodes outside of S.

When $P_{ij} = \Omega(1/n)$ for each edge $(i, j) \in E$ and the graph G is connected, $\Phi_P^s = \Omega(1/(ns))$ for any $s = 1, \ldots, n-1$. If every node contacts each of its neighbors uniformly at random and the graph has constant degree, then $\Phi_P^s = \Omega(1)$ when the number of edges crossing each cut (S, S^c) in the graph such that $|S| \leq s$ is at least a constant fraction of $|S|$. For example, a constant-degree expander graph would satisfy this property when s and $n - s$ are approximately balanced.

Each communication matrix for G, when combined with the random linear coding gossip protocol, gives rise to a coding-based information dissemination algorithm. We denote the δ-information-spreading time of the algorithm for a particular communication matrix P by $T_P^{\text{spr}}(\delta)$. We prove the following upper bound on the running time of the algorithm.

Theorem 2.4. *Consider the gossip algorithm based on random linear coding (over the finite field $\mathbf{F}_q$, $q \geq n$), using a communication matrix P for G that is symmetric. Suppose*

$\delta > 0$ is given and n is large enough. Let

$$\tau = \sum_{s=1}^{n-1} \frac{s}{\Phi_P^s}.$$

Then, in the asynchronous time model,

$$T_P^{\text{spr}}(\delta) = O\left(\frac{\tau}{n}\left(1 + \frac{\log \delta^{-1}}{n}\right)\right),$$

while in the synchronous time model,

$$T_P^{\text{spr}}(\delta) = O\left(\frac{\tau}{n} \log \delta^{-1}\right).$$

Remark. For a graph G, suppose that the communication matrix P is chosen to be $P_{ij} = 1/\Delta$ for each edge $(i, j) \in E$, where Δ is the maximum degree of the graph, and $P_{ii} = 1 - d_i/\Delta$, where d_i is the degree of node i. Then Theorem 2.4 implies that the δ-information-spreading time when $\delta = 1/n$ for complete graphs, constant-degree expanders, and ring graphs scales as $O(n \log n)$, $O(n \log n)$, and $O(n^2)$, respectively, in the asynchronous time model. Our bounds for the synchronous time model have an additional $\log n$ factor. Although the bound for the complete graph is weaker than the $O(n)$ bound of [12], this is a general bound that applies to graphs other than complete graphs.

To prove Theorem 2.4, we first prove the upper bound for the asynchronous time model, and then the upper bound for the synchronous time model. We first present some definitions and notation that are common to both time models. To this end, let t denote a certain instant of time when some nodes are communicating ($t \in \mathbf{R}_+$ for the asynchronous model and $t \in \mathbf{Z}_+$ for the synchronous model).

The subspaces spanned by the code vectors at node i before and after the communication at time t, respectively, are denoted by $U_i(t)^-$ and $U_i(t)^+$. We refer to the dimension of the subspace $U_i(t)^-$, denoted by $\dim(U_i(t)^-)$, as the dimension of the node i. In the synchronous model, $U_i(t)^+ = U_i(t+1)^-$.

Definition 2.6 (Type). Two nodes i and j are said to be of the same type at time t if $U_i(t)^- = U_j(t)^-$. Under this definition of type, all of the nodes are partitioned into different equivalence classes, which we refer to as type classes.

For example, if both nodes have enough code vectors to decode all n original messages, then the subspaces spanned by the code vectors at both of them will be the same, so they are of the same type. The subspaces spanned by the code vectors at all the nodes in a type class are the same, and the subspace corresponding to a type class is different than the subspaces corresponding to all the other type classes. At time t, let $L(t)$ be the size of the largest type class (the type class containing the most nodes).

When a node j transmits a random linear code to a node i such that $U_j(t)^- \not\subseteq U_i(t)^-$, from Lemma 2.3, $\dim(U_i(t)^+) \geq \dim(U_i(t)^-) + 1$ with probability at least $1 - 1/q$. Now, suppose that, at time t, two nodes i and j are not of the same type. Then it must be that either $U_i(t)^- \not\subseteq U_j(t)^-$, or $U_j(t)^- \not\subseteq U_i(t)^-$. Thus, if the nodes i and j are of different types, then the dimension of at least one of the two nodes will increase with probability at least $1 - 1/q$ when it pulls a coded message from the other node.

Since a node can decode all of the messages when the dimension of its subspace is n, the information will be disseminated to all of the nodes at the time

$$\inf\{t \mid \dim(U_i(t)^-) = n,\ \forall i \in V\}.$$

Initially, at $t = 0$ we have $\dim(U_i(0)^-) = 1$ for all i. Thus, the information spreads to all the nodes when the overall dimension increase among all the nodes is $n(n-1)$. Let

$$D(t) = \sum_{i=1}^{n} \dim(U_i(t)^-) - n$$

be the total dimension increase at time t. By definition, $D(0) = 0$ and the information has spread to all of the nodes when $D(t) = n(n-1)$. Now, for $s = 1, \ldots, n$, define

$$t_s = \inf\{t \mid L(t) \geq s\}$$

and $Y_s = D(t_s)$. In words, t_s is the first time when any type class has at least s nodes, and Y_s is the total dimension increase at time t_s. By definition, $t_1 = Y_1 = 0$. The following lemma provides a lower bound on Y_s.

Lemma 2.5. *For any $s = 1, \ldots, n$, $Y_s \geq s(s-1)$.*

Proof. Consider the time t_s, which is the first time any type class contains s nodes. At this time, there are nodes $i_1, \ldots, i_s$ in the same type class. Since these nodes are of the

same type, we have $U_{i_1}(t_s)^- = \cdots = U_{i_s}(t_s)^-$.

For a node $i \in V$, let $v_i \in \mathbf{F}_q^n$ be the code vector with a one in component i, and a zero in each component $u \neq i$. We note that the linear combination in (2.2) is equal to the original message m_i at node i when the code vector $(\theta_1, \ldots, \theta_n)$ is v_i. As a result, $v_{i\ell} \in U_{i\ell}(t)^-$ for all $\ell = 1, \ldots, s$ and $t \geq 0$. Hence, for all $\ell = 1, \ldots, s$,

$$\text{span}(v_{i_1}, \ldots, v_{i_s}) \subseteq U_{i\ell}(t_s)^-,$$

where $\text{span}(v_{i_1}, \ldots, v_{i_s})$ denotes the subspace spanned by the vectors $v_{i_1}, \ldots, v_{i_s}$. Since the vectors $v_{i_1}, \ldots, v_{i_s}$ are linearly independent, $\dim(U_{i\ell}(t_s)^-) \geq s$ for $\ell = 1, \ldots, s$, and so the total dimension increase is at least $s(s-1)$ by time t_s. That is, $Y_s = D(t_s) \geq s(s-1)$. □

We note that $Y_n = D(t_n) = n(n-1)$, and t_n is the first time when all nodes have received enough code vectors to decode the original messages.

2.3.1 Asynchronous Model

We now analyze the coding-based gossip algorithm under the asynchronous time model. Consider a sequence of independent geometric random variables $W_1, \ldots, W_k$ for any $k \geq 1$. For $i = 1, \ldots, k$, let $p_i \in (0, 1)$ be the parameter of the variable W_i, so that $\Pr(W_i > x) = (1 - p_i)^{\lfloor x \rfloor}$ for all $x \geq 0$. Now consider independent exponential random variables $\hat{W}_1, \ldots, \hat{W}_k$, where $\hat{W}_i$ is of rate $\lambda_i = \ln\left[(1 - p_i)^{-1}\right]$. For any $i = 1, \ldots, k$, by the definition of W_i and $\hat{W}_i$, W_i stochastically dominates $\hat{W}_i$ in the sense that $\Pr(\hat{W}_i > x) \leq \Pr(W_i > x)$ for all $x \geq 0$. Similarly, $\hat{W}_i + 1$ stochastically dominates W_i.

Define

$$B_k = \sum_{i=1}^{k} W_i$$

and

$$\hat{B}_k = k + \sum_{i=1}^{k} \hat{W}_i.$$

Since $\hat{W}_i + 1$ stochastically dominates W_i for each i, $\hat{B}_k$ stochastically dominates B_k. Thus, to obtain an upper bound on $\Pr(B_k > x)$, it suffices to obtain an upper bound on $\Pr(\hat{B}_k > x)$. To this end, in the following lemma we obtain an upper bound on the probability that $\hat{B}_k$ takes on a value larger than its mean by a multiplicative factor. Let

$\lambda^* = \min_{i=1}^k \lambda_i$ and $\hat{\mu}_k = E[\hat{B}_k]$; note that

$$\hat{\mu}_k = k + \sum_{i=1}^{k} \lambda_i^{-1}.$$

Lemma 2.6. *For any $\epsilon > 0$,*

$$\Pr(\hat{B}_k > (2+\epsilon)\hat{\mu}_k) \leq \exp\left(-\frac{\epsilon\lambda^*\hat{\mu}_k}{2}\right).$$

Proof. Let $z = \lambda^*/2$ and $y \geq \hat{\mu}_k$. Then, $0 < z < \lambda^*$, and

$$\begin{aligned}
\Pr(\hat{B}_k > y) &= \Pr(\exp(z\hat{B}_k) > \exp(zy)) \\
&\leq \exp(-zy)E[\exp(z\hat{B}_k)] \\
&= \exp(-zy)E\left[\exp\left(z\left(k + \sum_{i=1}^{k}\hat{W}_i\right)\right)\right] \\
&= \exp(zk - zy)\prod_{i=1}^{k} E[\exp(z\hat{W}_i)] \\
&= \exp(zk - zy)\prod_{i=1}^{k}\left(1 - \frac{z}{\lambda_i}\right)^{-1}, \qquad (2.3)
\end{aligned}$$

where we have used Markov's inequality, and the fact that for an exponential random variable $\hat{W}$ of rate λ,

$$E[\exp(z\hat{W})] = \left(1 - \frac{z}{\lambda}\right)^{-1}$$

for $z < \lambda$. Note that $z < \lambda_i$ because $z < \lambda^*$. Furthermore, $0 < z/\lambda_i \leq 1/2$ because $z = \lambda^*/2$. The Taylor series expansion of $\exp(2x)$ implies that $(1-x)^{-1} \leq \exp(2x)$ for $0 < x \leq 1/2$, and thus

$$\begin{aligned}
\prod_{i=1}^{k}\left(1 - \frac{z}{\lambda_i}\right)^{-1} &\leq \prod_{i=1}^{k}\exp\left(\frac{2z}{\lambda_i}\right) \\
&= \exp\left(2z\sum_{i=1}^{k}\lambda_i^{-1}\right). \qquad (2.4)
\end{aligned}$$

From (2.3) and (2.4), we obtain

$$\Pr(\hat{B}_k > y) \leq \exp\left(zk + 2z\sum_{i=1}^{k}\lambda_i^{-1} - zy\right).$$

Hence, for $y = (2+\epsilon)\hat{\mu}_k$,

$$\begin{aligned}
\Pr(\hat{B}_k > (2+\epsilon)\hat{\mu}_k) &\leq \exp\left(zk + 2z\sum_{i=1}^{k}\lambda_i^{-1} - z(2+\epsilon)\hat{\mu}_k\right) \\
&= \exp\left(zk + 2z\sum_{i=1}^{k}\lambda_i^{-1} - z(2+\epsilon)\left(k + \sum_{i=1}^{k}\lambda_i^{-1}\right)\right) \\
&= \exp\left(zk + 2z\sum_{i=1}^{k}\lambda_i^{-1} - 2zk - \epsilon zk - 2z\sum_{i=1}^{k}\lambda_i^{-1} - \epsilon z\sum_{i=1}^{k}\lambda_i^{-1}\right) \\
&\leq \exp\left(-\epsilon z\left(k + \sum_{i=1}^{k}\lambda_i^{-1}\right)\right).
\end{aligned}$$

Substituting into this expression using the definitions of $\hat{\mu}_k$ and z yields

$$\begin{aligned}
\Pr(\hat{B}_k > (2+\epsilon)\hat{\mu}_k) &\leq \exp(-\epsilon z\hat{\mu}_k) \\
&= \exp\left(-\frac{\epsilon\lambda^*\hat{\mu}_k}{2}\right),
\end{aligned}$$

which is the claimed inequality. □

The following corollary allows us to apply this upper bound to the random variable B_k. Let $\mu_k = E[B_k]$.

Corollary 2.7. *For any $\epsilon > 0$,*

$$\Pr(B_k > (2+\epsilon)(\mu_k + k)) \leq \exp\left(-\frac{\epsilon\lambda^*\mu_k}{2}\right).$$

Proof. By linearity of expectation,

$$\mu_k = \sum_{i=1}^{k} E[W_i].$$

Due to the stochastic domination relationships among the W_i and $\hat{W}_i$ variables, we have

$$\begin{aligned}\mu_k &\geq \sum_{i=1}^{k} E[\hat{W}_i] \\ &= \hat{\mu}_k - k.\end{aligned}$$

Using this inequality, the fact that $\hat{B}_k$ stochastically dominates B_k, and Lemma 2.6, we conclude that

$$\begin{aligned}\Pr(B_k > (2+\epsilon)(\mu_k + k)) &\leq \Pr(B_k > (2+\epsilon)\hat{\mu}_k) \\ &\leq \Pr(\hat{B}_k > (2+\epsilon)\hat{\mu}_k) \\ &\leq \exp\left(-\frac{\epsilon\lambda^*\hat{\mu}_k}{2}\right) \\ &\leq \exp\left(-\frac{\epsilon\lambda^*\mu_k}{2}\right).\end{aligned}$$

□

Consider now a time t when the global clock ticks (according to a Poisson process of rate n). At this instant, only one node receives a coded message from another node, so the total dimension increase is at most 1. We want to obtain a lower bound on the probability of dimension increase. To this end, suppose there are $c \leq n$ type classes. For each type class $k = 1, \ldots, c$, let S_k be the set of nodes in class k. Let S^i denote the set of nodes in the type class of a node i.

For a pair of nodes i, j, let X_{ij} be an indicator random variable that is 1 if node i contacts node j at time t and the dimension of i increases as a result of the communication, and is 0 otherwise. The node i becomes active with probability $1/n$ and contacts j with probability P_{ij}. Similarly, j contacts i with net probability $P_{ji}/n = P_{ij}/n$. If $S^i = S^j$, then there will be no increase in total dimension if i and j communicate. As noted before, however, if i and j belong to different type classes, then the dimension of at least one of the two nodes will increase with probability at least $1 - 1/q$ if it contacts the other node. This implies that whenever $S^i \neq S^j$,

$$E[X_{ij}] + E[X_{ji}] \geq \left(1 - \frac{1}{q}\right)\frac{P_{ij}}{n}.$$

This inequality leads to a lower bound on the probability of dimension increase. Let p

denote the probability that the dimension of the node activated at time t increases. We have

$$\begin{aligned} p &= \sum_{i\in V}\sum_{j\neq i} E[X_{ij}] \\ &= \sum_{i\in V}\sum_{j\notin S^i, j>i} (E[X_{ij}] + E[X_{ji}]) \\ &\geq \sum_{i\in V}\sum_{j\notin S^i, j>i} \left(1-\frac{1}{q}\right)\frac{P_{ij}}{n} \\ &= \frac{1}{2n}\left(1-\frac{1}{q}\right)\sum_{i\in V}\sum_{j\notin S^i} P_{ij}, \end{aligned} \tag{2.5}$$

where we have used the fact that P is symmetric. Now, we rewrite the sum in (2.5) in terms of the type classes as

$$\begin{aligned} p &\geq \frac{1}{2n}\left(1-\frac{1}{q}\right)\sum_{k=1}^{c}\sum_{i\in S_k, j\notin S_k} P_{ij} \\ &= \frac{1}{2n}\left(1-\frac{1}{q}\right)\sum_{k=1}^{c}|S_k|\frac{\sum_{i\in S_k, j\notin S_k} P_{ij}}{|S_k|}. \end{aligned} \tag{2.6}$$

Suppose that $t \in [t_s, t_{s+1})$ for $s = 1, \ldots, n-1$. Then, by definition, $|S_k| \leq s$ for all $k = 1, \ldots, c$. Using the definition of the s-conductance Φ_P^s and (2.6), we obtain

$$\begin{aligned} p &\geq \frac{1}{2n}\left(1-\frac{1}{q}\right)\sum_{k=1}^{c}|S_k|\Phi_P^s \\ &= \frac{\Phi_P^s}{2}\left(1-\frac{1}{q}\right). \end{aligned}$$

Thus, in the time interval $[t_s, t_{s+1})$, the number of clock ticks required for a unit dimension increase can be stochastically bounded from above by a geometric random variable with parameter

$$p_s \stackrel{\triangle}{=} \left(1-\frac{1}{q}\right)\frac{\Phi_P^s}{2}. \tag{2.7}$$

When the total dimension increase is $n(n-1)$, each node has received enough code vectors to obtain the original messages. As such, the number of global clock ticks K required for all nodes to decode all the original messages can be stochastically bounded

from above as

$$K \leq \sum_{d=1}^{n(n-1)} W_d,$$

where the W_d are independent geometric random variables with parameter p_s when $t \in [t_s, t_{s+1})$. By the definition of s-conductance, p_s is monotonically non-increasing in s. Hence, the smaller the t_s values are, the worse this stochastic upper bound on K is. From Lemma 2.5, the worst stochastic upper bound on K arises when $t_s = s(s-1)$, in which case $t_{s+1} - t_s = 2s$ and W stochastically dominates K, where

$$W = \sum_{s=1}^{n-1} \sum_{\ell=1}^{2s} W_\ell^s \tag{2.8}$$

and the W_ℓ^s are independent geometric random variables with parameter p_s. From the fact that W stochastically dominates K and (2.8), it follows that for $q \geq n \geq 2$,

$$\begin{aligned} E[K] &\leq E[W] \\ &= \sum_{s=1}^{n-1} \sum_{\ell=1}^{2s} E[W_\ell^s] \\ &= \sum_{s=1}^{n-1} \sum_{\ell=1}^{2s} p_s^{-1} \\ &= \frac{2}{1-\frac{1}{q}} \sum_{s=1}^{n-1} \sum_{\ell=1}^{2s} \frac{1}{\Phi_P^s} \\ &= \frac{4}{1-\frac{1}{q}} \sum_{s=1}^{n-1} \frac{s}{\Phi_P^s} \\ &= \Theta(\tau). \end{aligned} \tag{2.9}$$

To obtain the bound with probability $1 - \delta/2$, we use Corollary 2.7. Let

$$\lambda^* = \min_{s=1,\ldots,n-1} \ln\left[(1-p_s)^{-1}\right].$$

The Taylor series expansion of $\ln(1-x)$ implies that $\ln\left[(1-x)^{-1}\right] \geq x$ for $0 < x < 1$.

Using this inequality and the fact that Φ_P^s is monotonically non-increasing in s, we obtain

$$\begin{aligned}\lambda^* &\geq \min_{s=1,\ldots,n-1} p_s \\ &= \min_{s=1,\ldots,n-1} \left(1 - \frac{1}{q}\right) \frac{\Phi_P^s}{2} \\ &= \left(1 - \frac{1}{q}\right) \frac{\Phi_P^{n-1}}{2}.\end{aligned}$$

The definition of τ implies that

$$\tau \geq \frac{n-1}{\Phi_P^{n-1}},$$

and so

$$\lambda^* = \Omega\left(\frac{n}{\tau}\right). \tag{2.10}$$

Now, from (2.8) and Corollary 2.7, for $\epsilon > 0$,

$$\Pr\left(W > (2+\epsilon)(E[W] + n(n-1))\right) \leq \exp\left(-\frac{\epsilon\lambda^* E[W]}{2}\right). \tag{2.11}$$

For

$$\epsilon = \frac{2\ln(2\delta^{-1})}{\lambda^* E[W]},$$

let $\kappa = (2+\epsilon)(E[W] + n(n-1))$. Since $E[W] = \Omega(n^2)$, we have

$$\begin{aligned}\kappa &= O\left((1+\epsilon)E[W]\right) \\ &= O\left(\tau\left(1 + \frac{\log\delta^{-1}}{n}\right)\right).\end{aligned}$$

Substituting for ϵ in the inequality in (2.11), we obtain $\Pr(W > \kappa) \leq \delta/2$. This provides an upper bound on the number of clock ticks required for every node to obtain every message.

To extend the bound to absolute time, we apply Corollary 2.2, which implies that the probability that $\kappa = \Omega(\log\delta^{-1})$ clock ticks do not occur by absolute time $O(\kappa/n)$ is at most $\delta/2$. We conclude from the union bound, (2.9), and (2.10) that

$$T_P^{\text{spr}}(\delta) = O\left(\frac{\tau}{n}\left(1 + \frac{\log\delta^{-1}}{n}\right)\right).$$

This completes the proof of the upper bound in Theorem 2.4 for the asynchronous time model.

2.3.2 Synchronous Model

For the synchronous time model, we begin as in the analysis of the asynchronous model. Suppose that at time $t \in [t_s, t_{s+1})$ for $s = 1, \ldots, n-1$, there are c type classes, and the sets of nodes in the type classes are $S_1, \ldots, S_c$. As before, X_{ij} is an indicator random variable specifying whether node i contacts node j and the dimension of i increases in round t. Let $\Delta(t) = D(t+1) - D(t)$ denote the total dimension increase of all nodes in this round, so that

$$\Delta(t) = \sum_{i \in V} \sum_{j \in V} X_{ij}. \tag{2.12}$$

Repeating the argument for the asynchronous model, we consider two nodes i and j of different classes $S^i \neq S^j$. In the synchronous model, we have

$$E[X_{ij}] + E[X_{ji}] \geq \left(1 - \frac{1}{q}\right) P_{ij};$$

the factor of $1/n$ in the asynchronous case is not present because all nodes are simultaneously active in the synchronous model. We use (2.12) and this lower bound to obtain the lower bound

$$\begin{aligned}
E[\Delta(t)] &= \sum_{i \in V} \sum_{j \notin S^i, j > i} (E[X_{ij}] + E[X_{ji}]) \\
&\geq \sum_{i \in V} \sum_{j \notin S^i, j > i} \left(1 - \frac{1}{q}\right) P_{ij} \\
&= \frac{1}{2} \left(1 - \frac{1}{q}\right) \sum_{i \in V} \sum_{j \notin S^i} P_{ij} \\
&= \frac{1}{2} \left(1 - \frac{1}{q}\right) \sum_{k=1}^{c} |S_k| \frac{\sum_{i \in S_k, j \notin S_k} P_{ij}}{|S_k|} \\
&\geq \frac{n \Phi_P^s}{2} \left(1 - \frac{1}{q}\right).
\end{aligned}$$

Using the same definition of p_s as in (2.7) for the asynchronous model, we have $E[\Delta(t)] \geq np_s$. This provides a lower bound on the expected total dimension increase during any

round in the period $[t_s, t_{s+1})$. Note that this lower bound holds for every $t \in [t_s, t_{s+1})$ uniformly. Define

$$Z^s(t) = \sum_{u=t_s}^{t-1} (\Delta(u) - np_s) 1_{\{u<t_{s+1}\}},$$

where $1_{\{u<t_{s+1}\}}$ is an indicator function that takes the value one when $u < t_{s+1}$, and the value zero otherwise, and $Z^s(t_s) = 0$. For $t \geq t_s$, $Z^s(t)$ is a submartingale, as

$$E\left[Z^s(t+1) \mid Z^s(t)\right] \geq Z^s(t).$$

The quantity t_{s+1} is a stopping time with respect to the history of the algorithm. Since $\Phi_P^s > 0$, there is a positive probability that the dimension of at least one node increases in each round. This fact and the upper bound of $n(n-1)$ on the total dimension increase that occurs throughout the execution of the algorithm imply that t_{s+1} is stochastically dominated by a sum of a finite number of geometric random variables with positive parameters, from which we conclude that $E[t_{s+1}] < \infty$. Moreover, the submartingale $Z^s(t)$ has bounded increments. These properties imply that $Z^s(\min(t_{s+1}, t))$ is a uniformly integrable submartingale, and hence

$$E[Z^s(t_{s+1})] \geq E[Z^s(t_s)] = 0. \tag{2.13}$$

Now, from the definitions of $t_s, t_{s+1}, Y_s, Y_{s+1}$, and (2.13), we obtain

$$E[Y_{s+1} - Y_s] \geq np_s E[t_{s+1} - t_s]. \tag{2.14}$$

Recall that t_n is the time when all nodes can decode all the messages. Summing the inequality in (2.14) for all $s = 1, \ldots, n-1$ yields

$$E[t_n] \leq \sum_{s=1}^{n-1} \frac{E[Y_{s+1} - Y_s]}{np_s}. \tag{2.15}$$

From Lemma 2.5 and the fact that p_s is monotonically non-increasing in s, the quantity

in the right-hand side of the inequality in (2.15) is maximized when $Y_s = s(s-1)$. Hence,

$$\begin{aligned} E[t_n] &\leq \sum_{s=1}^{n-1} \frac{2s}{np_s} \\ &= \frac{2\tau}{n}. \end{aligned} \tag{2.16}$$

By Markov's inequality, the inequality in (2.16) implies that

$$\Pr\left(t_n > \frac{4\tau}{n}\right) < \frac{1}{2}.$$

Now, for the purpose of analysis, consider dividing time into epochs of length $4\tau/n$, and executing the information dissemination algorithm from the initial state in each epoch, independently of the other epochs. The probability that, after $\log \delta^{-1}$ epochs, some execution of the algorithm has run to completion in its epoch is greater than $1 - \delta$. Using the running time of this virtual process as a stochastic upper bound on the running time of the actual algorithm, we can conclude that

$$T_P^{\mathrm{spr}}(\delta) = O\left(\frac{\tau}{n} \log \delta^{-1}\right).$$

This completes the proof of Theorem 2.4.

Chapter 3

Computing Separable Functions via Gossip

This chapter considers the problem of computing separable functions in a distributed fashion. We develop a gossip algorithm in which each node has its own estimate of the value of the function, which evolves as the algorithm executes. Our goal is to minimize the amount of time required for all of these estimates to be close to the actual function value.

When each node has only a local view of the network, it is difficult, without global coordination, to simply transmit every node's value throughout the network so that each node can identify the values at all the nodes. As such, we develop an algorithm for computing separable functions that relies on an *order- and duplicate-insensitive* statistic [37] of a set of numbers, the minimum. The algorithm is based on properties of exponential random variables, and reduces the problem of computing the value of a separable function to the problem of determining the minimum of a collection of numbers, one for each node.

Because the minimum of a collection of numbers is not affected by the order in which the numbers appear, nor by the presence of duplicates of an individual number, the minimum computation required by our algorithm for computing separable functions can be performed by an information dissemination algorithm. The minimum computation can be implemented efficiently if, in the information dissemination algorithm, the messages transmitted between nodes are only the original messages that start at the nodes. Our analysis of the algorithm for computing separable functions establishes an upper bound on

its running time in terms of the running time of the information dissemination algorithm it uses as a subroutine.

Recall that the information dissemination algorithm based on network coding from Chapter 2 uses messages that are different than the original messages. As such, to obtain a complete algorithm for computing separable functions, we employ a gossip algorithm for information dissemination that only transmits the original messages. As in Chapter 2, the choices of communication partners by the nodes in this algorithm are governed by a communication matrix for the communication graph. We provide an upper bound on the running time of the algorithm in terms of the conductance of the communication matrix. By using the gossip algorithm to compute minima in the algorithm for computing separable functions, we obtain an algorithm for computing separable functions whose performance on certain graphs compares favorably with that of known iterative distributed algorithms [7] for computing averages in a network.

3.1 Model and Overview

The network model that we study in this chapter is the same as the one in Chapter 2. We consider an arbitrary connected network, represented by an undirected communication graph $G = (V, E)$, with $V = \{1, \ldots, n\}$. The communication between nodes is subject to the transmitter gossip constraint, so that each node is allowed to contact at most one other node at a given time.

Let 2^V denote the power set of the vertex set V (the set of all subsets of V). For an n-dimensional vector $x \in \mathbf{R}^n$, let $x_1, \ldots, x_n$ be the components of x.

Definition 3.1. A function $f : \mathbf{R}^n \times 2^V \to \mathbf{R}$ is *separable* if there exist functions $f_1, \ldots, f_n$ such that, for all $x \in \mathbf{R}^n$ and $S \subseteq V$,

$$f(x, S) = \sum_{i \in S} f_i(x_i). \tag{3.1}$$

Let $\mathcal{F}$ be the class of separable functions f for which $f_i(x_i) \geq 1$ for all $x_i \in \mathbf{R}$ and $i = 1, \ldots, n$. Given a function $f \in \mathcal{F}$, and a vector $x \in \mathbf{R}^n$ containing initial values x_i for all the nodes, we seek a gossip algorithm that the nodes in the network can use to compute the value $f(x, V)$.

Remark. Consider a function g for which there exist functions $g_1, \ldots, g_n$ satisfying, for all $S \subseteq V$, the condition $g(x, S) = \prod_{i \in S} g_i(x_i)$ in lieu of (3.1). Then, g is *logarithmic separable* in the sense that $f = \log_b g$ is separable. Our algorithm for computing separable functions can be used to compute the function $f = \log_b g$. The condition $f_i(x_i) \geq 1$ corresponds to $g_i(x_i) \geq b$ in this case. This lower bound of 1 on $f_i(x_i)$ is arbitrary, although our algorithm does require the terms $f_i(x_i)$ in the sum to be positive.

Before proceeding further, we list some practical situations where the distributed computation of separable functions arises naturally.

Example 3.1 (Summation). By definition, the sum of a set of numbers is a separable function. Suppose that the value at each node is $x_i = 1$, and the function f_i is the identity function. Then, the sum of the values is the number of nodes in the network.

Example 3.2 (Averaging). According to Definition 3.1, the average of a set of numbers is not a separable function. However, the nodes can estimate the separable function $\sum_{i=1}^{n} x_i$ and n separately, and use the ratio between these two estimates as an estimate of the mean of the numbers.

Suppose the values at the nodes are measurements of a quantity of interest. Then, the average provides an unbiased maximum likelihood estimate of the measured quantity. For example, if the nodes are temperature sensors, then the average of the sensed values at the nodes gives a good estimate of the ambient temperature.

For more sophisticated applications of a distributed averaging algorithm, we refer the reader to [26] and [30]. Averaging is used for the distributed computation of the top k eigenvectors of a graph in [26], while in [30] averaging is used in a throughput-optimal distributed scheduling algorithm in a wireless network.

In this chapter, we use the same two time models as in Chapter 2, the synchronous and asynchronous time models, which are defined in Definitions 2.1 and 2.2. We continue to measure the running times of algorithms in absolute time, which is the number of time slots in the synchronous model, and is (on average) the number of clock ticks divided by n in the asynchronous model.

Our algorithm for computing separable functions is randomized, and is not guaranteed to compute the exact quantity $f(x, V) = \sum_{i=1}^{n} f_i(x_i)$ at each node in the network. To study the accuracy of the algorithm's estimates, we analyze the probability that the

estimate of $f(x, V)$ at every node is within a $(1 \pm \epsilon)$ multiplicative factor of the true value $f(x, V)$ after the algorithm has run for some period of time. In this sense, the error in the estimates of the algorithm is relative to the magnitude of $f(x, V)$.

To measure the amount of time required for an algorithm's estimates to achieve a specified accuracy with a specified probability, we define the following quantity. For an algorithm $\mathcal{C}$ that estimates $f(x, V)$, let $\hat{y}_i(t)$ be the estimate of $f(x, V)$ at node i at time t. Furthermore, for notational convenience, given $\epsilon > 0$, let $D_i^\epsilon(t)$ be the event

$$D_i^\epsilon(t) = \{\hat{y}_i(t) \notin [(1-\epsilon)f(x,V), (1+\epsilon)f(x,V)]\}.$$

Definition 3.2. For any $\epsilon > 0$ and $\delta \in (0, 1)$, the $(\epsilon,\ \delta)$-computing time of $\mathcal{C}$, denoted $T_{\mathcal{C}}^{\text{cmp}}(\epsilon, \delta)$, is

$$T_{\mathcal{C}}^{\text{cmp}}(\epsilon, \delta) = \sup_{f \in \mathcal{F}} \sup_{x \in \mathbf{R}^n} \inf \left\{ \tau \;\middle|\; \forall t \geq \tau, \Pr\left(\bigcup_{i=1}^{n} D_i^\epsilon(t) \right) \leq \delta \right\}.$$

In words, if $\mathcal{C}$ runs for an amount of time that is at least $T_{\mathcal{C}}^{\text{cmp}}(\epsilon, \delta)$, then the probability that the estimates of $f(x, V)$ at the nodes are all within a $(1 \pm \epsilon)$ factor of the actual value of the function is at least $1 - \delta$.

As noted before, our algorithm for computing separable functions is based on a reduction to the problem of information spreading. In our analysis of the gossip algorithm for information spreading, we assume that when two nodes communicate, each node can send all of its messages to the other in a single communication. This rather unrealistic assumption of infinite link capacity is merely for convenience, as it provides a simpler analytical characterization of $T_{\mathcal{C}}^{\text{cmp}}(\epsilon, \delta)$ in terms of the δ-information-spreading time $T_{\mathcal{D}}^{\text{spr}}(\delta)$ of the information dissemination algorithm $\mathcal{D}$. As discussed below in Section 3.2.1, our algorithm for computing separable functions requires only links of unit capacity.

3.1.1 Organization

In Section 3.2, we develop and analyze an algorithm for computing separable functions in a distributed manner. Section 3.3 contains an analysis of a simple randomized gossip algorithm for information spreading, which can be used as a subroutine in the algorithm for computing separable functions. In Section 3.4, we discuss applications of our results

to particular types of graphs, and compare our results to previous results for computing averages.

3.2 Function Computation

In this section, we describe and analyze our algorithm for computing the value of a separable function. Our algorithm is randomized, and in particular uses exponential random variables. This usage of exponential random variables is analogous to that in an algorithm by Cohen for estimating the sizes of sets in a graph [9]. The basis for our algorithm is the following property of the exponential distribution.

Property 3.1. *Let $W_1, \ldots, W_n$ be n independent random variables such that, for $i = 1, \ldots, n$, the distribution of W_i is exponential with rate λ_i. Let W^* be the minimum of $W_1, \ldots, W_n$. Then, W^* is distributed as an exponential random variable of rate $\lambda = \sum_{i=1}^{n} \lambda_i$.*

Proof. For an exponential random variable W with rate λ, for any $z \in \mathbf{R}_+$,

$$\Pr(W > z) = \exp(-\lambda z).$$

Using this fact and the independence of the random variables W_i, we compute $\Pr(W^* > z)$ for any $z \in \mathbf{R}_+$ as

$$\begin{aligned}
\Pr(W^* > z) &= \Pr\left(\bigcap_{i=1}^{n} \{W_i > z\}\right) \\
&= \prod_{i=1}^{n} \Pr(W_i > z) \\
&= \prod_{i=1}^{n} \exp(-\lambda_i z) \\
&= \exp\left(-z \sum_{i=1}^{n} \lambda_i\right).
\end{aligned}$$

This establishes the property stated above. □

Let $y = f(x, V) = \sum_{i=1}^{n} f_i(x_i)$ be the value of the separable function f, where $f_i(x_i) \geq 1$. For simplicity of notation, let $y_i = f_i(x_i)$. Given x_i, each node can compute y_i on its

own. Next, the nodes use the algorithm shown in Figure 3.1, which we refer to as COMP, to compute estimates $\hat{y}_i$ of $y = \sum_{i=1}^{n} y_i$. The quantity r is a parameter to be chosen later.

Algorithm COMP

0. Initially, for $i = 1, \ldots, n$, node i has the value $y_i \geq 1$.
1. Each node i generates r independent random numbers $W_1^i, \ldots, W_r^i$, where the distribution of each W_ℓ^i is exponential with rate y_i (i.e., with mean $1/y_i$).
2. Each node i computes, for $\ell = 1, \ldots, r$, an estimate $\hat{W}_\ell^i$ of the minimum $W_\ell^* = \min_{i=1}^{n} W_\ell^i$. This computation can be done using an information spreading algorithm as described below.
3. Each node i computes $\hat{y}_i = \frac{r}{\sum_{\ell=1}^{r} \hat{W}_\ell^i}$ as its estimate of $\sum_{i=1}^{n} y_i$.

Figure 3.1: An algorithm for computing separable functions.

We describe how the minimum is computed as required by step 2 of the algorithm in Section 3.2.1. The running time of the algorithm COMP depends on the running time of the algorithm used to compute the minimum.

Now, we show that COMP effectively estimates the function value y when the estimates $\hat{W}_\ell^i$ are all correct by providing a lower bound on the conditional probability that the estimates produced by COMP are all within a $(1 \pm \epsilon)$ factor of y.

Lemma 3.2. *Let $y_1, \ldots, y_n$ be real numbers (with $y_i \geq 1$ for $i = 1, \ldots, n$), $y = \sum_{i=1}^{n} y_i$, and $W^* = (W_1^*, \ldots, W_r^*)$, where the W_ℓ^* are as defined in the algorithm* COMP. *For any node i, let $\hat{W}^i = (\hat{W}_1^i, \ldots, \hat{W}_r^i)$, and let $\hat{y}_i$ be the estimate of y obtained by node i in* COMP. *For every $\epsilon \in (0, 1/2)$,*

$$\Pr\left(\bigcup_{i=1}^{n} \{|\hat{y}_i - y| > 2\epsilon y\} \;\middle|\; \forall i \in V,\, \hat{W}^i = W^*\right) \leq 2\exp\left(-\frac{\epsilon^2 r}{3}\right).$$

Proof. Observe that the estimate $\hat{y}_i$ of y at node i is a function of r and $\hat{W}^i$. Under the hypothesis that $\hat{W}^i = W^*$ for all nodes $i \in V$, all nodes produce the same estimate $\hat{y} = \hat{y}_i$ of y. This estimate is

$$\hat{y} = \frac{r}{\sum_{\ell=1}^{r} W_\ell^*},$$

and so

$$\hat{y}^{-1} = \frac{\sum_{\ell=1}^{r} W_\ell^*}{r}.$$

Property 3.1 implies that each of the r random variables $W_1^*, \ldots, W_r^*$ has an exponential distribution with rate y. From Lemma 2.1, it follows that for every $\epsilon \in (0, 1/2)$,

$$\Pr\left(\left|\frac{1}{\hat{y}} - \frac{1}{y}\right| > \frac{\epsilon}{y} \;\middle|\; \forall i \in V,\, \hat{W}^i = W^*\right) \leq 2\exp\left(-\frac{\epsilon^2 r}{3}\right). \tag{3.2}$$

This inequality bounds the conditional probability of the event $\{\hat{y}^{-1} \notin [(1-\epsilon)y^{-1}, (1+\epsilon)y^{-1}]\}$, which is equivalent to the event $\{\hat{y} \notin [(1+\epsilon)^{-1}y, (1-\epsilon)^{-1}y]\}$. Now, for $\epsilon \in (0, 1/2)$,

$$(1-\epsilon)^{-1} \in [1+\epsilon, 1+2\epsilon] \tag{3.3}$$

and

$$(1+\epsilon)^{-1} \in [1-\epsilon, 1-2\epsilon/3]\,. \tag{3.4}$$

Applying the inequalities in (3.2), (3.3), and (3.4), we conclude that for $\epsilon \in (0, 1/2)$,

$$\Pr\left(|\hat{y} - y| > 2\epsilon y \mid \forall i \in V,\, \hat{W}^i = W^*\right) \leq 2\exp\left(-\frac{\epsilon^2 r}{3}\right).$$

Noting that the event

$$\bigcup_{i=1}^{n} \{|\hat{y}_i - y| > 2\epsilon y\}$$

is equivalent to the event $\{|\hat{y} - y| > 2\epsilon y\}$ when $\hat{W}^i = W^*$ for all nodes i completes the proof of Lemma 3.2. □

3.2.1 Using Information Spreading to Compute Minima

We now elaborate on step 2 of the algorithm COMP. Each node i in the graph starts this step with a vector $W^i = (W_1^i, \ldots, W_r^i)$, and the nodes seek the vector $W^* = (W_1^*, \ldots, W_r^*)$, where $W_\ell^* = \min_{i=1}^n W_\ell^i$. In the information spreading problem, each node i has a message m_i, and the nodes are to transmit messages across the links until every node has every message.

If all link capacities are infinite (i.e., in one time unit, a node can send an arbitrary amount of information to another node), then an information spreading algorithm can

be used directly to compute the minimum vector W^*. To see this, let the message m_i at the node i be the vector W^i, and then apply the information spreading algorithm to disseminate the vectors. Once every node has every message (vector), each node can compute W^* as the component-wise minimum of all the vectors. This implies that the running time of the resulting algorithm for computing W^* is the same as that of the information spreading algorithm.

The assumption of infinite link capacities allows a node to transmit an arbitrary number of vectors W^i to a neighbor in one time unit. If, in the information spreading algorithm, the nodes only transmit the original messages to each other, then a simple modification to the information spreading algorithm yields an algorithm for computing the minimum vector W^* using links of capacity r. Note that the information dissemination algorithm in Chapter 2 based on network coding uses coded messages between nodes, and as such does not satisfy the property that the nodes only transmit the original messages.

In the modified information spreading algorithm, each node i maintains a single r-dimensional vector $w^i(t)$ that evolves in time, starting with $w^i(0) = W^i$. Suppose that, in the information spreading algorithm, node j transmits the messages (vectors) $W^{i_1}, \ldots, W^{i_c}$ to node i at time t. Then, in the minimum computation algorithm, j sends to i the r quantities $w_1, \ldots, w_r$, where $w_\ell = \min_{u=1}^{c} W_\ell^{i_u}$. The node i sets $w_\ell^i(t^+) = \min(w_\ell^i(t^-), w_\ell)$ for $\ell = 1, \ldots, r$, where t^- and t^+ denote the times immediately before and after, respectively, the communication. At any time t, we will have $w^i(t) = W^*$ for all nodes $i \in V$ if, in the information spreading algorithm, every node i has all the vectors $W^1, \ldots, W^n$ at the same time t. In this way, we obtain an algorithm for computing the minimum vector W^* that uses links of capacity r and runs in the same amount of time as the information spreading algorithm.

An alternative to using links of capacity r in the computation of W^* is to make the time slot r times larger, and impose a unit capacity on all the links. Now, a node transmits the numbers $w_1, \ldots, w_r$ to its communication partner over a period of r time slots, and as a result the running time of the algorithm for computing W^* becomes greater than the running time of the information spreading algorithm by a factor of r. The preceding discussion, combined with the fact that nodes only gain messages as an information spreading algorithm executes, leads to the following lemma.

Lemma 3.3. *Suppose that the* COMP *algorithm is implemented using an information*

spreading algorithm $\mathcal{D}$ as described above. Let $\hat{W}^i(t)$ denote the estimate of W^* at node i at time t. For any $\delta \in (0,1)$, let $t_m = rT_{\mathcal{D}}^{\mathrm{spr}}(\delta)$. Then, for any time $t \geq t_m$, with probability at least $1-\delta$, $\hat{W}^i(t) = W^*$ for all nodes $i \in V$.

3.2.2 Analysis of Running Time

As described in the previous section, the algorithm COMP uses an information spreading algorithm as a subroutine. The faster the information spreading algorithm is, the better the COMP algorithm performs. Specifically, the following theorem provides an upper bound on the (ϵ, δ)-computing time of the COMP algorithm in terms of the δ-information-spreading time of the information spreading algorithm.

Theorem 3.4. *Suppose that the* COMP *algorithm is implemented using an information spreading algorithm $\mathcal{D}$ with δ-information-spreading time $T_{\mathcal{D}}^{\mathrm{spr}}(\delta)$ for $\delta \in (0,1)$. For any $\epsilon \in (0,1)$ and $\delta \in (0,1)$, there is a parameter value $r = r(\epsilon, \delta)$ such that*

$$T_{\mathrm{COMP}}^{\mathrm{cmp}}(\epsilon, \delta) \leq 18\epsilon^{-2}\left(1 + \ln \delta^{-1}\right) T_{\mathcal{D}}^{\mathrm{spr}}(\delta/2).$$

Proof. For any $\delta \in (0,1)$, let $\tau_m = rT_{\mathcal{D}}^{\mathrm{spr}}(\delta/2)$. By Lemma 3.3, for any time $t \geq \tau_m$, the probability that $\hat{W}^i \neq W^*$ for any node i at time t is at most $\delta/2$.

On the other hand, suppose that $\hat{W}^i = W^*$ for all nodes i at time $t \geq \tau_m$. For any $\epsilon \in (0,1)$, by choosing $r = \lceil 12\epsilon^{-2}\ln(4\delta^{-1})\rceil$, we obtain from Lemma 3.2 that

$$\Pr\left(\bigcup_{i=1}^{n}\{\hat{y}_i \notin [(1-\epsilon)y, (1+\epsilon)y]\} \;\middle|\; \forall i \in V,\, \hat{W}^i = W^*\right) \leq \frac{\delta}{2}. \tag{3.5}$$

Note that, because $\epsilon \in (0,1)$, $r \leq 12\epsilon^{-2}\ln(4\delta^{-1}) + 1 \leq 18\epsilon^{-2}(1 + \ln\delta^{-1})$.

Recall that $T_{\mathrm{COMP}}^{\mathrm{cmp}}(\epsilon, \delta)$ is the smallest time τ such that, under the algorithm COMP, at any time $t \geq \tau$, all the nodes have an estimate of the function value y within a multiplicative factor of $(1 \pm \epsilon)$ with probability at least $1-\delta$. By a straightforward union bound of events and (3.5), we conclude that, for any time $t \geq \tau_m$,

$$\Pr\left(\bigcup_{i=1}^{n}\{\hat{y}_i \notin [(1-\epsilon)y, (1+\epsilon)y]\}\right) \leq \delta.$$

For any $\epsilon \in (0,1)$ and $\delta \in (0,1)$, we now have, by the definition of (ϵ, δ)-computing time,

$$\begin{aligned} T^{\text{cmp}}_{\text{COMP}}(\epsilon, \delta) &\leq \tau_m \\ &\leq 18\epsilon^{-2}\left(1 + \ln \delta^{-1}\right) T^{\text{spr}}_{\mathcal{D}}(\delta/2). \end{aligned}$$

This completes the proof of Theorem 3.4. □

3.3 Information Spreading

In this section, we analyze a randomized gossip algorithm for information spreading in which the nodes only transmit the original messages to each other. Recall that our reduction in Section 3.2.1 from minimum computation to information spreading using links of unit capacity requires that the nodes only transmit the original messages, but the information dissemination algorithm in Chapter 2 does not satisfy this property.

The method by which nodes choose partners to contact when initiating a communication and the data transmitted during the communication are the same for both the synchronous and the asynchronous time models defined in Chapter 2. These models differ in the times at which nodes contact each other: in the asynchronous model, only one node can start a communication at any time, while in the synchronous model all the nodes can communicate in each time slot.

The gossip algorithm for information spreading that we study has the same gossip mechanism as the algorithm in Chapter 2. In particular, we assume that there is a communication matrix P for the communication graph G. When a node i initiates a communication, it contacts each node $j \neq i$ with probability P_{ij}.

To present the gossip protocol for the information spreading algorithm, we introduce the following notation. Let $M_i(t)$ denote the set of messages node i has at time t. Initially, $M_i(0) = \{m_i\}$ for all $i \in V$. For a communication that occurs at time t, let t^- and t^+ denote the times immediately before and after, respectively, the communication occurs. Figure 3.2 describes the gossip protocol for the information spreading algorithm, which we call SPREAD. For each communication matrix P, there is an instance of the information spreading algorithm, which we refer to as SPREAD(P).

We note that the data transmitted between two communicating nodes in SPREAD conform to the *push and pull mechanism*. That is, when node i contacts node v at time t,

Algorithm SPREAD(P)
When a node i initiates a communication at time t:

1. Node i chooses a node v at random, and contacts v. The choice of the communication partner v is made independently of all other random choices, and the probability that node i chooses any node j is P_{ij}.
2. Nodes v and i exchange all of their messages, so that

$$M_i(t^+) = M_v(t^+) = M_i(t^-) \cup M_v(t^-).$$

Figure 3.2: A gossip algorithm for information spreading.

both nodes v and i exchange all of their information with each other. We also note that the description in the algorithm assumes that the communication links in the network have infinite capacity. As discussed in Section 3.2.1, however, an information spreading algorithm that uses links of infinite capacity can be used to compute minima using links of unit capacity (with a slowdown of a factor r).

This algorithm is simple, distributed, and satisfies the transmitter gossip constraint. We now proceed to an analysis of the information spreading time of SPREAD(P) in the two time models. In particular, we obtain an upper bound on the δ-information-spreading time of SPREAD(P) in terms of the conductance of the matrix P, which is defined as follows.

Definition 3.3. The *conductance* of a communication matrix P for G, denoted Φ_P, is defined as

$$\Phi_P = \min_{S \subset V,\, 0<|S|\leq n/2} \frac{\sum_{i \in S, j \notin S} P_{ij}}{|S|}.$$

Observe that the conductance Φ_P of P is the s-conductance Φ_P^s of P for $s = \lfloor n/2 \rfloor$. In general, the above definition of conductance is not the same as the classical definition [47]. However, we restrict our attention in this chapter to doubly stochastic matrices P. When P is doubly stochastic, these two definitions are equivalent.

Note that the definition of conductance implies that $\Phi_P \leq 1$. Throughout the remainder of the chapter, we assume that $n \geq 3$ and $\Phi_P > 0$. Without these assumptions, each node in the network would have at most one neighbor to communicate with, or the network would contain a non-empty node subset $S \subset V$ such that no node in S could

contact a node outside of S.

The following theorem provides an upper bound on the δ-information-spreading time of SPREAD(P) when P is doubly stochastic.

Theorem 3.5. *Let P be a communication matrix for G that is doubly stochastic. Then, for any $\delta \in (0, 1)$,*

$$T^{\mathrm{spr}}_{\mathrm{SPREAD}(P)}(\delta) \leq \frac{62\left(\ln n + \ln \delta^{-1}\right)}{\Phi_P}.$$

Remark. Theorem 3.5 applies to any communication matrix P such that $\sum_{i=1}^{n} P_{ij} \leq 1$ for each column j, not just those matrices in which all the column sums are equal to one.

The upper bound in Theorem 3.5 holds for both the synchronous and asynchronous time models, and as a consequence the upper bound implied by Theorem 3.4 when SPREAD is used as the information dissemination algorithm also holds in both time models. Recall that δ-information-spreading time and (ϵ, δ)-computing time are defined with respect to absolute time in both models.

We now prove Theorem 3.5 for the asynchronous and synchronous time models. To this end, for any $i \in V$, let $S_i(t) \subseteq V$ denote the set of nodes that have the message m_i after any communication events that occur at absolute time t (communication events occur on a global clock tick in the asynchronous time model, and in each time slot in the synchronous time model). At the start of the algorithm, $S_i(0) = \{i\}$.

3.3.1 Asynchronous Model

As described in Chapter 2, in the asynchronous time model the global clock ticks according to a Poisson process of rate n, and on a tick one of the n nodes is chosen uniformly at random. This node initiates a communication, so the times at which the communication events occur correspond to the ticks of the clock. On any clock tick, at most one pair of nodes can exchange messages by communicating with each other.

Let $k \geq 0$ denote the index of a clock tick. Initially, $k = 0$, and the corresponding absolute time is 0. For simplicity of notation, we identify the time at which a clock tick occurs with its index, so that $S_i(k)$ denotes the set of nodes that have the message m_i at the end of clock tick k. The following lemma provides a bound on the number of clock ticks required for every node to receive every message.

Lemma 3.6. *For any $\delta \in (0,1)$, define*

$$K(\delta) = \inf\left\{ k \geq 0 \;\middle|\; \Pr\left(\bigcup_{i=1}^{n} \{S_i(k) \neq V\} \right) \leq \delta \right\}.$$

Then,

$$K(\delta) \leq n\left(\frac{14 \ln n + 5 \ln \delta^{-1}}{\Phi_P} \right).$$

Proof. Fix any node $v \in V$. We study the evolution of the size of the set $S_v(k)$. For simplicity of notation, we drop the subscript v, and write $S(k)$ to denote $S_v(k)$.

Note that $|S(k)|$ is monotonically non-decreasing over the course of the algorithm, with the initial condition $|S(0)| = 1$. For the purpose of analysis, we divide the execution of the algorithm into two phases based on the size of the set $S(k)$. In the first phase, $|S(k)| \leq n/2$, and in the second phase $|S(k)| > n/2$.

Under the gossip algorithm, after clock tick $k+1$, we have either $|S(k+1)| = |S(k)|$ or $|S(k+1)| = |S(k)| + 1$. Further, the size increases if a node $i \in S(k)$ contacts a node $j \notin S(k)$, as in this case i will push the message m_v to j. For each such pair of nodes i, j, the probability that this occurs on clock tick $k+1$ is P_{ij}/n. Since only one node is active on each clock tick,

$$E[|S(k+1)| - |S(k)| \mid S(k)] \geq \sum_{i \in S(k), j \notin S(k)} \frac{P_{ij}}{n}. \tag{3.6}$$

When $|S(k)| \leq n/2$, it follows from (3.6) and the definition of the conductance Φ_P of P that

$$\begin{aligned} E[|S(k+1)| - |S(k)| \mid S(k)] &\geq \frac{|S(k)|}{n} \frac{\sum_{i \in S(k), j \notin S(k)} P_{ij}}{|S(k)|} \\ &\geq \frac{|S(k)|}{n} \min_{S \subset V,\, 0 < |S| \leq n/2} \frac{\sum_{i \in S, j \notin S} P_{ij}}{|S|} \\ &= \frac{|S(k)|}{n} \Phi_P. \end{aligned}$$

Let $\hat{\Phi} = \frac{\Phi_P}{n}$, so that

$$E[|S(k+1)| - |S(k)| \mid S(k)] \geq |S(k)| \hat{\Phi}. \tag{3.7}$$

We seek an upper bound on the duration of the first phase. To this end, let

$$Z(k) = \frac{\exp\left(\frac{\hat{\Phi}}{4}k\right)}{|S(k)|}.$$

Define the stopping time $L = \inf\{k \mid |S(k)| > n/2\}$, and $L \wedge k = \min(L, k)$. If $|S(k)| > n/2$, then $L \wedge (k+1) = L \wedge k$, and thus $E[Z(L \wedge (k+1)) \mid S(L \wedge k)] = Z(L \wedge k)$.

Now, suppose that $|S(k)| \leq n/2$, in which case $L \wedge (k+1) = (L \wedge k) + 1$. The function $g(z) = 1/z$ is convex for $z > 0$, which implies that, for $z_1, z_2 > 0$,

$$g(z_2) \geq g(z_1) + g'(z_1)(z_2 - z_1). \tag{3.8}$$

Applying (3.8) with $z_1 = |S(k+1)|$ and $z_2 = |S(k)|$ yields

$$\frac{1}{|S(k+1)|} \leq \frac{1}{|S(k)|} - \frac{1}{|S(k+1)|^2}(|S(k+1)| - |S(k)|).$$

Since $|S(k+1)| \leq |S(k)| + 1 \leq 2|S(k)|$, it follows that

$$\frac{1}{|S(k+1)|} \leq \frac{1}{|S(k)|} - \frac{1}{4|S(k)|^2}(|S(k+1)| - |S(k)|). \tag{3.9}$$

Combining (3.7) and (3.9), and using the fact that $1 - z \leq \exp(-z)$ for $z \geq 0$, we obtain that, if $|S(k)| \leq n/2$, then

$$\begin{aligned} E\left[\frac{1}{|S(k+1)|} \,\middle|\, S(k)\right] &\leq \frac{1}{|S(k)|}\left(1 - \frac{\hat{\Phi}}{4}\right) \\ &\leq \frac{1}{|S(k)|}\exp\left(-\frac{\hat{\Phi}}{4}\right). \end{aligned}$$

This implies that

$$\begin{aligned}
&E[Z(L \wedge (k+1)) \mid S(L \wedge k)] \\
&= E\left[\frac{\exp\left(\frac{\hat{\Phi}}{4}(L \wedge (k+1))\right)}{|S(L \wedge (k+1))|} \;\middle|\; S(L \wedge k)\right] \\
&= \exp\left(\frac{\hat{\Phi}}{4}(L \wedge k)\right) \exp\left(\frac{\hat{\Phi}}{4}\right) E\left[\frac{1}{|S((L \wedge k)+1)|} \;\middle|\; S(L \wedge k)\right] \\
&\leq \exp\left(\frac{\hat{\Phi}}{4}(L \wedge k)\right) \exp\left(\frac{\hat{\Phi}}{4}\right) \exp\left(-\frac{\hat{\Phi}}{4}\right) \frac{1}{|S(L \wedge k)|} \\
&= Z(L \wedge k).
\end{aligned}$$

Therefore, $Z(L \wedge k)$ is a supermartingale.

Since $Z(L \wedge k)$ is a supermartingale, we have the inequality $E[Z(L \wedge k)] \leq E[Z(L \wedge 0)] = 1$ for any $k > 0$, as $Z(L \wedge 0) = Z(0) = 1$. The fact that the set $S(k)$ can contain at most the n nodes in the graph implies that

$$\begin{aligned}
Z(L \wedge k) &= \frac{\exp\left(\frac{\hat{\Phi}}{4}(L \wedge k)\right)}{|S(L \wedge k)|} \\
&\geq \frac{1}{n} \exp\left(\frac{\hat{\Phi}}{4}(L \wedge k)\right). \qquad (3.10)
\end{aligned}$$

Taking expectations on both sides of (3.10) yields

$$\begin{aligned}
E\left[\exp\left(\frac{\hat{\Phi}}{4}(L \wedge k)\right)\right] &\leq nE[Z(L \wedge k)] \\
&\leq n.
\end{aligned}$$

Because $\exp(\hat{\Phi}(L \wedge k)/4) \uparrow \exp(\hat{\Phi}L/4)$ as $k \to \infty$, the monotone convergence theorem implies that

$$E\left[\exp\left(\frac{\hat{\Phi}L}{4}\right)\right] \leq n.$$

Applying Markov's inequality, we obtain that, for $k_1 = 4(\ln 2 + 2\ln n + \ln \delta^{-1})/\hat{\Phi}$,

$$\begin{aligned}\Pr(L > k_1) &= \Pr\left(\exp\left(\frac{\hat{\Phi}L}{4}\right) > \frac{2n^2}{\delta}\right) \\ &< \frac{\delta}{2n}.\end{aligned}$$

For the second phase of the algorithm, when $|S(k)| > n/2$, we study the evolution of the size of the set of nodes that do not have the message, $|S(k)^c|$. This quantity will decrease as the message spreads from nodes in $S(k)$ to nodes in $S(k)^c$. For simplicity, let us consider restarting the process from clock tick 0 after L (i.e., when more than half the nodes in the graph have the message), so that we have $|S(0)^c| \leq n/2$.

In clock tick $k+1$, a node $j \in S(k)^c$ will receive the message if it contacts a node $i \in S(k)$ and pulls the message from i. As such,

$$E[|S(k)^c| - |S(k+1)^c| \mid S(k)^c] \geq \sum_{j \in S(k)^c, i \notin S(k)^c} \frac{P_{ji}}{n}.$$

Thus, we have

$$\begin{aligned}E[|S(k+1)^c| \mid S(k)^c] &\leq |S(k)^c| - \frac{\sum_{j \in S(k)^c, i \notin S(k)^c} P_{ji}}{n} \\ &= |S(k)^c| \left(1 - \frac{\sum_{j \in S(k)^c, i \notin S(k)^c} P_{ji}}{n|S(k)^c|}\right) \\ &\leq |S(k)^c| \left(1 - \hat{\Phi}\right). \end{aligned} \tag{3.11}$$

We note that this inequality holds even when $|S(k)^c| = 0$, and as a result it is valid for all clock ticks k in the second phase. Repeated application of (3.11) yields

$$\begin{aligned}E[|S(k)^c|] &= E[E[|S(k)^c| \mid S(k-1)^c]] \\ &\leq \left(1 - \hat{\Phi}\right) E[|S(k-1)^c|] \\ &\leq \left(1 - \hat{\Phi}\right)^k E[|S(0)^c|] \\ &\leq \exp\left(-\hat{\Phi}k\right)\left(\frac{n}{2}\right).\end{aligned}$$

For $k_2 = \ln(n^2/\delta)/\hat{\Phi} = (2\ln n + \ln \delta^{-1})/\hat{\Phi}$, we have $E[|S(k_2)^c|] \leq \delta/(2n)$. Markov's

inequality now implies that the probability that not all of the nodes have the message at the end of clock tick k_2 in the second phase is at most

$$\begin{aligned}\Pr(|S(k_2)^c| > 0) &= \Pr(|S(k_2)^c| \geq 1) \\ &\leq E[|S(k_2)^c|] \\ &\leq \frac{\delta}{2n}.\end{aligned}$$

Combining the analysis of the two phases, we obtain that, for $k' = k_1 + k_2$, $\Pr(S_v(k') \neq V) \leq \delta/n$. By applying the union bound over all the nodes in the graph, using the fact that $n \geq 2$, and recalling that $\hat{\Phi} = \Phi_P/n$, we conclude that

$$\begin{aligned}K(\delta) &\leq k' \\ &= \frac{4\left(\ln 2 + 2\ln n + \ln \delta^{-1}\right) + \left(2\ln n + \ln \delta^{-1}\right)}{\hat{\Phi}} \\ &\leq n\left(\frac{14\ln n + 5\ln \delta^{-1}}{\Phi_P}\right).\end{aligned}$$

This completes the proof of Lemma 3.6. □

To extend the bound in Lemma 3.6 to absolute time, observe that Corollary 2.2 implies that the probability that $\kappa = K(\delta/3) + 27\ln(3/\delta)$ clock ticks do not occur in absolute time $(4/3)\kappa/n$ is at most $2\delta/3$. Applying the union bound now yields $T^{\text{spr}}_{\text{SPREAD}(P)}(\delta) \leq (4/3)\kappa/n \leq 62(\ln n + \ln \delta^{-1})/\Phi_P$, where the last inequality follows from the inequalities $\Phi_P \leq 1$ and $n \geq 3$. This establishes the upper bound in Theorem 3.5 for the asynchronous time model.

3.3.2 Synchronous Model

In the synchronous time model, in each time slot every node contacts a neighbor to exchange messages. Thus, n communication events may occur simultaneously. Recall that absolute time is measured in rounds or time slots in the synchronous model.

The analysis of the randomized gossip algorithm for information spreading in the synchronous model is similar to the analysis for the asynchronous model. However, we need additional analytical arguments to reach analogous conclusions due to the technical challenges presented by multiple simultaneous transmissions.

In this section, we sketch a proof of the time bound in Theorem 3.5, $T^{\mathrm{spr}}_{\mathrm{SPREAD}(P)}(\delta) \leq 62(\ln n + \ln \delta^{-1})/\Phi_P$, for the synchronous time model. Since the proof follows a similar structure as the proof of Lemma 3.6, we only point out the significant differences.

As before, we fix a node $v \in V$, and study the evolution of the size of the set $S(t) = S_v(t)$. Again, we divide the execution of the algorithm into two phases based on the evolution of $S(t)$: in the first phase $|S(t)| \leq n/2$, and in the second phase $|S(t)| > n/2$. In the first phase, we analyze the increase in $|S(t)|$, while in the second we study the decrease in $|S(t)^c|$. For the purpose of analysis, in the first phase we ignore the effect of the increase in $|S(t)|$ due to the pull aspect of protocol; that is, when node i contacts node j, we assume (for the purpose of analysis) that i sends the messages it has to j, but that j does not send any messages to i. Clearly, an upper bound obtained on the time required for every node to receive every message under this restriction is also an upper bound for the actual algorithm.

Consider a time slot $t+1$ in the first phase. For $j \notin S(t)$, let X_j be an indicator random variable that is 1 if node j receives the message m_v via a push from some node $i \in S(t)$ in time slot $t+1$, and is 0 otherwise. The probability that j does not receive m_v via a push is the probability that no node $i \in S(t)$ contacts j, and so

$$\begin{aligned} E[X_j \mid S(t)] &= 1 - \Pr(X_j = 0 \mid S(t)) \\ &= 1 - \prod_{i \in S(t)} (1 - P_{ij}) \\ &\geq 1 - \prod_{i \in S(t)} \exp(-P_{ij}) \\ &= 1 - \exp\left(- \sum_{i \in S(t)} P_{ij} \right). \end{aligned} \tag{3.12}$$

The Taylor series expansion of $\exp(-z)$ about $z = 0$ implies that, if $0 \leq z \leq 1$, then

$$\begin{aligned} \exp(-z) &\leq 1 - z + z^2/2 \\ &\leq 1 - z + z/2 \\ &= 1 - z/2. \end{aligned} \tag{3.13}$$

For a doubly stochastic matrix P, we have $0 \leq \sum_{i \in S(t)} P_{ij} \leq 1$, and so we can combine

(3.12) and (3.13) to obtain

$$E[X_j \mid S(t)] \geq \frac{1}{2} \sum_{i \in S(t)} P_{ij}.$$

By linearity of expectation,

$$\begin{aligned} E[|S(t+1)| - |S(t)| \mid S(t)] &= \sum_{j \notin S(t)} E[X_j \mid S(t)] \\ &\geq \frac{1}{2} \sum_{i \in S(t), j \notin S(t)} P_{ij} \\ &= \frac{|S(t)|}{2} \frac{\sum_{i \in S(t), j \notin S(t)} P_{ij}}{|S(t)|}. \end{aligned}$$

When $|S(t)| \leq n/2$, we have

$$E[|S(t+1)| - |S(t)| \mid S(t)] \geq |S(t)| \frac{\Phi_P}{2}. \tag{3.14}$$

Inequality (3.14) is analogous to inequality (3.7) for the asynchronous time model, with $\Phi_P/2$ in the place of $\hat{\Phi}$. We now proceed as in the proof of Lemma 3.6 for the asynchronous model. Note that $|S(t+1)| \leq 2|S(t)|$ here in the synchronous model because of the restriction in the analysis to only consider the push aspect of the protocol in the first phase, as each node in $S(t)$ can push a message to at most one other node in a single time slot. Repeating the analysis from the asynchronous model leads to the conclusion that the first phase of the algorithm ends in at most $8(\ln 2 + 2 \ln n + \ln \delta^{-1})/\Phi_P$ rounds with probability at least $1 - \delta/2n$.

The analysis of the second phase is the same as that presented for the asynchronous time model, with $\hat{\Phi}$ replaced by Φ, and thus the second phase requires at most $(2 \ln n + \ln \delta^{-1})/\Phi_P$ rounds with probability at least $1 - \delta/2n$. Combining these two bounds, we conclude that it takes at most $26(\ln n + \ln \delta^{-1})/\Phi_P$ rounds for the algorithm to spread all the messages to all the nodes with probability at least $1 - \delta$. The constant here is smaller than the corresponding one for the asynchronous model because absolute time is measured in rounds in the synchronous model, and as a consequence there is no need here to convert between clock ticks and absolute time as in the asynchronous model. This completes the proof of Theorem 3.5 for the synchronous time model.

3.4 Applications

We study here the application of our preceding results to several types of graphs. Theorems 3.4 and 3.5 imply that, given a doubly stochastic communication matrix P, the time required for our algorithm to obtain a $(1 \pm \epsilon)$ approximation with probability at least $1 - \delta$ is, up to constant factors, at most

$$\frac{\epsilon^{-2}(1 + \ln \delta^{-1})(\ln n + \ln \delta^{-1})}{\Phi_P}. \tag{3.15}$$

The classes of graphs that we consider are complete graphs, constant-degree expander graphs, and grid graphs. We use grid graphs as an example to compare the performance of our algorithm for computing separable functions with that of a known iterative averaging algorithm.

For each class of graphs, we are interested in the conductance Φ_P of a doubly stochastic communication matrix P that assigns equal probability to each of the neighbors of any node. Specifically, the probability P_{ij} that a node i contacts a node $j \neq i$ such that $(i, j) \in E$ when i becomes active is $1/\Delta$, where Δ is the maximum degree of the graph, and $P_{ii} = 1 - d_i/\Delta$, where d_i is the degree of i (setting $P_{ij} = 1/d_i$ for each neighbor j of i would only make the algorithm run faster, but the resulting matrix P would not in general be doubly stochastic). Examining (3.15), we see that, for any fixed $\epsilon \in (0, 1)$ and $\delta \in (0, 1)$, the amount of time required for our algorithm to achieve a $(1 \pm \epsilon)$ approximation with probability at least $1 - \delta$ is a function of Φ_P. We consider how this running time scales with the number of nodes n for different graphs in each class.

3.4.1 Complete Graph

On a complete graph, the communication matrix P has $P_{ii} = 0$ for $i = 1, \ldots, n$, and $P_{ij} = 1/(n - 1)$ for $j \neq i$. This regular structure allows us to directly evaluate the conductance of P, which is $\Phi_P \approx 1/2$. This implies that the (ϵ, δ)-computing time of the algorithm for computing separable functions based on SPREAD(P) is, up to constant factors, at most $\epsilon^{-2}(1 + \ln \delta^{-1})(\ln n + \ln \delta^{-1})$. Thus, for a constant $\epsilon \in (0, 1)$ and $\delta = 1/n$, the computation time scales as $O(\log^2 n)$ as n increases.

3.4.2 Expander Graph

Expander graphs have been used for numerous applications, and explicit constructions are known for constant-degree expanders [43]. We consider here undirected graphs in which the maximum degree of any vertex, Δ, is a constant. For a set of vertices $S \subseteq V$ in a graph $G = (V, E)$, let $F(S, S^c)$ be the set of edges with one endpoint in S and the other endpoint in S^c. The edge expansion of the graph is denoted by $\alpha(G)$ and defined as

$$\alpha(G) = \min_{S \subset V,\, 0<|S|\leq n/2} \frac{|F(S, S^c)|}{|S|}.$$

In a family of expander graphs of various different sizes, $G_1, G_2, \ldots$, the edge expansion is bounded from below by $\alpha(G_\ell) \geq \alpha$ for each graph G_ℓ, where α is a positive constant. For a graph in such a family, the communication matrix P satisfies $P_{ij} = 1/\Delta$ for all $i \neq j$ such that $(i, j) \in E$, from which we obtain $\Phi_P \geq \alpha/\Delta$. When α and Δ are constants, this leads to a similar conclusion as in the case of the complete graph: for any constant $\epsilon \in (0, 1)$ and $\delta = 1/n$, the computation time is $O(\log^2 n)$.

3.4.3 Grid Graph

We now consider a d-dimensional grid graph on n nodes, where $c = n^{1/d}$ is an integer. Each node in the grid can be represented as a d-dimensional vector $a = (a_i)$, where $a_i \in \{1, \ldots, c\}$ for $1 \leq i \leq d$. There is one node for each distinct vector of this type, and so the total number of nodes in the graph is $c^d = (n^{1/d})^d = n$. For any two nodes a and b, there is an edge (a, b) in the graph if and only if, for some $i \in \{1, \ldots, d\}$, $|a_i - b_i| = 1$, and $a_j = b_j$ for all $j \neq i$.

In [2], it is shown that the edge expansion of this grid graph is

$$\min_{S \subset V,\, 0<|S|\leq n/2} \frac{|F(S, S^c)|}{|S|} = \Theta\left(\frac{1}{c}\right) = \Theta\left(\frac{1}{n^{1/d}}\right).$$

By the definition of the edge set, the maximum degree of a node in the graph is $2d$. This means that $P_{ij} = 1/(2d)$ for all $i \neq j$ such that $(i, j) \in E$, and it follows that $\Phi_P = \Omega\left(\frac{1}{dn^{1/d}}\right)$. Hence, for any $\epsilon \in (0, 1)$ and $\delta \in (0, 1)$, the (ϵ, δ)-computing time of the algorithm for computing separable functions is $O(\epsilon^{-2}(1 + \log \delta^{-1})(\log n + \log \delta^{-1})dn^{1/d})$.

3.4.4 Comparison with Iterative Averaging

We briefly contrast the performance of our algorithm for computing separable functions with that of some iterative averaging algorithms. For iterative averaging algorithms such as the ones in [49] and [7], the convergence time largely depends on the mixing time of the communication matrix P, which is bounded from below by $\Omega(1/\Phi_P)$ (see [47], for example).

Our algorithm can be used to compute the average of a set of numbers by estimating both the sum of the numbers and the number of nodes in the graph. From (3.15), we see that when the network size n and the accuracy parameters ϵ and δ are fixed, the running time of our algorithm scales in proportion to $1/\Phi_P$, a factor that captures the dependence of the algorithm on the matrix P. Thus, our algorithm is (up to a $\ln n$ factor) no slower than the fastest iterative algorithm based on time-invariant linear dynamics.

When our algorithm is used to compute the average of a set of numbers on a d-dimensional grid graph, it follows from the analysis in Section 3.4.3 that the amount of time required to ensure the estimate is within a $(1\pm\epsilon)$ factor of the average with probability at least $1-\delta$ is, up to constant factors, at most $\epsilon^{-2}(1+\ln\delta^{-1})(\ln n+\ln\delta^{-1})dn^{1/d}$ for any $\epsilon\in(0,1)$ and $\delta\in(0,1)$. So, for a constant $\epsilon\in(0,1)$ and $\delta=1/n$, the computation time scales as $O(dn^{1/d}\log^2 n)$ with the size of the graph, n. The algorithm in [7] requires $\Omega(n^{2/d}\log n)$ time for this computation. Hence, the running time of our algorithm is (for fixed d, and up to logarithmic factors) the square root of the running time of the iterative algorithm. This relationship also holds on other graphs for which the spectral gap is proportional to the square of the conductance.

Chapter 4

Distributed Convex Optimization

In this chapter, we explore an application of the algorithm in Chapter 3 for approximately computing separable functions. We show that the algorithm can be used as a subroutine for computing sums in a simple distributed algorithm for a class of convex optimization problems. The optimization algorithm effectively reduces the convex optimization problem to the problem of computing sums via the technique of dual gradient ascent.

As in Chapters 2 and 3, we consider a connected network of n nodes described by a communication graph $G = (V, E)$ with $V = \{1, \ldots, n\}$. Each node is assigned a non-negative variable x_i. The goal is to choose values for the x_i variables to optimize a global network objective function. We assume that the global objective function f is separable, so that $f(x) = \sum_{i=1}^{n} f_i(x_i)$. The feasible region is described by a set of linear constraints.

This class of optimization problems captures some key operational network problems, such as routing and congestion control.

Example 4.1 (Network resource allocation). Given a capacitated network $G = (V, E)$, users wish to transfer data to specific destinations. Each user is associated with a particular path in the network, and has a utility function that depends on the rate x_i that the user is allocated. The goal is to maximize the global network utility, which is the sum of the utilities of individual users. The rate allocation $x = [x_1 \cdots x_n]^T$ must satisfy capacity constraints, which are linear [22].

Our algorithm for convex optimization is based on the Lagrange dual problem. Due to the separable primal objective function, the dual problem can be decomposed so that an individual node can recover the value of its variable in a primal solution from a dual

feasible solution. We solve the dual problem via a dual gradient ascent algorithm. To respect the non-negativity constraints on the primal variables in the gradient ascent, we introduce a barrier function that is inspired by (centralized) interior-point mathematical programming algorithms. Each iteration of the gradient ascent procedure requires the computation of one component of the gradient for each constraint in the primal problem. This computation involves a sum that has one term for each node, and as such the gossip algorithm from Chapter 3 can be used to compute the gradient.

Since the gossip algorithm only approximates sums, the convex optimization algorithm only computes approximate solutions to the convex optimization problem. Given an error parameter ϵ, the optimization algorithm produces an ϵ-approximately feasible solution with objective function value close to that of an optimal feasible solution. The number of iterations of the gradient ascent procedure performed by the algorithm depends on ϵ and a measure of curvature variation in the dual objective function. Each iteration requires one invocation of the summation algorithm for each constraint, and as such the time required for an iteration depends on the number of constraints and the running time of the algorithm in Chapter 3.

4.1 Model and Overview

We consider an undirected communication graph $G = (V, E)$ with $V = \{1, \ldots, n\}$, where each node i has a non-negative decision variable $x_i \geq 0$. The vector $x \in \mathbf{R}^n$ contains the variables in the optimization problem.

Throughout this chapter, $\|v\|$ denotes the ℓ_2-norm of a vector $v \in \mathbf{R}^d$, which is defined as

$$\|v\| = \sqrt{\sum_{i=1}^{d} v_i^2}.$$

The ball of radius r about the point v is denoted by $B(v, r)$ and is defined as

$$B(v, r) = \{w \mid \|w - v\| \leq r\}.$$

We consider optimization problems of the following general form. The objective function is

$$f(x) = \sum_{i=1}^{n} f_i(x_i),$$

and we assume that each $f_i : \mathbf{R}_+ \to \mathbf{R}$ has a continuous second derivative and is convex, with $\lim_{x_i \downarrow 0} f_i'(x_i) < \infty$ and $\lim_{x_i \uparrow \infty} f_i'(x_i) = \infty$. The constraints are linear equality constraints of the form $Ax = b$, specified by a matrix $A \in \mathbf{R}_+^{m \times n}$ and a vector $b \in \mathbf{R}_{++}^m$, and non-negativity constraints $x_i \geq 0$ on the variables. Section 4.5 describes modifications to our approach for handling linear inequality constraints. We assume that $m \leq n$, and the matrix A has linearly independent rows. For $i = 1, \ldots, n$, let $a_i = [A_{1i} \cdots A_{mi}]^T$ denote the ith column of the matrix A. In this distributed setting, we assume that node i is given the vectors b and a_i, but not the other columns of the matrix A.

For a real matrix M, we write $\sigma_{\min}(M)$ and $\sigma_{\max}(M)$ to denote the smallest and largest singular values, respectively, of M, so that $\sigma_{\min}(M)^2$ and $\sigma_{\max}(M)^2$ are the smallest and largest eigenvalues of $M^T M$. Note that $\sigma_{\min}(M) = \min\{\|Mv\| \mid \|v\| = 1\}$ and $\sigma_{\max}(M) = \max\{\|Mv\| \mid \|v\| = 1\}$. If M is symmetric, then the singular values and the eigenvalues of M coincide, so $\sigma_{\min}(M)$ and $\sigma_{\max}(M)$ are the smallest and largest eigenvalues of M.

We refer to the following convex optimization problem as the primal problem:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & Ax = b, \\ & x_i \geq 0, \quad i = 1, \ldots, n. \end{array} \tag{P}$$

Let OPT denote the optimal value of (P). Associated with the primal problem (P) is the Lagrangian function

$$L(x, \lambda, \nu) = f(x) + \lambda^T (Ax - b) - \nu^T x,$$

which is defined for $\lambda \in \mathbf{R}^m$ and $\nu \in \mathbf{R}^n$, and the Lagrange dual function

$$\begin{aligned} g(\lambda, \nu) &= \inf_{x \in \mathbf{R}_+^n} L(x, \lambda, \nu) \\ &= -b^T \lambda + \sum_{i=1}^{n} \inf_{x_i \in \mathbf{R}_+} \left(f_i(x_i) + \left(a_i^T \lambda - \nu_i \right) x_i \right). \end{aligned}$$

The Lagrange dual problem to (P) is

$$\begin{array}{ll} \text{maximize} & g(\lambda, \nu) \\ \text{subject to} & \nu_i \geq 0, \quad i = 1, \ldots, n. \end{array} \tag{D}$$

Although we seek a solution to the primal problem (P), to avoid directly enforcing the non-negativity constraints, we introduce a logarithmic barrier. For a parameter $\theta > 0$, we consider the primal barrier problem

$$\begin{array}{ll} \text{minimize} & f(x) - \theta \sum_{i=1}^{n} \ln x_i \\ \text{subject to} & Ax = b. \end{array} \tag{P_θ}$$

The Lagrange dual function corresponding to (P_θ) is

$$g_\theta(\lambda) = -b^T \lambda + \sum_{i=1}^{n} \inf_{x_i \in \mathbf{R}_{++}} \left(f_i(x_i) - \theta \ln x_i + a_i^T \lambda x_i \right),$$

and the associated Lagrange dual problem is the unconstrained optimization problem

$$\text{maximize} \quad g_\theta(\lambda). \tag{D_θ}$$

We assume that the primal barrier problem (P_θ) is feasible; that is, there exists a vector $x \in \mathbf{R}^n_{++}$ such that $Ax = b$. Under this assumption, the optimal value of (P_θ) is finite, and Slater's condition implies that the dual problem (D_θ) has the same optimal value, and there exists a dual solution λ^* that achieves this optimal value [8]. Furthermore, because (D_θ) is an unconstrained maximization problem with a strictly concave objective function, the optimal solution λ^* is unique.

For a vector of dual variables $\lambda \in \mathbf{R}^m$, let $x(\lambda) \in \mathbf{R}^n_{++}$ denote the corresponding primal minimizer in the Lagrange dual function: for $i = 1, \ldots, n$,

$$x_i(\lambda) = \arg \inf_{x_i \in \mathbf{R}_{++}} \left(f_i(x_i) - \theta \ln x_i + a_i^T \lambda x_i \right). \tag{4.1}$$

We can solve for each $x_i(\lambda)$ explicitly. As $f_i(x_i) - \theta \ln x_i + a_i^T \lambda x_i$ is convex in x_i,

$$f_i'(x_i(\lambda)) - \frac{\theta}{x_i(\lambda)} + a_i^T \lambda = 0. \tag{4.2}$$

For $i = 1, \ldots, n$, define the function $h_i : \mathbf{R}_{++} \to \mathbf{R}$ as

$$h_i(x_i) = f_i'(x_i) - \frac{\theta}{x_i}.$$

Since

$$h_i'(x_i) = f_i''(x_i) + \frac{\theta}{x_i^2}$$

and f_i is convex, h_i is strictly increasing. Thus, h_i^{-1} is a well-defined and strictly increasing function. Since $\lim_{x_i \downarrow 0} f_i'(x_i) < \infty$ and $\lim_{x_i \uparrow \infty} f_i'(x_i) = \infty$, the inverse function $h_i^{-1}(y)$ is defined for all $y \in \mathbf{R}$. We now have $x_i(\lambda) = h_i^{-1}\left(-a_i^T \lambda\right)$.

Also, we assume that, given a vector λ, a node i can compute $x_i(\lambda)$. This is reasonable since computing $x_i(\lambda)$ is simply an unconstrained convex optimization problem in a single variable as in (4.1), which can be done by several methods, such as Newton's method.

Next, in our convergence analysis, we will argue about the gradient of the Lagrange dual function g_θ. A component $j \in \{1, \ldots, m\}$ of the gradient of the Lagrange dual function g_θ is

$$\begin{aligned}
\frac{\partial g_\theta(\lambda)}{\partial \lambda_j} &= -b_j + \sum_{i=1}^n \frac{\partial}{\partial \lambda_j} \left(f_i(x_i(\lambda)) - \theta \ln x_i(\lambda) + a_i^T \lambda x_i(\lambda) \right) \\
&= -b_j + \sum_{i=1}^n \left(f_i'(x_i(\lambda)) \frac{\partial x_i(\lambda)}{\partial \lambda_j} - \left(\frac{\theta}{x_i(\lambda)} \right) \frac{\partial x_i(\lambda)}{\partial \lambda_j} + a_i^T \lambda \frac{\partial x_i(\lambda)}{\partial \lambda_j} + A_{ji} x_i(\lambda) \right) \\
&= -b_j + \sum_{i=1}^n A_{ji} x_i(\lambda),
\end{aligned}$$

where the last equality follows from (4.2). So the gradient of g_θ is

$$\begin{aligned}
\nabla g_\theta(\lambda) &= -b + \sum_{i=1}^n a_i x_i(\lambda) \\
&= Ax(\lambda) - b.
\end{aligned} \tag{4.3}$$

We will use $p(\lambda)$ to denote $\|\nabla g_\theta(\lambda)\| = \|Ax(\lambda) - b\|$ for a vector $\lambda \in \mathbf{R}^m$. We note that

at the optimal dual solution λ^*, we have $p(\lambda^*) = 0$ and $Ax(\lambda^*) = b$.

To control the rate of decrease in the gradient norm $p(\lambda)$, we must understand the Hessian of g_θ. For $j_1, j_2 \in \{1, \ldots, m\}$, component (j_1, j_2) of the Hessian $\nabla^2 g_\theta(\lambda)$ of g_θ at a point λ is

$$\begin{aligned}\frac{\partial g_\theta(\lambda)}{\partial \lambda_{j_1} \partial \lambda_{j_2}} &= \sum_{i=1}^{n} A_{j_1 i} \frac{\partial x_i(\lambda)}{\partial \lambda_{j_2}} \\ &= -\sum_{i=1}^{n} A_{j_1 i} A_{j_2 i} \left(h_i^{-1}\right)' \left(-a_i^T \lambda\right). \end{aligned} \tag{4.4}$$

As the functions h_i^{-1} are strictly increasing, $\min_{\ell=1,\ldots,n} \left(\left(h_\ell^{-1}\right)' \left(-a_\ell^T \lambda\right) \right) > 0$. Hence, for any $\mu \in \mathbf{R}^m$ other than the zero vector,

$$\begin{aligned}\mu^T \nabla^2 g_\theta(\lambda) \mu &= \sum_{j_1=1}^{m} \mu_{j_1} \sum_{j_2=1}^{m} \frac{\partial g_\theta(\lambda)}{\partial \lambda_{j_1} \partial \lambda_{j_2}} \mu_{j_2} \\ &= -\sum_{j_1=1}^{m} \mu_{j_1} \sum_{j_2=1}^{m} \sum_{i=1}^{n} A_{j_1 i} A_{j_2 i} \left(h_i^{-1}\right)' \left(-a_i^T \lambda\right) \mu_{j_2} \\ &= -\sum_{i=1}^{n} \left(h_i^{-1}\right)' \left(-a_i^T \lambda\right) \sum_{j_1=1}^{m} A_{j_1 i} \mu_{j_1} \sum_{j_2=1}^{m} A_{j_2 i} \mu_{j_2} \\ &= -\sum_{i=1}^{n} \left(h_i^{-1}\right)' \left(-a_i^T \lambda\right) \left(a_i^T \mu\right)^2 \\ &\leq -\min_{\ell=1,\ldots,n} \left(\left(h_\ell^{-1}\right)' \left(-a_\ell^T \lambda\right) \right) \left(A^T \mu\right)^T \left(A^T \mu\right) \qquad (4.5) \\ &< 0, \end{aligned}$$

and g_θ is a strictly concave function.

4.1.1 Organization

Section 4.2 presents our distributed algorithm for solving a convex optimization problem in the class described in Section 4.1, under the assumption that certain parameters of the problem instance are known to the algorithm. An analysis of the convergence rate of the algorithm appears in Section 4.3. Section 4.4 describes how to set and efficiently search for the necessary parameter values. In Section 4.5, we discuss modifications to our algorithm,

which is presented in the case of linear equality constraints, for handling linear inequality constraints.

4.2 Algorithm Description

This section describes our approach to approximately solving the primal problem (P). In Section 4.2.1, we present the dual gradient ascent procedure that is the basis of the convex optimization algorithm. Section 4.2.2 specifies the values of the parameters used in the gradient ascent procedure.

4.2.1 Basic Algorithm

We consider an iterative algorithm for obtaining an approximate solution to (P), which uses gradient ascent for the dual barrier problem (D_θ). The algorithm generates a sequence of feasible solutions λ^0, λ^1, λ^2, ... for (D_θ), where λ^0 is the initial vector. To update λ^{k-1} to λ^k in an iteration k, the algorithm uses the gradient $\nabla g_\theta\left(\lambda^{k-1}\right)$ to determine the direction of the difference $\lambda^k - \lambda^{k-1}$. We assume that the algorithm is given as inputs the initial point λ^0, and an accuracy parameter ϵ such that $0 < \epsilon \leq 1$. The goal of the algorithm is to find a point $x \in \mathbf{R}^n_+$ that is nearly feasible in the sense that $\|Ax-b\| \leq \epsilon\|b\|$, and that has objective function value close to that of an optimal feasible point.

In this section, we describe the operation of the algorithm under the assumption that it has knowledge of certain parameters that affect its execution and performance. We refer to an execution of the algorithm with a particular set of parameters as an *inner run* of the algorithm. To address the fact that these parameters are not available to the algorithm at the outset, we add an *outer loop* to the algorithm. The outer loop uses binary search to find appropriate values for the parameters, and performs an inner run for each set of parameters encountered during the search. Section 4.4 discusses the operation of the outer loop of the algorithm.

An inner run of the algorithm consists of a sequence of iterations. Iteration k, for $k = 1, 2, \ldots$, begins with a current vector of dual variables λ^{k-1}, from which each node i computes $x_i\left(\lambda^{k-1}\right)$. Let $s^{k-1} = Ax\left(\lambda^{k-1}\right)$, so that, by (4.3), $\nabla g_\theta\left(\lambda^{k-1}\right) = s^{k-1} - b$.

In order for the algorithm to perform gradient ascent, each node must compute the vector s^{k-1}. A component $s_j^{k-1} = \sum_{i=1}^n A_{ji}x_i\left(\lambda^{k-1}\right)$ of s^{k-1} is the sum of the values

$y_i = A_{ji} x_i \left(\lambda^{k-1}\right)$ for those nodes i such that $A_{ji} > 0$. As such, any algorithm for computing sums of this form can be used as a subroutine for the gradient ascent.

Our implementation of the gradient ascent uses the distributed gossip algorithm from Chapter 3 as the summation subroutine. To adapt the gossip algorithm to this setting, in which some of the terms in the sum that defines s_j^{k-1} may be zero, we change the way the exponential random variables are generated in the gossip algorithm. During the computation of a component s_j^{k-1}, each node i with $A_{ji} = 0$ does not generate any exponential random variables, and participates in the minimum computation process using ∞ in place of the value of each random variable.

Recall that the algorithm from Chapter 3 approximately computes sums. For each component $j = 1, \ldots, m$ of the vector s^{k-1}, the nodes apply the gossip algorithm to compute an estimate $\hat{s}_j^{k-1}$ of s_j^{k-1}. Let ϵ_1 be the accuracy parameter and δ be the error probability parameter provided as input to the gossip algorithm. Then, for each component j, the estimate $\hat{s}_j^{k-1}$ the summation subroutine produces will satisfy

$$(1 - \epsilon_1)\, s_j^{k-1} \leq \hat{s}_j^{k-1} \leq (1 + \epsilon_1)\, s_j^{k-1} \tag{4.6}$$

with probability at least $1 - \delta$. Let $\hat{s}^{k-1} \in \mathbf{R}^m$ denote the vector containing the estimates $\hat{s}_j^{k-1}$ for $j = 1, \ldots, m$.

We discuss the choice of the accuracy parameter ϵ_1 in Section 4.2.2. In the analysis of an inner run, we assume that each invocation of the summation routine succeeds, so that (4.6) is satisfied. In Section 4.4, we describe how to choose δ to be sufficiently small so that this assumption will hold with high probability.

A description of an iteration k of an inner run of the algorithm is shown in Figure 4.1. We specify values for the step size t and the error tolerance ϵ_1 in the next section. An inner run is essentially standard gradient ascent, where the stopping criterion (sufficiently small gradient norm) is modified to reflect the potential error in nodes' estimates of the gradient. Note that (4.8) does not imply (4.7); the nodes must check both of the two conditions, because the error tolerance ϵ_1 for the summation subroutine could be much smaller than ϵ. The summation subroutine ensures that all nodes obtain a common estimate of the sum, and as a consequence either all or no nodes will determine that both stopping conditions are met in a given iteration.

Iteration k

1. For $j = 1, \ldots, m$, the nodes compute an estimate $\hat{s}_j^{k-1}$ of $s_j^{k-1} = \sum_{i=1}^{n} A_{ji} x_i \left(\lambda^{k-1}\right)$.
2. The nodes check the following two stopping conditions.

$$(1 - \epsilon_1)\left(1 - \frac{2}{3}\epsilon\right)\|b\| \leq \left\|\hat{s}^{k-1}\right\| \leq (1 + \epsilon_1)\left(1 + \frac{2}{3}\epsilon\right)\|b\|. \tag{4.7}$$

$$\left\|\hat{s}^{k-1} - b\right\| \leq \left(\frac{2}{3}\epsilon + \epsilon_1\left(\frac{1 + \epsilon_1}{1 - \epsilon_1}\right)\left(1 + \frac{2}{3}\epsilon\right)\right)\|b\|. \tag{4.8}$$

   If both conditions (4.7) and (4.8) are satisfied, the inner run terminates, producing as output the vector $x\left(\lambda^{k-1}\right)$.
3. The nodes update the dual vector by setting $\Delta\lambda^{k-1} = \hat{s}^{k-1} - b$, and $\lambda^k = \lambda^{k-1} + t\Delta\lambda^{k-1}$.

Figure 4.1: The kth iteration of an inner run.

4.2.2 Choosing Parameters

The step size t and the convergence rate of our algorithm are governed by the variation in curvature of the Lagrange dual function. (This is standard in a dual ascent context; intuitively, regions of large curvature necessitate a small step size to guarantee convergence, and if small steps are taken in regions with small curvature, then progress toward an optimal solution is slow.) Examining the Hessian of the Lagrange dual function in (4.4), we see that the curvature variation depends both on variation in $(h_i^{-1})'$, which roughly corresponds to variation in the curvature of the f_i's, and on the variation in the singular values of A^T. Precisely, note that

$$\begin{aligned}\left(h_i^{-1}\right)'\left(-a_i^T\lambda\right) &= \frac{1}{h_i'\left(h_i^{-1}\left(-a_i^T\lambda\right)\right)} \\ &= \frac{1}{f_i''\left(h_i^{-1}\left(-a_i^T\lambda\right)\right) + \frac{\theta}{\left(h_i^{-1}\left(-a_i^T\lambda\right)\right)^2}}.\end{aligned}$$

The fact that each function f_i has a continuous second derivative implies that the derivative

of h_i^{-1} is continuous as well. For a distance $r > 0$, define

$$q_f(r) = \min_{\ell=1,\ldots,n} \min_{\lambda \in B(\lambda^*, r)} \left(h_\ell^{-1}\right)' \left(-a_\ell^T \lambda\right);$$
$$Q_f(r) = \max_{\ell=1,\ldots,n} \max_{\lambda \in B(\lambda^*, r)} \left(h_\ell^{-1}\right)' \left(-a_\ell^T \lambda\right).$$

Our step size and convergence rate will depend on the parameters defined as follows:

$$q = q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right) \sigma_{\min}\left(A^T\right)^2;$$
$$Q = Q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right) \sigma_{\max}\left(A^T\right)^2;$$
$$R = \frac{Q}{q}.$$

For simplicity of notation, we have suppressed the dependence of these parameters on $\left\|\lambda^0 - \lambda^*\right\|$ and the matrix A. Note that $R \geq 1$. These parameters measure the minimum and maximum curvature variation of the Lagrange dual function only in a ball of radius $\left\|\lambda^0 - \lambda^*\right\|$ around the optimal dual solution λ^*; this is because the sequence of dual solutions generated by our algorithm grows monotonically closer to λ^*, as shown below in Lemma 4.4, and we are concerned only with variation in the region in which our algorithm executes (as opposed to the entire feasible region, which is all of $\mathbf{R}^m$). Thus a better initial estimate of the optimal dual solution yields a tighter bound on curvature variation and a better convergence result.

When we analyze the inner run, we assume that both q and Q are known to the algorithm. We discharge this assumption in Section 4.4 using standard binary search techniques.

We define $\alpha = 1/(6R) = q/(6Q)$. For the summation subroutine, nodes use the accuracy parameter $\epsilon_1 = \epsilon\alpha/3$, where ϵ is the error tolerance given to the convex optimization algorithm. For gradient ascent, nodes compute and employ the following step size:

$$t = \frac{\left(1 - \alpha\left(\frac{1}{2} + \frac{\epsilon}{3}\right)\right)^2 - \frac{1}{6}\left(\frac{1}{2} + \frac{\epsilon}{3}\right)\left(1 + \alpha\left(\frac{1}{2} + \frac{\epsilon}{3}\right)\right)}{\left(1 + \alpha\left(\frac{1}{2} + \frac{\epsilon}{3}\right)\right)^2 QR}. \tag{4.9}$$

We have $t > 0$ since $\alpha \leq 1/6$ and $\epsilon \leq 1$. Note that $t = \Theta(q/Q^2)$. An inner run continues to execute iterations for increasing values of k until both stopping conditions are satisfied, or the outer loop of the algorithm terminates the inner run as described in Section 4.4.

4.3 Convergence Analysis

In this section, we provide an analysis of the number of iterations required for an inner run of the algorithm to obtain a solution $x\left(\lambda^k\right)$ such that $\left\|Ax\left(\lambda^k\right)-b\right\| \leq \epsilon\|b\|$, and we also prove an approximation bound on the objective function value of the final solution. We assume in this analysis that the summation subroutine used by an inner run is always successful; that is, (4.6) holds for every sum computation. Furthermore, we assume that an inner run executes until both stopping conditions are satisfied. The possibility of an inner run being terminated by the outer loop is addressed in Section 4.4.

First, we consider the extent to which the update direction $\Delta\lambda^{k-1}$ deviates from the correct gradient $\nabla g_\theta\left(\lambda^{k-1}\right)$, provided that the inner run does not terminate in iteration k. To this end, let $u^{k-1}=\hat{s}^{k-1}-s^{k-1}$ be a vector representing the error in the computation of s^{k-1}. Note that $\Delta\lambda^{k-1}=\nabla g_\theta\left(\lambda^{k-1}\right)+u^{k-1}$. The following lemma shows that the error introduced by the approximate summation subroutine is small relative to our key measure of progress, the gradient norm.

Lemma 4.1. *If the stopping conditions* (4.7) *and* (4.8) *are not both satisfied in iteration* k*, then*

$$\left\|u^{k-1}\right\| \leq \alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\| \tag{4.10}$$

and

$$\left(1-\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\| \leq \left\|\Delta\lambda^{k-1}\right\| \leq \left(1+\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\|. \tag{4.11}$$

Proof. As the inequalities in (4.11) follow from (4.10) and the triangle inequality, we focus on proving the inequality in (4.10). If (4.7) is not satisfied, then

$$\left\|\hat{s}^{k-1}\right\|<(1-\epsilon_1)\left(1-\frac{2}{3}\epsilon\right)\|b\| \text{ or } \left\|\hat{s}^{k-1}\right\|>(1+\epsilon_1)\left(1+\frac{2}{3}\epsilon\right)\|b\|,$$

and so, by (4.6),

$$\left\|s^{k-1}\right\|<\left(1-\frac{2}{3}\epsilon\right)\|b\| \text{ or } \left\|s^{k-1}\right\|>\left(1+\frac{2}{3}\epsilon\right)\|b\|.$$

By the triangle inequality, this implies that

$$\begin{aligned}\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\| &= \left\|s^{k-1}-b\right\| \\ &\geq \left|\left\|s^{k-1}\right\| - \|b\|\right| \\ &> \frac{2}{3}\epsilon\|b\|. \end{aligned} \tag{4.12}$$

Suppose that (4.7) is satisfied and (4.8) is not satisfied. Note that (4.6) implies that $\left\|u^{k-1}\right\| \leq \epsilon_1 \left\|s^{k-1}\right\|$, and so (4.7) and (4.6) yield

$$\begin{aligned}\left\|u^{k-1}\right\| &\leq \epsilon_1 \left\|s^{k-1}\right\| \\ &\leq \epsilon_1 \left(\frac{1+\epsilon_1}{1-\epsilon_1}\right)\left(1+\frac{2}{3}\epsilon\right)\|b\|. \end{aligned} \tag{4.13}$$

By the triangle inequality and (4.13),

$$\begin{aligned}\left\|\Delta\lambda^{k-1}\right\| &= \left\|\hat{s}^{k-1}-b\right\| \\ &= \left\|\nabla g_\theta\left(\lambda^{k-1}\right)+u^{k-1}\right\| \\ &\leq \left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\| + \left\|u^{k-1}\right\| \\ &\leq \left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\| + \epsilon_1\left(\frac{1+\epsilon_1}{1-\epsilon_1}\right)\left(1+\frac{2}{3}\epsilon\right)\|b\|, \end{aligned}$$

and so the fact that (4.8) is not satisfied implies that

$$\begin{aligned}\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\| &\geq \left\|\hat{s}^{k-1}-b\right\| - \epsilon_1\left(\frac{1+\epsilon_1}{1-\epsilon_1}\right)\left(1+\frac{2}{3}\epsilon\right)\|b\| \\ &> \left(\frac{2}{3}\epsilon + \epsilon_1\left(\frac{1+\epsilon_1}{1-\epsilon_1}\right)\left(1+\frac{2}{3}\epsilon\right)\right)\|b\| - \epsilon_1\left(\frac{1+\epsilon_1}{1-\epsilon_1}\right)\left(1+\frac{2}{3}\epsilon\right)\|b\| \\ &= \frac{2}{3}\epsilon\|b\|. \end{aligned} \tag{4.14}$$

Combining (4.12) and (4.14), it follows that if the two stopping conditions are not both satisfied, then

$$\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\| > \frac{2}{3}\epsilon\|b\|.$$

Now, applying the triangle inequality yields

$$\begin{aligned}\left\|u^{k-1}\right\| &\leq \epsilon_1 \left\|s^{k-1}\right\| \\ &\leq \epsilon_1 \left(\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\| + \|b\|\right) \\ &\leq \epsilon_1 \left(1 + \frac{3}{2\epsilon}\right) \left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\| \\ &= \alpha \left(\frac{1}{2} + \frac{\epsilon}{3}\right) \left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\|,\end{aligned}$$

where the last equality follows from the fact that $\epsilon_1 = \epsilon\alpha/3$. This proves the inequality in (4.10), and completes the proof of the lemma. □

Next, we develop some inequalities that will be useful in understanding the evolution of an inner run from one iteration to the next. The following lemma translates the parameters q and Q of Section 4.2.2 to inequalities that bound the variation in the gradient at different dual points.

Lemma 4.2. *For any two points* $\rho^1, \rho^2 \in B\left(\lambda^*, \left\|\lambda^0 - \lambda^*\right\|\right)$,

$$\left\|Ax\left(\rho^2\right) - Ax\left(\rho^1\right)\right\| \leq Q\left\|\rho^2 - \rho^1\right\| \tag{4.15}$$

and

$$\left(\nabla g_\theta\left(\rho^2\right) - \nabla g_\theta\left(\rho^1\right)\right)^T \left(\rho^2 - \rho^1\right) \leq -q\left\|\rho^2 - \rho^1\right\|^2. \tag{4.16}$$

Proof. Let $\left[\rho^1, \rho^2\right]$ denote the line segment joining ρ^1 and ρ^2. Since $B\left(\lambda^*, \left\|\lambda^0 - \lambda^*\right\|\right)$ is a convex set, for any $i = 1, \ldots, n$ and any $\lambda \in \left[\rho^1, \rho^2\right]$, $\left(h_i^{-1}\right)'\left(-a_i^T\lambda\right) \leq Q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right)$. As a result,

$$\begin{aligned}\left|x_i\left(\rho^2\right) - x_i\left(\rho^1\right)\right| &= \left|h_i^{-1}\left(-a_i^T\rho^2\right) - h_i^{-1}\left(-a_i^T\rho^1\right)\right| \\ &\leq Q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right)\left|a_i^T\left(\rho^2 - \rho^1\right)\right| \\ &= Q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right) a_i^T\rho,\end{aligned}$$

where $\rho \in \mathbf{R}^m$ is defined by $\rho_j = |\rho_j^2 - \rho_j^1|$ for $j = 1, \ldots, m$. This implies that

$$\begin{aligned}\left\|Ax\left(\rho^2\right) - Ax\left(\rho^1\right)\right\| &= \left\|A\left(x\left(\rho^2\right) - x\left(\rho^1\right)\right)\right\| \\ &\leq Q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right)\left\|AA^T\rho\right\| \\ &\leq Q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right)\sigma_{\max}\left(AA^T\right)\|\rho\| \\ &= Q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right)\sigma_{\max}\left(A^T\right)^2\left\|\rho^2 - \rho^1\right\|,\end{aligned}$$

and the inequality in (4.15) is proved.

For any $\lambda \in \left[\rho^1, \rho^2\right]$ and any $\mu \in \mathbf{R}^m$, a calculation analogous to the one in (4.5) yields

$$\begin{aligned}\mu^T\nabla^2 g_\theta(\lambda)\mu &= -\sum_{j_1=1}^{m}\mu_{j_1}\sum_{j_2=1}^{m}\sum_{i=1}^{n}A_{j_1 i}A_{j_2 i}\left(h_i^{-1}\right)'\left(-a_i^T\lambda\right)\mu_{j_2} \\ &\leq -q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right)\mu^T AA^T\mu \\ &\leq -q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right)\sigma_{\min}\left(AA^T\right)\mu^T\mu \\ &= -q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right)\sigma_{\min}\left(A^T\right)^2\|\mu\|^2. \qquad (4.17)\end{aligned}$$

From the second-order expansion of the function g_θ, there exist vectors $\mu^1, \mu^2 \in \left[\rho^1, \rho^2\right]$ such that

$$g_\theta\left(\rho^2\right) = g_\theta\left(\rho^1\right) + \nabla g_\theta\left(\rho^1\right)^T\left(\rho^2 - \rho^1\right) + \frac{1}{2}\left(\rho^2 - \rho^1\right)^T\nabla^2 g_\theta\left(\mu^1\right)\left(\rho^2 - \rho^1\right)$$

and

$$g_\theta\left(\rho^1\right) = g_\theta\left(\rho^2\right) + \nabla g_\theta\left(\rho^2\right)^T\left(\rho^1 - \rho^2\right) + \frac{1}{2}\left(\rho^1 - \rho^2\right)^T\nabla^2 g_\theta\left(\mu^2\right)\left(\rho^1 - \rho^2\right).$$

Adding these two equations and applying (4.17) yields

$$\begin{aligned}&\left(\nabla g_\theta\left(\rho^2\right) - \nabla g_\theta\left(\rho^1\right)\right)^T\left(\rho^2 - \rho^1\right) \\ &= \frac{1}{2}\left(\rho^2 - \rho^1\right)^T\nabla^2 g_\theta\left(\mu^1\right)\left(\rho^2 - \rho^1\right) + \frac{1}{2}\left(\rho^1 - \rho^2\right)^T\nabla^2 g_\theta\left(\mu^2\right)\left(\rho^1 - \rho^2\right) \\ &\leq -q_f\left(\left\|\lambda^0 - \lambda^*\right\|\right)\sigma_{\min}\left(A^T\right)^2\left\|\rho^2 - \rho^1\right\|^2.\end{aligned}$$

This establishes the inequality in (4.16) and completes the proof of the lemma. □

Corollary 4.3. *For any* $\lambda \in B\left(\lambda^*, \left\|\lambda^0 - \lambda^*\right\|\right)$,

$$\|\nabla g_\theta(\lambda)\| \leq Q \|\lambda - \lambda^*\|$$

and

$$\nabla g_\theta(\lambda)^T (\lambda - \lambda^*) \leq -q \|\lambda - \lambda^*\|^2 .$$

Proof. This follows from an application of Lemma 4.2 with $\rho^1 = \lambda^*$ and $\rho^2 = \lambda$, using the additional observations that $\nabla g_\theta(\lambda) = Ax(\lambda) - b = Ax(\lambda) - Ax(\lambda^*)$, and $\nabla g_\theta(\lambda^*) = 0$ because λ^* is an optimal solution to (D_θ). □

We now show that the dual vector λ^k at the end of any iteration k in which the stopping conditions are not satisfied is as close to the optimal solution λ^* as the vector λ^{k-1} at the start of the iteration. While this guarantee is too weak to imply directly a convergence analysis, it is necessary to justify our use of the fixed parameters q and Q throughout the entire course of the algorithm.

Lemma 4.4. *For each iteration* k *executed by an inner run in which the stopping conditions are not satisfied, if* $\lambda^{k-1} \in B\left(\lambda^*, \left\|\lambda^0 - \lambda^*\right\|\right)$, *then* $\|\lambda^k - \lambda^*\| \leq \|\lambda^{k-1} - \lambda^*\|$.

Proof. Suppose the stopping conditions are not satisfied in an iteration k, and so $\lambda^k - \lambda^{k-1} = t\Delta\lambda^{k-1}$. The square of the distance from λ^k to λ^* can be expressed as

$$\begin{aligned}
\left\|\lambda^k - \lambda^*\right\|^2 &= \left\|\left(\lambda^k - \lambda^{k-1}\right) + \left(\lambda^{k-1} - \lambda^*\right)\right\|^2 \\
&= \left\|\lambda^{k-1} - \lambda^*\right\|^2 + \left\|\lambda^k - \lambda^{k-1}\right\|^2 + 2\left(\lambda^k - \lambda^{k-1}\right)^T \left(\lambda^{k-1} - \lambda^*\right) \\
&= \left\|\lambda^{k-1} - \lambda^*\right\|^2 + t^2 \left\|\Delta\lambda^{k-1}\right\|^2 + 2t\left(\Delta\lambda^{k-1}\right)^T \left(\lambda^{k-1} - \lambda^*\right). \qquad (4.18)
\end{aligned}$$

The third term in the right-hand side of (4.18) can be bounded from above by applying

Corollary 4.3, the Cauchy-Schwarz inequality, and Lemma 4.1 to obtain

$$\begin{aligned}\left(\Delta\lambda^{k-1}\right)^T\left(\lambda^{k-1}-\lambda^*\right) &= \left(\nabla g_\theta\left(\lambda^{k-1}\right)+u^{k-1}\right)^T\left(\lambda^{k-1}-\lambda^*\right)\\ &\leq -q\left\|\lambda^{k-1}-\lambda^*\right\|^2+\left\|u^{k-1}\right\|\left\|\lambda^{k-1}-\lambda^*\right\|\\ &\leq \left\|\lambda^{k-1}-\lambda^*\right\|^2\left(\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)Q-q\right).\end{aligned}$$

Substituting this inequality into (4.18), and again applying Lemma 4.1 and Corollary 4.3 yields

$$\left\|\lambda^k-\lambda^*\right\|^2 \leq \left\|\lambda^{k-1}-\lambda^*\right\|^2\left(1+t^2\left(1+\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)^2Q^2+2t\left(\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)Q-q\right)\right).$$

As $\alpha Q = q/6$, we will have $\left\|\lambda^k-\lambda^*\right\| \leq \left\|\lambda^{k-1}-\lambda^*\right\|$ provided that

$$t \leq \frac{\left(2-\frac{1}{3}\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)q}{\left(1+\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)^2Q^2}.$$

The step size in (4.9) used by an inner run satisfies this inequality because $\epsilon \leq 1$. This completes the proof of the lemma. □

To establish that an inner run makes progress as it executes iterations, we show that the norm of the gradient of $g_\theta\left(\lambda^k\right)$, $p\left(\lambda^k\right) = \left\|Ax\left(\lambda^k\right)-b\right\|$, decreases by a multiplicative factor in each iteration.

Lemma 4.5. *For each iteration k executed by an inner run in which the stopping conditions are not satisfied,*

$$\left\|\nabla g_\theta\left(\lambda^k\right)\right\| \leq \left(\sqrt{1-\frac{1}{4R^2}}\right)\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\|.$$

Proof. If the stopping conditions are not satisfied in iteration k, then Lemma 4.4 implies that $\lambda^{k-1}, \lambda^k \in B\left(\lambda^*, \left\|\lambda^0-\lambda^*\right\|\right)$. The squared norm of the gradient of g_θ at λ^k can be

expressed as

$$\begin{aligned}\left\|\nabla g_\theta\left(\lambda^k\right)\right\|^2 &= \left\|\left(\nabla g_\theta\left(\lambda^k\right)-\nabla g_\theta\left(\lambda^{k-1}\right)\right)+\nabla g_\theta\left(\lambda^{k-1}\right)\right\|^2 \\ &= \left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\|^2+\left\|\nabla g_\theta\left(\lambda^k\right)-\nabla g_\theta\left(\lambda^{k-1}\right)\right\|^2 \\ &\quad +2\left(\nabla g_\theta\left(\lambda^k\right)-\nabla g_\theta\left(\lambda^{k-1}\right)\right)^T\nabla g_\theta\left(\lambda^{k-1}\right). \end{aligned} \tag{4.19}$$

By Lemmas 4.2 and 4.1, the second term in the right-hand side of (4.19) can be bounded from above by

$$\begin{aligned}\left\|\nabla g_\theta\left(\lambda^k\right)-\nabla g_\theta\left(\lambda^{k-1}\right)\right\| &= \left\|\left(Ax\left(\lambda^k\right)-b\right)-\left(Ax\left(\lambda^{k-1}\right)-b\right)\right\| \\ &= \left\|Ax\left(\lambda^k\right)-Ax\left(\lambda^{k-1}\right)\right\| \\ &\le Q\left\|\lambda^k-\lambda^{k-1}\right\| \\ &= tQ\left\|\Delta\lambda^{k-1}\right\| \\ &\le t\left(1+\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)Q\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\|. \end{aligned} \tag{4.20}$$

To bound the third term in the right-hand side of (4.19), we again apply Lemmas 4.2 and 4.1 to obtain

$$\begin{aligned}&\left(\nabla g_\theta\left(\lambda^k\right)-\nabla g_\theta\left(\lambda^{k-1}\right)\right)^T\nabla g_\theta\left(\lambda^{k-1}\right) \\ &= \left(\nabla g_\theta\left(\lambda^k\right)-\nabla g_\theta\left(\lambda^{k-1}\right)\right)^T\left(\Delta\lambda^{k-1}-u^{k-1}\right) \\ &\le -tq\left\|\Delta\lambda^{k-1}\right\|^2+\left\|u^{k-1}\right\|\left\|\nabla g_\theta\left(\lambda^k\right)-\nabla g_\theta\left(\lambda^{k-1}\right)\right\| \\ &\le -t\left(1-\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)^2 q\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\|^2 \\ &\quad +t\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\left(1+\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)Q\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\|^2. \end{aligned} \tag{4.21}$$

Substituting (4.20) and (4.21) into (4.19) yields

$$\begin{aligned}&\left\|\nabla g_\theta\left(\lambda^k\right)\right\|^2\\&\leq\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\|^2\left(1+t^2\left(1+\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)^2Q^2\right.\\&\qquad\left.+2t\left(\frac{1}{6}\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\left(1+\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)-\left(1-\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)^2\right)q\right),\end{aligned}$$

where we have used the fact that $\alpha Q = q/6$. For the step size t in (4.9), we have

$$\begin{aligned}&\left\|\nabla g_\theta\left(\lambda^k\right)\right\|^2\\&\leq\left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\|^2\left(1-\left(\frac{\left(\left(1-\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)^2-\frac{1}{6}\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\left(1+\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)\right)q}{\left(1+\alpha\left(\frac{1}{2}+\frac{\epsilon}{3}\right)\right)Q}\right)^2\right).\end{aligned}$$

Since $\alpha \leq 1/6$ and $\epsilon \leq 1$, it follows that

$$\left\|\nabla g_\theta\left(\lambda^k\right)\right\|^2 \leq \left\|\nabla g_\theta\left(\lambda^{k-1}\right)\right\|^2\left(1-\left(\frac{q}{2Q}\right)^2\right),$$

and the proof is complete. □

Lemma 4.5 implies an upper bound on the number of iterations executed by an inner run.

Theorem 4.6. *An inner run terminates after*

$$O\left(R^2\log\left(\frac{p\left(\lambda^0\right)}{\epsilon\|b\|}\right)\right)$$

iterations with a solution $x(\lambda)$ such that $\|Ax\left(\lambda\right)-b\| \leq \epsilon\|b\|$.

Proof. If an inner run terminates with a solution $x(\lambda)$, then the stopping conditions (4.7) and (4.8) are both satisfied for the estimate $\hat{s} = s + u$ of the vector $s = Ax(\lambda)$. Applying

(4.6) and the triangle inequality yields

$$\begin{aligned}\|Ax(\lambda) - b\| &= \|s - b\| \\ &\leq \|\hat{s} - b\| + \|u\| \\ &\leq \|\hat{s} - b\| + \epsilon_1 \|s\| \\ &\leq \left(\frac{2}{3}\epsilon + \epsilon_1 \left(\frac{1+\epsilon_1}{1-\epsilon_1}\right)\left(1 + \frac{2}{3}\epsilon\right)\right)\|b\| + \epsilon_1 \left(\frac{1+\epsilon_1}{1-\epsilon_1}\right)\left(1 + \frac{2}{3}\epsilon\right)\|b\| \\ &= \left(\frac{2}{3}\epsilon + \frac{2\epsilon\alpha}{3}\left(\frac{1+\epsilon_1}{1-\epsilon_1}\right)\left(1 + \frac{2}{3}\epsilon\right)\right)\|b\|.\end{aligned}$$

Because $\epsilon \leq 1$ and $\alpha \leq 1/6$, $\epsilon_1 = \epsilon\alpha/3 \leq 1/18$, and so we obtain $\|Ax(\lambda) - b\| \leq \epsilon\|b\|$.

Now, we show that if $\left\|s^{k-1} - b\right\| \leq (2/3)\epsilon\|b\|$ at the start of an iteration k, then the inner run will terminate in that iteration. Since $\left|\left\|s^{k-1}\right\| - \|b\|\right| \leq \left\|s^{k-1} - b\right\|$, (4.6) implies that

$$(1 - \epsilon_1)\left(1 - \frac{2}{3}\epsilon\right)\|b\| \leq \left\|\hat{s}^{k-1}\right\| \leq (1 + \epsilon_1)\left(1 + \frac{2}{3}\epsilon\right)\|b\|,$$

and (4.7) is satisfied. Moreover,

$$\begin{aligned}\left\|\hat{s}^{k-1} - b\right\| &\leq \left\|s^{k-1} - b\right\| + \left\|u^{k-1}\right\| \\ &\leq \frac{2}{3}\epsilon\|b\| + \epsilon_1\left\|s^{k-1}\right\| \\ &\leq \left(\frac{2}{3}\epsilon + \epsilon_1\left(1 + \frac{2}{3}\epsilon\right)\right)\|b\| \\ &\leq \left(\frac{2}{3}\epsilon + \epsilon_1\left(\frac{1+\epsilon_1}{1-\epsilon_1}\right)\left(1 + \frac{2}{3}\epsilon\right)\right)\|b\|,\end{aligned}$$

and so (4.8) is satisfied as well, and the inner run terminates.

Repeated application of Lemma 4.5 implies that, if an inner run does not terminate in or before an iteration k, then

$$\left\|\nabla g_\theta\left(\lambda^k\right)\right\| \leq \left(1 - \frac{1}{4R^2}\right)^{\frac{k}{2}} p\left(\lambda^0\right).$$

For

$$k \geq 8R^2 \ln\left(\frac{3p\left(\lambda^0\right)}{2\epsilon\|b\|}\right),$$

we have $\left\|\nabla g_\theta\left(\lambda^k\right)\right\| \leq (2/3)\epsilon\|b\|$, and hence the stopping conditions will be satisfied and an inner run will terminate in the claimed number of iterations. □

Finally, we bound the difference between the objective function value of the solution produced by an inner run and the optimal value of the primal problem.

Corollary 4.7. *The objective function value of the solution $x(\lambda)$ produced by an inner run satisfies*

$$f(x(\lambda)) \leq \mathrm{OPT} + \epsilon\|b\|\|\lambda\| + n\theta.$$

Proof. Given the solution $x(\lambda)$ produced by an inner run, define a vector $\nu(\lambda) \in \mathbf{R}^n_{++}$ by, for all $i = 1, \ldots, n$,

$$\nu_i(\lambda) = \frac{\theta}{x_i(\lambda)}.$$

The pair $(\lambda, \nu(\lambda))$ is a feasible solution to the dual problem (D) with objective function value

$$\begin{aligned} g(\lambda, \nu(\lambda)) &= \inf_{x \in \mathbf{R}^n_+} L(x, \lambda, \nu(\lambda)) \\ &= -b^T\lambda + \sum_{i=1}^n \inf_{x_i \in \mathbf{R}_+} \left(f_i(x_i) + \left(a_i^T\lambda - \frac{\theta}{x_i(\lambda)} \right) x_i \right). \end{aligned}$$

As the components of the vector $x(\lambda)$ satisfy (4.2), we have $L(x(\lambda), \lambda, \nu(\lambda)) = g(\lambda, \nu(\lambda))$.

From the definition of the Lagrangian and the fact that $(\lambda, \nu(\lambda))$ is feasible for (D),

$$\begin{aligned} f(x(\lambda)) + \lambda^T(Ax(\lambda) - b) - \nu(\lambda)^T x(\lambda) &= L(x(\lambda), \lambda, \nu(\lambda)) \\ &= g(\lambda, \nu(\lambda)) \\ &\leq \mathrm{OPT}. \end{aligned}$$

Applying the Cauchy-Schwarz inequality and Theorem 4.6 yields

$$\begin{aligned} f(x(\lambda)) &\leq \text{OPT} - \lambda^T(Ax(\lambda) - b) + \nu(\lambda)^T x(\lambda) \\ &\leq \text{OPT} + \|\lambda\| \, \|Ax(\lambda) - b\| + \sum_{i=1}^{n} \left(\frac{\theta}{x_i(\lambda)} \right) x_i(\lambda) \\ &\leq \text{OPT} + \epsilon \|b\| \|\lambda\| + n\theta, \end{aligned}$$

which is the claimed upper bound on the objective function value of the vector $x(\lambda)$. □

Since the dual solution λ produced by the algorithm satisfies $\|\lambda\| \leq \|\lambda^0\| + 2\|\lambda^0 - \lambda^*\|$, by choosing the parameters ϵ and θ appropriately, the approximation error can be made as small as desired (though, of course, the convergence time increases as each of these parameters decreases).

4.4 Setting Parameters

In this section, we consider the setting of some parameters that were assumed known by an inner run in Section 4.3. First, we describe the outer loop of the algorithm. The purpose of the outer loop is to invoke inner runs with various parameter values, and to terminate runs if they do not end in the allotted number of iterations.

As the outer loop does not know the values q and Q, it uses binary search to choose the parameter values for the inner runs. Note that the analysis in Section 4.3 remains valid if we replace the former quantity with a lower bound on it, and the latter quantity with an upper bound on it. Let $U > 0$ be an upper bound on the ratio between the largest and smallest possible values of these two quantities.

The outer loop enumerates $\log U$ possible values $q_1, q_2, \ldots, q_{\log U}$ for q, with $q_{\ell+1} = 2q_\ell$ for each ℓ. Similarly, it considers values $Q_1, Q_2, \ldots, Q_{\log U}$ for Q. For each pair of values (q_{ℓ_1}, Q_{ℓ_2}) such that $q_{\ell_1} \leq Q_{\ell_2}$, it computes an upper bound $T(q_{\ell_1}, Q_{\ell_2})$ on the number of iterations required for an inner run with these parameter values, using Theorem 4.6.

Now, the outer loop sorts the $T(q_{\ell_1}, Q_{\ell_2})$ values, and executes inner runs according to this sorted order. When an inner run is executed with parameter values (q_{ℓ_1}, Q_{ℓ_2}), the outer loop lets it execute for $T(q_{\ell_1}, Q_{\ell_2})$ iterations. If it terminates due to the stopping

conditions being satisfied within this number of iterations, then by Theorem 4.6 the solution $x(\lambda)$ produced satisfies $\|Ax(\lambda) - b\| \leq \epsilon\|b\|$, and so the outer loop outputs this solution. On the other hand, if the stopping conditions for the inner run are not satisfied within the allotted number of iterations, the outer loop terminates the inner run, and then executes the next inner run with new parameter values according to the order induced by $T(q_{\ell_1}, Q_{\ell_2})$.

By the choice of q_{ℓ_1} and Q_{ℓ_2}, there exist $q_{\ell_1^*}$ and $Q_{\ell_2^*}$ such that $q/2 \leq q_{\ell_1^*} \leq q$ and $Q \leq Q_{\ell_2^*} \leq 2Q$. For the parameter pair $(q_{\ell_1^*}, Q_{\ell_2^*})$, $T(q_{\ell_1^*}, Q_{\ell_2^*})$ is, up to constant factors, the bound in Theorem 4.6. Hence, when the outer loop reaches the pair $(q_{\ell_1^*}, Q_{\ell_2^*})$, the corresponding inner run will terminate with the stopping conditions satisfied in the number of iterations specified in Theorem 4.6. Since the inner runs executed prior to this one will also be terminated in at most this number of iterations, and there are at most $\log^2 U$ such runs, we obtain the following upper bound on the total number of iterations executed by the algorithm.

Lemma 4.8. *The total number of iterations executed in all the inner runs initiated by the outer loop is*

$$O\left(R^2 \log\left(\frac{p\left(\lambda^0\right)}{\epsilon\|b\|}\right) \log^2 U\right).$$

In an iteration k of an inner run, the nodes must compute an estimate $\hat{s}_j^{k-1}$ for each of the m components of the vector s^{k-1}. As such, the summation routine must be invoked m times in each iteration. When the error probability δ satisfies $\delta \leq 1/n$, the summation algorithm in Chapter 3 computes an estimate satisfying (4.6) with probability at least $1 - \delta$ in $O(\epsilon_1^{-2} \log^2 \delta^{-1}/\Phi_P)$ time, where Φ_P is the conductance of the doubly stochastic communication matrix P that determines how nodes communicate with each other. We assume here that the nodes have an upper bound N on the number of nodes in the network, and a lower bound ϕ on Φ_P. Using these bounds, the nodes can terminate the summation algorithm in $O(\epsilon_1^{-2} \log^2 \delta^{-1}/\phi)$ time with an estimate $\hat{s}_j^{k-1}$ such that the probability that (4.6) is not satisfied is at most δ.

Given Lemma 4.8 and the fact that there are m summation computations per iteration, to ensure that every summation computation satisfies (4.6) with high probability, it suffices

to set

$$\delta \leq \left(N^2 m R^2 \log \left(\frac{p\left(\lambda^0\right)}{\epsilon \|b\|} \right) \log^2 U \right)^{-1}.$$

By setting δ within a constant factor of this upper bound, and using the fact that $\epsilon_1 = \epsilon\alpha/3 = \epsilon/(18R)$, we conclude that one invocation of the summation subroutine will run in

$$O\left(\frac{R^2}{\epsilon^2 \phi} \left(\log \left(NmR \log \left(\frac{p\left(\lambda^0\right)}{\epsilon \|b\|} \right) \log U \right) \right)^2 \right)$$

time. Combining this with Lemma 4.8 yields the following upper bound on the total running time of the algorithm.

Theorem 4.9. *The algorithm produces a solution $x(\lambda)$ that satisfies $\|Ax(\lambda) - b\| \leq \epsilon\|b\|$ and the objective function value bound in Corollary 4.7 with high probability in a total running time of*

$$O\left(\frac{mR^4}{\epsilon^2 \phi} \log \left(\frac{p\left(\lambda^0\right)}{\epsilon \|b\|} \right) \log^2 U \left(\log \left(NmR \log \left(\frac{p\left(\lambda^0\right)}{\epsilon \|b\|} \right) \log U \right) \right)^2 \right).$$

4.5 Extension to Linear Inequalities

The algorithm we have presented for the primal problem (P) can be adapted to handle problems with linear inequalities of the form $Ax \leq b$. Our first step is to rewrite the inequalities as equalities by introducing slack variables z_j for the constraints. The slack variables are constrained to be non-negative, and so the Lagrange dual function now has an additional vector of dual variables corresponding to these constraints on the slack variables. Also, the infimum in the definition of the Lagrange dual function is now taken over the slack variables z as well as the primal variables x. To ensure that the infimum is finite, we introduce an additional term $\psi(z) = \sum_{j=1}^{m} \psi_j(z_j)$ into the objective function for the slack variables. The functions ψ_j must satisfy the same technical conditions as the functions f_i. An example of a natural choice of $\psi_j(z_j)$ is a quadratic function in z_j.

We introduce logarithmic barriers for the non-negativity constraints on both the primal

and the slack variables to obtain the primal barrier problem

$$\begin{aligned} \text{minimize} \quad & f(x) - \theta \sum_{i=1}^{n} \ln x_i + \psi(z) - \theta \sum_{j=1}^{m} \ln z_j \\ \text{subject to} \quad & Ax + z = b. \end{aligned}$$

The corresponding Lagrange dual function g_θ has the gradient

$$\nabla g_\theta(\lambda) = Ax(\lambda) + z(\lambda) - b,$$

where $z(\lambda)$ is the vector defined by

$$z_j(\lambda) = \arg \inf_{z_j \in \mathbf{R}_{++}} (\psi_j(z_j) - \theta \ln z_j + \lambda_j z_j)$$

for all $j = 1, \ldots, m$.

In the basic algorithm, the nodes now perform gradient ascent for the new dual barrier problem. Computation of the gradient in each iteration requires the nodes to compute the vector $z\left(\lambda^{k-1}\right)$ for the current dual vector λ^{k-1}. This can be accomplished by each node if it is given the functions ψ_j for all j. The stopping conditions that the nodes check to determine whether the approximate feasibility guarantee is satisfied are modified to account for the additional term $z(\lambda)$ in the gradient.

When an inner run terminates with a solution $x(\lambda)$ such that $\|Ax(\lambda) + z(\lambda) - b\| \leq \epsilon\|b\|$, the corresponding approximate feasibility bound for the inequality constraints is $\|v(x(\lambda))\| \leq \epsilon\|b\|$, where the vector v is defined by $v_j(x(\lambda)) = \max((Ax(\lambda))_j - b_j, 0)$ for all $j = 1, \ldots, m$. The addition of the $\psi(z)$ term to the primal objective function leads to further error in the approximate guarantee on the value of $f(x(\lambda))$ relative to OPT, although this error can be made small by appropriate choice of the functions ψ_j. Choosing the functions ψ_j to minimize this error will affect the running time of the algorithm, however, because the definitions of the curvature variation parameters q and Q now have additional terms that account for variation in the curvature of the ψ_j functions.

Chapter 5

Price of Anarchy in Cost Sharing

In this chapter, we study a different model of a distributed system than the ones we have previously considered. So far, we have assumed that we can program the nodes before they are deployed, and each node will execute its program. The challenge is to develop an algorithm that, if executed at each node, will cause the global system to converge to a desired state. Nodes have limited ability to communicate directly with each other.

We now consider a system of independent agents over which we do not have direct control. Since we cannot dictate the behavior of the agents, we must account for different possible actions by them. We assume that the agents are rational in that they act in their own best interests. Instead of specifying the behavior of the agents, the goal of the system designer is to set up the system so that, when the agents behave selfishly, the outcome is not much worse than the best outcome possible if the designer could dictate the agents' behavior.

The setting that we study involves a good that can be produced in any amount. The cost of producing some amount of the good is a convex function of the amount produced. This convexity property corresponds to an assumption that the marginal cost of production increases as the amount produced increases. There is a set of agents, or users, each of whom can request any quantity of the good. All of the requests are always satisfied, so that a user always receives an amount of the good that is equal to the requested amount, and the total amount produced is the sum of the requests of the users. A canonical example is when requested quantities correspond to traffic rates or amounts in a network, and production cost corresponds to aggregate delay.

Producing the total amount of the good incurs some cost, which is returned to the users as follows. A cost sharing method is a procedure that takes as inputs the cost function, the number of users, and the requested quantities of all the users, and assigns a cost share to each user. The cost share of a user is an amount the user must pay to the system to cover the production cost. We consider cost sharing methods that are budget-balanced in the sense that the sum of the cost shares is always equal to the cost of producing the total quantity requested. When the cost sharing method is budget-balanced, the system always recovers the total cost of production from the users in aggregate. In our canonical example, every queue service discipline (such as FIFO) used on communication links in the network yields a corresponding cost-sharing method.

We assume that the welfare of an individual user is quasi-linear in that it is equal to the difference between the utility the user derives from the requested quantity of the good and the cost share assigned to the user. The aggregate surplus of the system is the sum of the welfares of all the users. From the point of view of the system as a whole, a natural goal is to maximize the aggregate surplus. We call the set of requested quantities of the users that maximizes the aggregate surplus the centralized optimal solution, as this is the state of the system that would be chosen by a central authority that could dictate the behavior of all the users.

Rational users will choose quantities of the good to request to maximize their own welfares. Since the cost share of a particular user depends on the requested quantities of the other users as well as the user's quantity, this strategic behavior gives rise to a game among the users. To evaluate the outcome of the game, we use the concept of Nash equilibrium from game theory. The set of Nash equilibria of the game depends on the cost sharing method used to assign cost shares. We define the price of anarchy of a cost sharing method to be the worst-case ratio between the aggregate surplus of the system under a Nash equilibrium and the aggregate surplus of the centralized optimal solution.

This general framework has a number of applications, and as such has been studied in various instantiations. The paper by Moulin [35] has references to other work that fits into the framework.

The problem of the system designer who cannot dictate the behavior of the users is to design the cost sharing method to achieve a desirable price of anarchy. While the task of creating a cost sharing method that optimizes the price of anarchy in some way is complicated, a first step in this direction is to determine the price of anarchy of a

known cost sharing method. The two known cost sharing methods that have been studied the most are average cost pricing and serial cost sharing. In a queueing context, these correspond to the well-known FIFO and Fair Share service discplines. We develop here an analysis approach that allows us to determine the price of anarchy of each of these two cost sharing methods under quadratic cost functions. Our analysis also determines the price of anarchy of each cost sharing method in a class that interpolates between average cost pricing and serial cost sharing.

5.1 Model and Overview

We consider a cost function $C : \mathbf{R}_+ \to \mathbf{R}_+$ that, for any $y \geq 0$, specifies the cost of producing y units of the good. We assume that C is non-decreasing and convex, and $C(0) = 0$.

Suppose that there are n users, identified by the integers in the set $\{1, \ldots, n\}$, which we denote by $[n]$. Let $x \in \mathbf{R}_+^n$ be a vector whose ith component x_i is the amount of the good requested by user i. Then the total amount of the good requested is $\sum_{i=1}^n x_i$, and the cost of producing that amount is $C\left(\sum_{i=1}^n x_i\right)$.

Definition 5.1. For a given cost function C and number of users n, a *cost sharing method* is a mapping $\xi : \mathbf{R}_+^n \to \mathbf{R}_+^n$ that satisfies the following conditions:

- Budget balance: $\forall x \in \mathbf{R}_+^n$, $\sum_{i=1}^n \xi_i(x) = C\left(\sum_{i=1}^n x_i\right)$;
- Symmetry: ξ is symmetric in all variables;
- Subset coverage: For all $x \in \mathbf{R}_+^n$ and any set $S \subseteq [n]$, $\sum_{i \in S} \xi_i(x) \geq C\left(\sum_{i \in S} x_i\right)$.

The symmetry condition is motivated by the assumption that the only information about a user that the cost sharing method can use to determine the cost shares is the quantity requested by the user. The subset-coverage condition states that a group of users cannot benefit by being assigned smaller cost shares in aggregate when other users are added to the system. It prevents the cost sharing method from overly penalizing a group of users by passing on some of the cost due to their requests to other users.

The following two cost sharing methods are well known and have been studied previously.

Example 5.1 (Average cost pricing). For any cost function C, number of users n, vector $x \in \mathbf{R}^n_+$ such that $\sum_{j=1}^n x_j > 0$, and user $i \in [n]$,

$$\xi_i(x) = \left(\frac{x_i}{\sum_{j=1}^n x_j} \right) C \left(\sum_{j=1}^n x_j \right).$$

If $x_j = 0$ for all $j \in [n]$, then $\xi_i(x) = 0$ for all users $i \in [n]$.

In our canonical queueing example, average cost pricing corresponds to the FIFO service discipline.

While average cost pricing assigns cost shares to users in proportion to their requested quantities, the second cost sharing method attempts to penalize users that request large quantities by assigning them relatively larger cost shares, and insulate users that request smaller quantities from the large requests. We use the following general notation to specify the cost shares. For a vector $x \in \mathbf{R}^n$, we say that a permutation $\pi : [n] \to [n]$ is an *ordering* of x if the vector $z \in \mathbf{R}^n$ such that $z_{\pi(i)} = x_i$ for all $i \in [n]$ satisfies $z_1 \leq z_2 \leq \cdots \leq z_n$. The vector z is said to be the *ordered version* of x. Note that there are multiple different orderings of a vector x when the elements of x are not all distinct, but all the orderings give rise to the same ordered version z.

Example 5.2 (Serial cost sharing [36]). For any cost function C, number of users n, and vector $x \in \mathbf{R}^n_+$, let π be any ordering of x, and let z be the ordered version of x. For $k \in [n]$, let $s_k = \sum_{\ell=1}^{k-1} z_\ell + (n-k+1)z_k$. Then the cost share of user $i \in [n]$ is

$$\xi_i(x) = \frac{C(s_{\pi(i)})}{n - \pi(i) + 1} - \sum_{k=1}^{\pi(i)-1} \frac{C(s_k)}{(n-k+1)(n-k)}.$$

These cost shares arise from the following process, which consists of n phases. Under the ordering π, we have $x_{\pi^{-1}(1)} \leq \cdots \leq x_{\pi^{-1}(n)}$. In the first phase, the users $\pi^{-1}(2), \ldots, \pi^{-1}(n)$ are treated as if they all requested the quantity $x_{\pi^{-1}(1)}$ and the total requested quantity is s_1. The cost $C(s_1)$ is divided equally among all the users, and the user $\pi^{-1}(1)$ requesting the smallest quantity is removed from future phases. In a later phase $k = 2, \ldots, n$, the incremental cost $C(s_k) - C(s_{k-1})$ is divided equally among the active users $\pi^{-1}(k), \ldots, \pi^{-1}(n)$. The final cost share of each user is the sum of the costs assigned to the user in all the phases.

We also consider the following class of cost sharing methods, which interpolates between average cost pricing and serial cost sharing.

Example 5.3. For any cost function C, number of users n, vector $x \in \mathbf{R}_+^n$, and user $i \in [n]$, let $\xi_i^{\text{AVG}}(x)$ and ξ_i^{SER} denote the cost shares of user i under average cost pricing and serial cost sharing, respectively. Then, for a parameter $\theta \in [0, 1]$, the cost share of user i is

$$\xi_i(x) = \theta \xi_i^{\text{SER}}(x) + (1 - \theta)\xi_i^{\text{AVG}}(x). \tag{5.1}$$

We refer to the cost sharing method in Example 5.3 corresponding to the parameter value θ as the *θ-combination*. Note that average cost pricing and serial cost sharing are the special cases of the θ-combination for $\theta = 0$ and $\theta = 1$, respectively.

The level of satisfaction of a user is a function of the amount of the good the user receives and the cost share assigned to the user. We assume that, for each user $i \in [n]$, there is a utility function $U_i : \mathbf{R}_+ \to \mathbf{R}_+$ that specifies the amount $U_i(x_i)$ of utility user i derives from a quantity x_i of the good. The function U_i is assumed to be a member of the set $\mathcal{U}$ of non-decreasing, continuous, and concave functions U such that $U(0) = 0$ and the right directional derivative $U'(0)$ exists and is finite. Fix a cost function C and a cost sharing method ξ. We assume that the payoff of user i is

$$P_i(x) = U_i(x_i) - \xi_i(x). \tag{5.2}$$

Payoff functions of this separable form are said to be *quasi-linear*.

In a distributed system such as a shared network in which there is no central authority that dictates the behavior of all agents, one typically assumes that the users act to maximize their own payoff functions. This assumption gives rise to a game in which the strategic behavior of the users determines the outcome. In this game, which we refer to as the *production game*, the possible actions of a user i are the numbers in $\mathbf{R}_+$, which correspond to the possible quantities of the good that the user can request. When each user chooses a particular quantity x_i to request, the cost sharing method assigns a cost share $\xi_i(x)$ to user i, and user i receives the payoff $P_i(x) = U_i(x_i) - \xi_i(x)$.

A central tool in game theory for understanding strategic games is the concept of Nash equilibrium. To define a Nash equilibrium for the production game, for any vector $x \in \mathbf{R}_+^n$

and $i \in [n]$, we introduce the notation x_{-i} to denote the vector $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ containing the entries of x other than entry i. For a vector $x \in \mathbf{R}_+^n$, a scalar $y \in \mathbf{R}_+$, and any $i \in [n]$, we write $P_i(y; x_{-i})$ to denote the payoff $P_i(x')$ of the user i under the vector of quantities $x' = (x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$.

Definition 5.2. A *Nash equilibrium* of the production game is a vector $x^* \in \mathbf{R}_+^n$ such that, for each user $i \in [n]$,

$$\forall y \in \mathbf{R}_+,\ P_i(x_i^*; x_{-i}^*) \geq P_i(y; x_{-i}^*). \tag{5.3}$$

Intuitively, a Nash equilibrium in the production game is an outcome of the game such that no user can benefit through an increase in payoff by deviating unilaterally from the quantity of the good requested by that user in the equilibrium.

The set of Nash equilibria of the production game depends on the cost sharing method used to assign cost shares to the users. Under certain cost functions and cost sharing methods, the production game may have no Nash equilibria, exactly one Nash equilibrium, or multiple Nash equilibria. In general, a system designer that selects the cost sharing method would want the resulting game to have Nash equilibria, because a lack of equilibria would limit the designer's ability to predict the outcome of the game. Furthermore, the existence of multiple equilibria would present the possibility of several potential outcomes, again limiting the predictive ability of the designer. (Though this issue is less serious if all equilibria are "good.")

We consider the task of the system designer, which is the choice of the cost sharing method. To evaluate the outcome of the production game from the point of view of the system designer, we define the following measure.

Definition 5.3. For a vector $x \in \mathbf{R}_+^n$, the aggregate surplus of the system is the quantity

$$A(x) = \sum_{i=1}^{n} P_i(x).$$

Note that the quasi-linear form of the payoff functions and the budget-balance condition of a cost sharing method imply that the aggregate surplus can be expressed as

$$\begin{aligned} A(x) &= \sum_{i=1}^{n} (U_i(x_i) - \xi_i(x)) \\ &= \sum_{i=1}^{n} U_i(x_i) - C\left(\sum_{i=1}^{n} x_i\right). \end{aligned} \tag{5.4}$$

In particular, for a given vector x of requested quantities, the aggregate surplus does not depend on the cost sharing method used to assign cost shares to the users. The aggregate surplus does depend on the utility functions of the users, however. When the collection of utility functions under discussion is not apparent, we write $A_U(x)$ to denote the aggregate surplus of the system under a collection $U = (U_1, \ldots, U_n)$ of utility functions and a vector of quantities x.

Maximizing the aggregate surplus of the system is a reasonable goal for a system designer who is interested in the welfare of all the users. For a collection U of utility functions, let

$$\mathrm{OPT}_U = \max_{x \in \mathbf{R}_+^n} A_U(x)$$

be the maximum aggregate surplus under any vector of requested quantities. We consider any Nash equilibrium of the production game to be a potential expected outcome of the game. This perspective motivates the following performance measure for cost sharing methods.

Definition 5.4. Fix a cost function C, number of users n, and cost sharing method ξ. For any collection $U = (U_1, \ldots, U_n)$ of utility functions, where $U_i \in \mathcal{U}$ for each $i \in [n]$, let $\mathcal{E}(U)$ denote the set of Nash equilibria of the production game under C, n, ξ, and U. The price of anarchy of ξ is the quantity

$$\inf_{U \in \mathcal{U}^n} \frac{\inf_{x^* \in \mathcal{E}(U)} A_U(x^*)}{\mathrm{OPT}_U}. \tag{5.5}$$

The price of anarchy of a cost sharing method is the worst-case ratio (over collections of utility functions and Nash equilibria) between the aggregate surplus of the system under a Nash equilibrium and the maximum possible aggregate surplus under any vector of requested quantities. In this sense, we are comparing a cost sharing method under its

worst Nash equilibrium to the outcome that would be chosen by a system designer who seeks to maximize aggregate surplus and can dictate the actions of all the users. Thus, the price of anarchy reflects the loss in the efficiency of the system due to the independent and selfish behavior of the users.

We are interested in finding the cost sharing method for the production game with the best possible (largest) price of anarchy. This problem has several applications [35]. We discuss here an application studied by Shenker [46].

Example 5.4. Consider a network with a link that serves as a bottleneck to a collection of users who transmit data to other destinations in the network across the link. There is a queue on the link that stores packets buffered at the link. Packets arriving at the link from the users are placed in the queue, and departing packets are chosen from the queue. A service discipline specifies the order in which packets depart from the queue.

We can model this system in the production game framework as follows. The good is the transmission of data, and the action of a user is the amount of data the user transmits across the link. The cost of producing a quantity of the good is the delay experienced by the packets when a certain amount of data crosses the link. A service discipline partitions the delay among the users by determining the order in which the packets of the various users are sent across the link. The payoff of a user is the difference between the utility the user derives from the amount of data the user transmits and the delay experienced by the packets of the user.

In this setting, the first-in-first-out (FIFO) service discipline gives rise to average cost pricing, and the Fair Share service discipline leads to serial cost sharing. The Fair Share service discipline is designed so that users receive their fair shares of the service, regardless of the actions of the other users. It can be implemented using a round-robin scheduler that cycles through the users, sending one packet from each user that has a packet in the queue during each round.

For a parameter $\theta \in [0, 1]$, the θ-combination can be implemented in a system with two queues, one using the FIFO service discipline, and the other using the Fair Share service discipline. Each packet arriving at the link is placed into the Fair Share queue with probability θ, and into the FIFO queue with probability $1 - \theta$. Whenever a packet is to depart from the link, the Fair Share and FIFO queues are again chosen with probabilities θ and $1 - \theta$, respectively, and the next packet to be served in the chosen queue is selected

for departure.

Moulin studied the price of anarchy of various cost sharing methods for the production game [35]. As the problem of finding the cost sharing method with the best price of anarchy seems difficult, Moulin focuses on the price of anarchy of average cost pricing and that of serial cost sharing under various cost functions. Using ad hoc calculations, he determines the price of anarchy of each of these two cost sharing methods when the cost function is quadratic. We provide a unified analysis for determining the price of anarchy of any cost sharing method in the class of θ-combinations, which includes both average cost pricing and serial cost sharing as special cases, when the cost function is quadratic. We use this analysis to identify (for each n) the best cost sharing method in this class (using price of anarchy as the measure of quality).

5.1.1 Organization

In Section 5.2, we develop necessary conditions for a Nash equilibrium of the production game, and we show that, under any cost sharing method in Example 5.3 and a quadratic cost function, the production game has a unique Nash equilibrium. Section 5.3 presents our analysis for determining the price of anarchy of each cost sharing method in this class under a quadratic cost function. In Section 5.4, we explain how the analysis is limited in a sense to this class of cost sharing methods.

5.2 Equilibrium Conditions and Quadratic Cost Functions

To understand the conditions that must hold at a Nash equilibrium, we consider the optimization problem that a particular user $i \in [n]$ solves to choose an action. Extending our previous notation, for any vector $x \in \mathbf{R}_+^n$ and scalar $y \in \mathbf{R}_+$, we write $\xi_i(y; x_{-i})$ to denote the cost share $\xi_i(x')$ of user i under the cost sharing method ξ and the vector of requested quantities $x' = (x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$. Fix a vector $x \in \mathbf{R}_+^n$ of quantities. With the quantities of the other users fixed, the payoff of user i as a function of the quantity $y \in \mathbf{R}_+$ requested by i is

$$P_i(y; x_{-i}) = U_i(y) - \xi_i(y; x_{-i}). \tag{5.6}$$

Since U_i is a concave function, if the function $\xi_i(y; x_{-i})$ is convex in y for any fixed vector $x \in \mathbf{R}_+^n$, then $P_i(y; x_{-i})$ is concave in y for any fixed x. If, in addition, U_i is

differentiable and $\xi_i(y; x_{-i})$ is differentiable in y for any fixed x, then $P_i(y; x_{-i})$ is also differentiable in y for any fixed x. In this case, we write $P_i'(y; x_{-i})$ and $\xi_i'(y; x_{-i})$ to denote the partial derivatives $\frac{\partial P_i(y;x_{-i})}{\partial y}$ and $\frac{\partial \xi_i(y;x_{-i})}{\partial y}$, respectively. The necessary condition for user i in (5.3) for a Nash equilibrium vector $x^* \in \mathbf{R}^n_+$ then becomes

$$\begin{aligned} x_i^* > 0 &\Rightarrow P_i'(x_i^*; x_{-i}^*) = 0; \\ x_i^* = 0 &\Rightarrow P_i'(0; x_{-i}^*) \leq 0. \end{aligned}$$

From the equation characterizing $P_i(y; x_{-i})$ in (5.6), these conditions are equivalent to

$$\begin{aligned} x_i^* > 0 &\Rightarrow U_i'(x_i^*) = \xi_i'(x_i^*; x_{-i}^*); \\ x_i^* = 0 &\Rightarrow U_i'(0) \leq \xi_i'(0; x_{-i}^*). \end{aligned} \tag{5.7}$$

The form of these first-order conditions suggests that the defining characteristic of the cost sharing method in the Nash equilibria of the production game is the collection of partial derivatives

$$\xi_i'(x_i; x_{-i}) = \frac{\partial \xi_i(x_i; x_{-i})}{\partial x_i} = \frac{\partial \xi_i(x)}{\partial x_i}$$

for $i \in [n]$.

From now on, we assume that the cost function is $C(y) = y^2$. The analysis to follow is also valid for any cost function of the form $C(y) = c_2 y^2 + c_1 y$, where $c_2 > 0$ and $c_1 \geq 0$. This assumption of a quadratic cost function is restrictive, but we will be rewarded with an exact characterization of the price of anarchy of every θ-combination.

Under this quadratic cost function and average cost pricing, the cost share of any user $i \in [n]$ for a vector of requested quantities $x \in \mathbf{R}^n_+$ is

$$\xi_i(x) = x_i \left(\sum_{j=1}^{n} x_j \right). \tag{5.8}$$

In the case of serial cost sharing, let π be an ordering of x and z be the ordered version of x. For any $i \in [n]$, define the sets $L_i(x) = \{j \in [n] \mid x_j < x_i\}$ and $Q_i(x) = \{j \in [n] - \{i\} \mid$

$x_j = x_i\}$. The cost share of user i under the quadratic cost function is

$$\begin{aligned}\xi_i(x) &= (n - \pi(i) + 1)x_i^2 + 2x_i \sum_{j=1}^{\pi(i)-1} z_j - \sum_{j=1}^{\pi(i)-1} z_j^2 \\ &= nx_i^2 - \sum_{j \in L_i(x)} (x_i - x_j)^2. \end{aligned} \tag{5.9}$$

The θ-combinations assign cost shares that are convex combinations of the cost shares under average cost pricing and serial cost sharing. As such, these cost sharing methods inherit properties that are preserved by linear combinations and are satisfied by both average cost pricing and serial cost sharing. The following lemma presents some properties that both average cost pricing and serial cost sharing have.

Lemma 5.1. *For the cost function $C(y) = y^2$, any number of users n, and any user $i \in [n]$, let ξ_i denote the cost share of i under either average cost pricing or serial cost sharing.*

(i) *The cost share $\xi_i(x)$ is continuous in x.*

(ii) *For any vector $x \in \mathbf{R}_+^n$, the partial derivative $\xi_i'(x_i; x_{-i})$ exists.*

(iii) *The function $\xi_i'(x_i; x_{-i})$ is continuous in x and is defined by*

$$\xi_i'(x_i; x_{-i}) = 2x_i + \sum_{j \neq i} x_j$$

for average cost pricing, and by

$$\xi_i'(x_i; x_{-i}) = 2\left(nx_i - \sum_{j \in L_i(x)} (x_i - x_j)\right)$$

for serial cost sharing.

(iv) *For any fixed $x \in \mathbf{R}_+^n$, the function $\xi_i(y; x_{-i})$ is convex in y.*

Proof. The expression in (5.8) defining the cost share $\xi_i(x)$ of a user $i \in [n]$ under average cost pricing and a quadratic cost function is continuous in x and differentiable with respect

to x_i. The partial derivative $\xi_i'(x_i; x_{-i}) = \frac{\partial \xi_i(x)}{\partial x_i}$ is

$$\frac{\partial \xi_i(x)}{\partial x_i} = 2x_i + \sum_{j \neq i} x_j,$$

and so $\xi_i'(x_i; x_{-i})$ is continuous in x.

We now consider the expression in (5.9) for the cost share $\xi_i(x)$ of a user i under serial cost sharing and a quadratic cost function. For any $j \neq i$, consider the function $f_j(x) = 1_{\{x_j < x_i\}}(x_i - x_j)^2$, where $1_{\{x_j < x_i\}}$ is an indicator function that, for any vector x, is one if $x_j < x_i$, and zero if $x_j \geq x_i$. Since the function f_j is continuous in x for each $j \neq i$, the cost share $\xi_i(x)$ is also continuous in x.

Suppose now that we hold the entries in x other than the one for user i fixed, and vary x_i. As we decrease x_i by an infinitesimal amount, $L_i(x)$ remains constant, so the left directional derivative of $\xi_i(x)$ with respect to x_i is

$$2\left(nx_i - \sum_{j \in L_i(x)} (x_i - x_j)\right). \tag{5.10}$$

On the other hand, as x_i increases by an infinitesimal amount, the elements in $Q_i(x)$ enter $L_i(x)$. Thus, the right directional derivative of $\xi_i(x)$ with respect to x_i is

$$2\left(nx_i - \sum_{j \in L_i(x) \cup Q_i(x)} (x_i - x_j)\right) = 2\left(nx_i - \sum_{j \in L_i(x)} (x_i - x_j)\right),$$

because $x_i = x_j$ for all $j \in Q_i(x)$. Since the left and right directional derivatives of $\xi_i(x)$ with respect to x_i are equal, the partial derivative $\frac{\partial \xi_i(x)}{\partial x_i}$ exists for serial cost sharing and is given in (5.10). An argument similar to the one for the continuity of $\xi_i(x)$ shows that $\xi_i'(x_i; x_{-i})$ is also continuous in x.

To show that $\xi_i(y; x_{-i})$ is convex in y for any $i \in [n]$ and fixed vector $x \in \mathbf{R}_+^n$, we consider how $\xi_i'(y; x_{-i})$ varies as a function of y. Under average cost pricing,

$$\xi_i'(y; x) = 2y + \sum_{j \neq i} x_j,$$

and so $\xi_i'(y; x_{-i})$ is increasing in y as x remains fixed.

For serial cost sharing, consider any pair of quantities y_1 and y_2 such that $0 \leq y_1 < y_2$. Define two vectors of quantities $x', \hat{x} \in \mathbf{R}^n_+$ as $x' = (x_1, \ldots, x_{i-1}, y_1, x_{i+1}, \ldots, x_n)$ and $\hat{x} = (x_1, \ldots, x_{i-1}, y_2, x_{i+1}, \ldots, x_n)$, and note that $L_i(x') \subseteq L_i(\hat{x})$. The difference $\xi_i'(y_2; x_{-i}) - \xi_i'(y_1; x_{-i})$ can be bounded from below by

$$\begin{aligned}
&\xi_i'(y_2; x_{-i}) - \xi_i'(y_1; x_{-i}) \\
&= 2\left(ny_2 - \sum_{j \in L_i(\hat{x})} (y_2 - \hat{x}_j) - ny_1 + \sum_{j \in L_i(x')} \left(y_1 - x'_j\right)\right) \\
&= 2\left(n(y_2 - y_1) - \sum_{j \in L_i(\hat{x}) - L_i(x')} (y_2 - \hat{x}_j) - \sum_{j \in L_i(x')} \left(y_2 - \hat{x}_j - y_1 + x'_j\right)\right) \\
&\geq 2\left(n(y_2 - y_1) - \left(|L_i(\hat{x})| - |L_i(x')|\right)(y_2 - y_1) - |L_i(x')|(y_2 - y_1)\right) \\
&= 2\left(n - |L_i(\hat{x})|\right)(y_2 - y_1) \\
&> 0,
\end{aligned}$$

where we have used the facts that $x'_j = \hat{x}_j$ for all $j \in L_i(x')$, $\hat{x}_j \geq y_1$ for all $j \in L_i(\hat{x}) - L_i(x')$, and $|L_i(\hat{x})| \leq n - 1$. We conclude that, as with average cost pricing, $\xi_i'(y; x_{-i})$ is increasing in y for any fixed x for serial cost sharing.

Hence, under both average cost pricing and serial cost sharing, for any $i \in [n]$ and fixed $x \in \mathbf{R}^n_+$, the function $\xi_i(y; x_{-i})$ is differentiable in y and the derivative with respect to y is increasing in y. These properties imply that $\xi_i(y; x_{-i})$ is convex in y for any fixed x. □

The θ-combinations inherit the properties of continuity, differentiability, and convexity in Lemma 5.1 from average cost pricing and serial cost sharing.

Corollary 5.2. *For the cost function $C(y) = y^2$, any number of users n, any user $i \in [n]$, and any $\theta \in [0, 1]$, let ξ_i denote the cost share of user i under the θ-combination.*

(i) *The cost share $\xi_i(x)$ is continuous in x.*

(ii) *For all $x \in \mathbf{R}^n_+$, the partial derivative $\xi_i'(x_i; x_{-i})$ exists.*

(iii) *The function $\xi_i'(x_i; x_{-i})$ is continuous in x.*

(iv) *For any fixed vector $x \in \mathbf{R}^n_+$, the function $\xi_i(y; x_{-i})$ is convex in y.*

Proof. From the definition in (5.1) of the cost share $\xi_i(x)$ of a user $i \in [n]$ under the θ-combination, we see that $\xi_i(x)$ is a convex combination of the cost shares under average cost pricing and serial cost sharing, $\xi_i^{\mathrm{AVG}}(x)$ and $\xi_i^{\mathrm{SER}}(x)$. By Lemma 5.1, both functions $\xi_i^{\mathrm{AVG}}(x)$ and $\xi_i^{\mathrm{SER}}(x)$ are continuous in x and differentiable with respect to x_i for any $x \in \mathbf{R}_+^n$, with these partial derivatives also continuous in x. As a result, for the θ-combination, $\xi_i(x)$ is continuous in x, the partial derivative $\xi_i'(x_i; x_{-i})$ exists, and $\xi_i'(x_i; x_{-i})$ is continuous in x. Lemma 5.1 also implies that, for any fixed vector $x \in \mathbf{R}_+^n$, the functions $\xi_i^{\mathrm{AVG}}(y; x_{-i})$ and $\xi_i^{\mathrm{SER}}(y; x_{-i})$ are convex in y. Since $\xi_i(y; x_{-i})$ under the θ-combination is a convex combination of these two functions, it is convex in y as well. □

Lemma 5.1 leads to the following description of the partial derivatives $\xi_i'(x_i; x_{-i})$ for the θ-combinations.

Corollary 5.3. *For the cost function $C(y) = y^2$, any number of users n, any user $i \in [n]$, and any $\theta \in [0, 1]$, let ξ_i denote the cost share of user i under the θ-combination. Define an $n \times n$ matrix B by*

$$B_{k\ell} = \begin{cases} 2(1 + \theta(n - k)), & \text{if } k = \ell; \\ 1 + \theta, & \text{if } k > \ell; \\ 1 - \theta, & \text{if } k < \ell \end{cases}$$

for any $k, \ell \in [n]$. For any vector $x \in \mathbf{R}_+^n$, let π be any ordering of x, and let z be the ordered version of x.

(i) *The vector $p \in \mathbf{R}^n$ with $p_{\pi(i)} = \xi_i'(x_i; x_{-i})$ for all $i \in [n]$ is given by*

$$p = Bz.$$

(ii) *For any $k_1, k_2 \in [n]$ such that $k_1 < k_2$, $p_{k_1} \leq p_{k_2}$, with strict inequality holding if $z_{k_1} < z_{k_2}$.*

Before proving this corollary, we provide examples in Figure 5.1 of the matrix B under average cost pricing and serial cost sharing.

Proof of Corollary 5.3. First, we consider average cost pricing. Define a vector $p^{\mathrm{AVG}} \in \mathbf{R}^n$ by $p_{\pi(i)}^{\mathrm{AVG}} = \xi_i'(x_i; x_{-i})$ for all $i \in [n]$, where ξ_i denotes the cost share of user i under average

$$B^0 = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \qquad B^1 = \begin{bmatrix} 8 & 0 & 0 & 0 \\ 2 & 6 & 0 & 0 \\ 2 & 2 & 4 & 0 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

Figure 5.1: The matrix B defined in Corollary 5.3 for average cost pricing (B^0) and serial cost sharing (B^1) when $n = 4$.

cost pricing. For any $k \in [n]$, let $i = \pi^{-1}(k)$. Since π is an ordering of x and z is the ordered version of x, $z_{\pi(j)} = x_j$ for all $j \in [n]$. Applying Lemma 5.1, we have

$$\begin{aligned} p_k^{\mathrm{AVG}} &= \xi_i'(x_i; x_{-i}) \\ &= 2x_i + \sum_{j \neq i} x_j \\ &= 2z_k + \sum_{\ell \neq k} z_k. \end{aligned}$$

Thus, $p^{\mathrm{AVG}} = B^{\mathrm{AVG}} z$, where B^{AVG} is an $n \times n$ matrix defined by

$$B_{k\ell}^{\mathrm{AVG}} = \begin{cases} 2, & \text{if } k = \ell; \\ 1, & \text{if } k \neq \ell \end{cases}$$

for any $k, \ell \in [n]$. For any $k_1, k_2 \in [n]$ such that $k_1 < k_2$, $p_{k_2}^{\mathrm{AVG}} - p_{k_1}^{\mathrm{AVG}} = z_{k_2} - z_{k_1}$. This quantity is always non-negative, and is strictly positive if $z_{k_1} < z_{k_2}$.

In the case of serial cost sharing, we define $p^{\mathrm{SER}} \in \mathbf{R}^n$ by $p_{\pi(i)}^{\mathrm{SER}} = \xi_i'(x_i; x_{-i})$ for all $i \in [n]$, where ξ_i denotes the cost share of user i under serial cost sharing. For any $k \in [n]$, again set $i = \pi^{-1}(k)$. A user j is in the set $L_i(x)$ if and only if $1 \leq \pi(j) \leq |L_i(x)|$. Moreover, for any ℓ such that $|L_i(x)| < \ell < k$, we have $z_\ell = z_k$. By Lemma 5.1, then, we

have

$$\begin{aligned} p_k^{\mathrm{SER}} &= \xi_i'(x_i; x_{-i}) \\ &= 2\left(nx_i - \sum_{j \in L_i(x)} (x_i - x_j) \right) \\ &= 2\left(nz_k - \sum_{\ell=1}^{|L_i(x)|} (z_k - z_\ell) - \sum_{\ell=|L_i|+1}^{k-1} (z_k - z_\ell) \right) \\ &= 2\left((n-k+1)z_k + \sum_{\ell=1}^{k-1} z_\ell \right). \end{aligned}$$

We conclude that $p^{\mathrm{SER}} = B^{\mathrm{SER}} z$, where B^{SER} is an $n \times n$ matrix defined by

$$B_{k\ell}^{\mathrm{SER}} = \begin{cases} 2(n-k+1), & \text{if } k = \ell; \\ 2, & \text{if } k > \ell; \\ 0, & \text{if } k < \ell \end{cases}$$

for any $k, \ell \in [n]$.

Consider any $k_1, k_2 \in [n]$ such that $k_1 < k_2$. We have

$$\begin{aligned} p_{k_2}^{\mathrm{SER}} - p_{k_1}^{\mathrm{SER}} &= 2\left((n-k_2+1)z_{k_2} + \sum_{\ell=1}^{k_2-1} z_\ell - (n-k_1+1)z_{k_1} - \sum_{\ell=1}^{k_1-1} z_\ell \right) \\ &= 2\left((n-k_2+1)(z_{k_2} - z_{k_1}) - (k_2-k_1)z_{k_1} + \sum_{\ell=k_1}^{k_2-1} z_\ell \right) \\ &= 2\left((n-k_2+1)(z_{k_2} - z_{k_1}) + \sum_{\ell=k_1}^{k_2-1} (z_\ell - z_{k_1}) \right). \end{aligned}$$

As $k_2 \leq n$ and $z_\ell \geq z_{k_1}$ for $k_1 \leq \ell < k_2$, this implies that $p_{k_2}^{\mathrm{SER}} - p_{k_1}^{\mathrm{SER}} \geq 0$, with the inequality becoming strict if $z_{k_1} < z_{k_2}$.

Now, by the definition in (5.1) of the cost share ξ_i of a user $i \in [n]$ under the θ-combination,

$$\begin{aligned} p &= \theta p^{\mathrm{SER}} + (1-\theta)p^{\mathrm{AVG}} \\ &= \theta B^{\mathrm{SER}} z + (1-\theta)B^{\mathrm{AVG}} z \\ &= Bz, \end{aligned}$$

where $B = \theta B^{\mathrm{SER}} + (1-\theta)B^{\mathrm{AVG}}$ is the $n \times n$ matrix defined in the corollary statement. The fact that $p_{k_1} \leq p_{k_2}$ for any $k_1, k_2 \in [n]$ such that $k_1 < k_2$, with strict inequality if $z_{k_1} < z_{k_2}$, follows from the corresponding inequalities for the two vectors p^{SER} and p^{AVG}. □

By applying the work of Rosen [45], we can use the properties in Corollary 5.2 to show that the production game under a quadratic cost function and any θ-combination always has a Nash equilibrium.

Theorem 5.4 (Rosen [45]). *For any game with n players in which the action of player i is $x_i \in [0, b_i]$, the payoff function $P_i(x)$ for i is continuous in x, and $P_i(y; x_{-i})$ is concave in y for every fixed vector x_{-i} containing the actions of the users other than i, there exists a Nash equilibrium.*

Corollary 5.5. *For any $\theta \in [0, 1]$, under a quadratic cost function and the θ-combination, the production game has a Nash equilibrium.*

Proof. In the production game, the payoff of user i from (5.2) has the quasi-linear form $P_i(x) = U_i(x_i) - \xi_i(x)$. To restrict the possible actions of a user i to an interval $[0, b_i]$, note that the assumptions that U_i is concave, $U_i(0) = 0$, and $U_i'(0)$ exists and is finite imply that $U_i(x_i) \leq U_i'(0)x_i$. By the subset coverage property of a cost sharing method, $\xi_i(x) = 0$ if $x_i = 0$ and $\xi_i(x) \geq x_i^2$ under the cost function $C(y) = y^2$. Thus, we have $P_i(x) = 0$ if $x_i = 0$ and $P_i(x) \leq U_i'(0)x_i - x_i^2 = x_i(U_i'(0) - x_i)$. When $x_i > U_i'(0)$, then, $P_i(x) < 0$, and so we can assume that the action of user i is in the interval $[0, U_i'(0)]$.

Now, continuity of the payoff function $P_i(x)$ for the θ-combination under a quadratic cost function follows from the continuity of U_i and the continuity of $\xi_i(x)$ from Corollary 5.2. Similarly, noting the form of the payoff function $P_i(y; x_{-i})$ in (5.6), we use the concavity of U_i and the convexity of $\xi_i(y; x_{-i})$ from Corollary 5.2 to conclude that $P_i(y; x_{-i})$

is concave in y for any fixed vector $x \in \mathbf{R}^n_+$. Thus, the conditions of Theorem 5.4 are satisfied by the production game under a quadratic cost function and the θ-combination, and so Theorem 5.4 implies that there exists a Nash equilibrium for the production game. □

The work of Rosen [45] also implies that, if the utility functions of the users satisfy additional properties, then the Nash equilibrium of the production game under a quadratic cost function and any θ-combination is unique. In order to present Rosen's uniqueness theorem, we first introduce some notation. For two vectors $x, r \in \mathbf{R}^n_+$, define the scalar $\sigma(x, r)$ as

$$\sigma(x, r) = \sum_{i=1}^{n} r_i P_i(x),$$

and the pseudogradient of $\sigma(x, r)$ as the vector $g(x, r) \in \mathbf{R}^n$ defined by

$$g(x, r)_i = r_i \left(\frac{\partial P_i(x)}{\partial x_i} \right)$$

for all $i \in [n]$.

Definition 5.5. For any fixed $r \in \mathbf{R}^n_+$, the function $\sigma(x, r)$ is *diagonally strictly concave* for r if, for every distinct $v, w \in \mathbf{R}^n_+$,

$$(w - v)^T g(v, r) + (v - w)^T g(w, r) > 0.$$

Theorem 5.6 (Rosen [45]). *For any game satisfying the conditions of Theorem 5.4 such that the payoff function $P_i(x)$ of each user i has a continuous first derivative with respect to x_i, and the function $\sigma(x, r)$ is diagonally strictly concave for some $r \in \mathbf{R}^n_{++}$, the Nash equilibrium is unique.*

Corollary 5.7. *For any $\theta \in [0, 1]$, under a quadratic cost function, the θ-combination, and a collection of utility functions $U \in \mathcal{U}^n$ such that each function U_i has a continuous first derivative, the Nash equilibrium of the production game is unique.*

As the proof of Corollary 5.7 is not necessary for the following sections, the reader who is primarily interested in the results on price of anarchy in Section 5.3 may skip it.

Proof of Corollary 5.7. From the proof of Corollary 5.5, we know that the production game satisfies the conditions of Theorem 5.4. To see that $P_i(x)$ has a continuous first

derivative with respect to x_i, recall the quasi-linear form of $P_i(x)$ from (5.2), $P_i(x) = U_i(x_i) - \xi_i(x)$. Corollary 5.2 implies that, under a quadratic cost function and any θ-combination, $\xi_i(x)$ has a continuous first derivative with respect to x_i. Combining this with the assumption that U_i has a continuous first derivative, we conclude that $P_i(x)$ also has a continuous first derivative with respect to x_i.

Now, to apply Theorem 5.6 to the production game, we show that $\sigma(x, r)$ is diagonally strictly concave for the vector $r \in \mathbf{R}^n_{++}$ with $r_i = 1$ for all $i \in [n]$. For any two distinct vectors $v, w \in \mathbf{R}^n_+$,

$$\begin{aligned}
&(w - v)^T g(v, r) + (v - w)^T g(w, r) \\
&= \sum_{i=1}^{n} (w_i - v_i)(U_i'(v_i) - \xi_i'(v_i; v_{-i}) - U_i'(w_i) + \xi_i'(w_i; w_{-i})) \\
&\geq \sum_{i=1}^{n} (w_i - v_i)(\xi_i'(w_i; w_{-i}) - \xi_i'(v_i; v_{-i})), \qquad (5.11)
\end{aligned}$$

where we have used the fact that $(w_i - v_i)(U_i'(v_i) - U_i'(w_i)) \geq 0$, which follows from the concavity of U_i. Note that, if the sum in (5.11) is strictly positive for both average cost pricing and serial cost sharing, then it is strictly positive whenever $v \neq w$ for any θ-combination as well.

To show that the sum in (5.11) is strictly positive for average cost pricing and serial cost sharing, we use an argument in Rosen's paper [45]. This argument is included here because additional technical issues arise for serial cost sharing that necessitate a slight modification of the argument. For any $\gamma \in [0, 1]$, let $x(\gamma) = \gamma w + (1 - \gamma)v$. Under the cost function $C(y) = y^2$ and average cost pricing, by Lemma 5.1, we have

$$\begin{aligned}
\frac{d\xi_i'(x(\gamma)_i; x(\gamma)_{-i})}{d\gamma} &= \sum_{j=1}^{n} \left(\frac{\partial \xi_i'(x(\gamma)_i; x(\gamma)_{-i})}{\partial x(\gamma)_j} \right) \frac{dx(\gamma)_j}{d\gamma} \\
&= 2(w_i - v_i) + \sum_{j \neq i} (w_j - v_j) \\
&= \sum_{j=1}^{n} B_{ij}^{\mathrm{AVG}} (w_j - v_j),
\end{aligned}$$

where B^{AVG} is the $n \times n$ matrix defined in the proof of Corollary 5.3. The sum in (5.11)

can now be expressed as

$$\begin{aligned}\sum_{i=1}^{n}(w_i - v_i)(\xi_i'(w_i; w_{-i}) - \xi_i'(v_i; v_{-i})) &= \sum_{i=1}^{n}(w_i - v_i)\int_0^1 \frac{d\xi_i'(x(\gamma)_i; x(\gamma)_{-i})}{d\gamma}d\gamma \\ &= \sum_{i=1}^{n}(w_i - v_i)\int_0^1 \sum_{j=1}^{n} B_{ij}^{\mathrm{AVG}}(w_j - v_j)d\gamma \\ &= \int_0^1 \sum_{i=1}^{n}(w_i - v_i)\sum_{j=1}^{n} B_{ij}^{\mathrm{AVG}}(w_j - v_j)d\gamma \\ &= (w - v)^T B^{\mathrm{AVG}}(w - v).\end{aligned}$$

From this equation, we see that if B^{AVG} is positive definite, then the sum in (5.11) is strictly positive. Observe that $B^{\mathrm{AVG}} = I + E$, where E is an $n \times n$ matrix with $E_{ij} = 1$ for all $i, j \in [n]$, and I is the $n \times n$ identity matrix. Thus, for any vector $x \in \mathbf{R}^n$ such that $x^T x \neq 0$,

$$\begin{aligned}x^T B^{\mathrm{AVG}} x &= x^T I x + x^T E x \\ &= x^T x + \left(\sum_{i=1}^{n} x_i\right)^2 \\ &> 0.\end{aligned}$$

This argument does not apply directly to serial cost sharing because the partial derivative $\frac{\partial \xi_i'(x(\gamma)_i; x(\gamma)_{-i})}{\partial x(\gamma)_j}$ does not exist when $x(\gamma)_i = x(\gamma)_j$. Whenever $x(\gamma)_i \neq x(\gamma)_j$, however, this partial derivative exists. As such, we partition the users into sets so that two users $i \neq j$ are in the same set if and only if $v_i = v_j$ and $w_i = w_j$. Let $S_1, \ldots, S_m$ be the sets in the partition of $[n]$. For all $k \in [m]$, let $n_k = |S_k|$, and let i_k be any user in the set S_k.

Note that, for any two users $i \neq j$ in the same set S_k, $x(\gamma)_i = x(\gamma)_j = x(\gamma)_{i_k}$ for all $\gamma \in [0, 1]$. By Lemma 5.1, then, under a quadratic cost function and serial cost sharing, for any $\gamma \in [0, 1]$, we have

$$\begin{aligned}\sum_{i=1}^{n}(w_i - v_i)\xi_i'(x(\gamma)_i; x(\gamma)_{-i}) &= \sum_{k=1}^{m}\sum_{i \in S_k}(w_i - v_i)\xi_i'(x(\gamma)_i; x(\gamma)_{-i}) \\ &= \sum_{k=1}^{m} n_k(w_{i_k} - v_{i_k})\xi_{i_k}'(x(\gamma)_{i_k}; x(\gamma)_{-i_k}),\end{aligned}$$

and

$$\xi'_{i_k}(x(\gamma)_{i_k}; x(\gamma)_{-i_k}) = 2\left(nx(\gamma)_{i_k} - \sum_{j:x(\gamma)_j < x(\gamma)_{i_k}} (x(\gamma)_{i_k} - x(\gamma)_j)\right)$$
$$= 2\left(nx(\gamma)_{i_k} - \sum_{\ell:x(\gamma)_{i_\ell} < x(\gamma)_{i_k}} n_\ell (x(\gamma)_{i_k} - x(\gamma)_{i_\ell})\right). \tag{5.12}$$

The definition of the sets S_k implies that, for any $k, \ell \in [m]$, $k \neq \ell$, there is at most one value of γ in the interval $[0,1]$ such that $x(\gamma)_{i_k} = x(\gamma)_{i_\ell}$. Therefore, there exists a finite sequence of values $0 = \gamma_0 < \gamma_1 < \cdots < \gamma_{h-1} < \gamma_h = 1$ such that, for all $t \in [h]$ and $\gamma \in (\gamma_{t-1}, \gamma_t)$, the values $x(\gamma)_{i_k}$ for $k \in [m]$ are all distinct.

Consider an open interval (γ_{t-1}, γ_t) for any $t \in [h]$. By the definition of the γ_t values, for any $k \neq \ell$, either $x(\gamma)_{i_k} < x(\gamma)_{i_\ell}$ for all γ in this interval, or $x(\gamma)_{i_k} > x(\gamma)_{i_\ell}$ for all γ in the interval. The equation in (5.12) now implies that, for any $\gamma \in (\gamma_{t-1}, \gamma_t)$ and $k \in [m]$,

$$\begin{aligned}
&\frac{d\xi'_{i_k}(x(\gamma)_{i_k}; x(\gamma)_{-i_k})}{d\gamma} \\
&= \sum_{\ell=1}^{m} \left(\frac{\partial \xi'_{i_k}(x(\gamma)_{i_k}; x(\gamma)_{-i_k})}{\partial x(\gamma)_{i_\ell}}\right) \frac{dx(\gamma)_{i_\ell}}{d\gamma} \\
&= 2\left(\left(n - \sum_{\ell:x(\gamma)_{i_\ell} < x(\gamma)_{i_k}} n_\ell\right)(w_{i_k} - v_{i_k}) + \sum_{\ell:x(\gamma)_{i_\ell} < x(\gamma)_{i_k}} n_\ell (w_{i_\ell} - v_{i_\ell})\right) \\
&= \sum_{\ell=1}^{m} H^t_{k\ell} n_\ell (w_{i_\ell} - v_{i_\ell}),
\end{aligned}$$

where H^t is a $m \times m$ matrix defined by

$$H^t_{k\ell} = \begin{cases} 2\left(\frac{n - \sum_{j:x(\gamma')_{i_j} < x(\gamma')_{i_k}} n_j}{n_k}\right), & \text{if } k = \ell; \\ 2, & \text{if } k \neq \ell \text{ and } x(\gamma')_{i_k} > x(\gamma')_{i_\ell}; \\ 0, & \text{if } k \neq \ell \text{ and } x(\gamma')_{i_k} < x(\gamma')_{i_\ell} \end{cases}$$

for any $\gamma' \in (\gamma_{t-1}, \gamma_t)$ and all $k, \ell \in [m]$.

We can extend the definition of this derivative to the endpoints of the interval, γ_{t-1} and γ_t, because the value of the expression in (5.12) for $\xi'_{i_k}(x(\gamma)_{i_k}; x(\gamma)_{-i_k})$ is unchanged

by including additional terms in the sum in (5.12) corresponding to those $\ell \in [m]$ such that $x(\gamma)_{i_k} \neq x(\gamma)_{i_\ell}$ for those γ in the open interval, but $x(\gamma)_{i_k} = x(\gamma)_{i_\ell}$ for a $\gamma \in \{\gamma_{t-1}, \gamma_t\}$. Therefore,

$$\begin{aligned}\xi'_{i_k}(x(\gamma_t)_{i_k}; x(\gamma_t)_{-i_k}) - \xi'_{i_k}(x(\gamma_{t-1})_{i_k}; x(\gamma_{t-1})_{-i_k}) &= \int_{\gamma_{t-1}}^{\gamma_t} \frac{d\xi'_{i_k}(x(\gamma)_{i_k}; x(\gamma)_{-i_k})}{d\gamma} d\gamma \\ &= \int_{\gamma_{t-1}}^{\gamma_t} \sum_{\ell=1}^{m} H^t_{k\ell} n_\ell (w_{i_\ell} - v_{i_\ell}) d\gamma,\end{aligned}$$

and we can write the sum in (5.11) as

$$\begin{aligned}&\sum_{i=1}^{n} (w_i - v_i)(\xi'_i(w_i; w_{-i}) - \xi'_i(v_i; v_{-i})) \\ &\quad = \sum_{i=1}^{n} (w_i - v_i) \sum_{t=1}^{h} (\xi'_i(x(\gamma_t)_i; x(\gamma_t)_{-i}) - \xi'_i(x(\gamma_{t-1})_i; x(\gamma_{t-1})_{-i})) \\ &\quad = \sum_{t=1}^{h} \sum_{i=1}^{n} (w_i - v_i)(\xi'_i(x(\gamma_t)_i; x(\gamma_t)_{-i}) - \xi'_i(x(\gamma_{t-1})_i; x(\gamma_{t-1})_{-i})) \\ &\quad = \sum_{t=1}^{h} \sum_{k=1}^{m} n_k (w_{i_k} - v_{i_k})(\xi'_{i_k}(x(\gamma_t)_{i_k}; x(\gamma_t)_{-i_k}) - \xi'_{i_k}(x(\gamma_{t-1})_{i_k}; x(\gamma_{t-1})_{-i_k})) \\ &\quad = \sum_{t=1}^{h} \sum_{k=1}^{m} n_k (w_{i_k} - v_{i_k}) \int_{\gamma_{t-1}}^{\gamma_t} \sum_{\ell=1}^{m} H^t_{k\ell} n_\ell (w_{i_\ell} - v_{i_\ell}) d\gamma \\ &\quad = \sum_{t=1}^{h} \int_{\gamma_{t-1}}^{\gamma_t} \sum_{k=1}^{m} n_k (w_{i_k} - v_{i_k}) \sum_{\ell=1}^{m} H^t_{k\ell} n_\ell (w_{i_\ell} - v_{i_\ell}) d\gamma \\ &\quad = \sum_{t=1}^{h} \int_{\gamma_{t-1}}^{\gamma_t} u^T H^t u d\gamma \\ &\quad = \sum_{t=1}^{h} \int_{\gamma_{t-1}}^{\gamma_t} u^T \left(\frac{1}{2} \left(H^t + \left(H^t\right)^T \right) \right) u d\gamma,\end{aligned}$$

where $u \in \mathbf{R}^m$ is the vector defined by $u_k = n_k(w_{i_k} - v_{i_k})$ for all $k \in [m]$.

This equation shows that if the matrix $(1/2)(H^t + (H^t)^T)$ is positive definite for each $t \in [h]$, then the sum in (5.11) is strictly positive. To see that this matrix is positive definite, note that $(1/2)(H^t + (H^t)^T) = D + E$, where E is a $m \times m$ matrix with $E_{k\ell} = 1$ for all $k, \ell \in [m]$, and D is a $m \times m$ diagonal matrix. The entries on the diagonal of D are

given by, for $k \in [m]$ and any $\gamma' \in (\gamma_{t-1}, \gamma_t)$,

$$D_{kk} = 2\left(\frac{n - \sum_{j:x(\gamma')_{i_j} < x(\gamma')_{i_k}} n_j}{n_k}\right) - 1 \geq 2\left(\frac{n_k}{n_k}\right) - 1,$$

and so $D_{kk} \geq 1$. A calculation similar to the one for the matrix B^{AVG} now shows that $(1/2)(H^t + (H^t)^T)$ is positive definite.

Combining all of the preceding analysis, we conclude that, under a quadratic cost function and any θ-combination, the function $\sigma(x, r)$ is diagonally strictly concave for the vector r with $r_i = 1$ for all $i \in [n]$. Theorem 5.6 now implies that the Nash equilibrium of the production game is unique. □

5.3 Tight Bounds on Price of Anarchy

We now show how to determine the price of anarchy of every θ-combination under a quadratic cost function. First, we present a lemma that allows us to restrict our attention to linear utility functions U_i. This lemma is proved by Moulin [35], and is based on the work of Johari and Tsitsiklis [20].

Lemma 5.8. *Fix a cost function C, number of users n, and cost sharing method ξ. For a collection of utility functions $U \in \mathcal{U}^n$, let x^* be any Nash equilibrium of the production game under C, n, ξ, and U. Define a new collection of utility functions $V \in \mathcal{U}^n$ by $V_i(y_i) = U_i'(x_i^*)y_i$ for all $i \in [n]$. Then x^* is also a Nash equilibrium of the production game under C, n, ξ, V, and*

$$\frac{A_U(x^*)}{OPT_U} \geq \frac{A_V(x^*)}{OPT_V}.$$

Lemma 5.8 implies that the infimum in the defining expression for the price of anarchy in (5.5) is approached by collections of linear utility functions. As a result, we now assume that the utility functions of the users are all linear functions. For any $i \in [n]$, let $U_i(x_i) = a_i x_i$ for a constant $a_i \in \mathbf{R}_+$. We order the users so that $a_1 \leq a_2 \leq \cdots \leq a_n$. Let $a \in \mathbf{R}_+^n$ be the vector containing the a_i values.

As a first step towards determining the price of anarchy of any θ-combination under a quadratic cost function, we consider the maximum possible aggregate surplus that can be

attained by any vector of requested quantities under a collection of utility functions defined by a vector a. We can assume that $a_n > 0$. By the equation in (5.4), for any vector $x \in \mathbf{R}_+^n$ such that $\sum_{i=1}^n x_i = s$, the aggregate surplus of the system is $A(x) = \sum_{i=1}^n a_i x_i - s^2$. Among all vectors x such that the quantities x_i sum to s, then, $A(x)$ is maximized by the vector x with $x_i = 0$ for all $i \in [n-1]$, and $x_n = s$. For this vector, the aggregate surplus is $A(x) = a_n s - s^2$. The value of s that maximizes this quantity is $s = a_n/2$, and the maximum possible aggregate surplus is $\mathrm{OPT}_a = a_n^2/4$.

We now turn to the production game under the θ-combination and linear utility functions with $a_n > 0$. Corollary 5.5 implies that the game has a Nash equilibrium x^*. In the following lemma, we show that the requested quantities in x^* are in the same order as the a_i values, determine a remarkable formula for the aggregate surplus of the system under x^*, and develop a constraint that relates the x_i^* values to a_n.

Lemma 5.9. *For the cost function $C(y) = y^2$, any number of users n, any $\theta \in [0,1]$, and any $a \in \mathbf{R}_+^n$ such that $a_1 \le a_2 \le \cdots \le a_n$ and $a_n > 0$, let x^* be a Nash equilibrium of the corresponding production game under the θ-combination.*

(i) *The requested quantities in x^* are in the order $x_1^* \le x_2^* \le \cdots \le x_n^*$.*

(ii) *The aggregate surplus of the system under x^* is*

$$A(x^*) = \sum_{i=1}^{n} (2\theta(n-i)+1)\,(x_i^*)^2\,. \tag{5.13}$$

(iii) *The components of x^* satisfy the equation*

$$(1+\theta)\sum_{i=1}^{n-1} x_i^* + 2x_n^* = a_n. \tag{5.14}$$

Proof. By Corollary 5.2, the function $\xi_i(y; x_{-i})$ for any user $i \in [n]$ is convex and differentiable in y for any fixed $x \in \mathbf{R}_+^n$. Since the utility function $U_i(x_i) = a_i x_i$ is linear, it is differentiable, and so the Nash equilibrium conditions in (5.7) for the production game in this setting become

$$\begin{aligned} x_i^* > 0 &\Rightarrow a_i = \xi_i'(x_i^*; x_{-i}^*); \\ x_i^* = 0 &\Rightarrow a_i \le \xi_i'(0; x_{-i}^*). \end{aligned} \tag{5.15}$$

To see that $x_1^* \leq x_2^* \leq \cdots \leq x_n^*$, suppose for the purpose of contradiction that there are two users i_1 and i_2 such that $i_1 < i_2$ and $x_{i_1}^* > x_{i_2}^*$. Because $x_{i_2}^* \geq 0$, we have $x_{i_1}^* > 0$, and so the Nash equilibrium conditions in (5.15) imply that $a_{i_1} = \xi'_{i_1}(x_{i_1}^*; x_{-i_1}^*)$ and $a_{i_2} \leq \xi'_{i_2}(x_{i_2}^*; x_{-i_2}^*)$. By the ordering of the a_i values, $a_{i_1} \leq a_{i_2}$, from which we conclude that $\xi'_{i_1}(x_{i_1}^*; x_{-i_1}^*) \leq \xi'_{i_2}(x_{i_2}^*; x_{-i_2}^*)$. On the other hand, since $x_{i_1}^* > x_{i_2}^*$, Corollary 5.3 implies that $\xi'_{i_1}(x_{i_1}^*; x_{-i_1}^*) > \xi'_{i_2}(x_{i_2}^*; x_{-i_2}^*)$, contradicting this inequality.

Given that $x_1^* \leq x_2^* \leq \cdots \leq x_n^*$, we can apply Corollary 5.3 with the ordering π of x^* being the identity permutation to rewrite the Nash equilibrium conditions in (5.15) as

$$\begin{aligned} x_i^* > 0 &\Rightarrow a_i = \sum_{j=1}^{n} B_{ij} x_j^*; \\ x_i^* = 0 &\Rightarrow a_i \leq \sum_{j=1}^{n} B_{ij} x_j^*, \end{aligned} \tag{5.16}$$

where B is the $n \times n$ matrix defined in the statement of Corollary 5.3. Now, from the equation in (5.4), the aggregate surplus of the system under x^* is

$$A(x^*) = \sum_{i=1}^{n} a_i x_i^* - \left(\sum_{i=1}^{n} x_i^* \right)^2. \tag{5.17}$$

The Nash equilibrium conditions in (5.16) imply that

$$\begin{aligned} \sum_{i=1}^{n} a_i x_i^* &= \sum_{i=1}^{n} x_i^* \sum_{j=1}^{n} B_{ij} x_j^* \\ &= (x^*)^T B x^* \\ &= (x^*)^T \left(\frac{1}{2} \left(B + B^T \right) \right) x^*. \end{aligned}$$

Substituting this expression into the equation in (5.17), we obtain

$$\begin{aligned} A(x^*) &= (x^*)^T \left(\frac{1}{2} \left(B + B^T \right) \right) x^* - (x^*)^T E x^* \\ &= (x^*)^T \left(\frac{1}{2} \left(B + B^T \right) - E \right) x^*, \end{aligned} \tag{5.18}$$

where E is an $n \times n$ matrix with $E_{ij} = 1$ for all $i, j \in [n]$.

Define the symmetric matrix D as $D = (1/2)(B + B^T) - E$. By the definition of B, the diagonal entries of D are $D_{ii} = B_{ii} - 1 = 2\theta(n-i)+1$ for all $i \in [n]$. For any $i, j \in [n]$ such that $i \neq j$, we have $D_{ij} = D_{ji} = (1/2)(1+\theta+1-\theta) - 1 = 0$. Thus, D is a diagonal matrix, and the equation involving the quadratic form $(x^*)^T Dx^*$ in (5.18) simplifies to

$$A(x^*) = \sum_{i=1}^{n} D_{ii} (x_i^*)^2 ,$$

which yields the expression in (5.13) upon substitution of the D_{ii} values.

If $a_n > 0$, then the Nash equilibrium condition in (5.16) for user n cannot be satisfied by the vector x^* with $x_i^* = 0$ for all $i \in [n]$. We must have $x_n^* > 0$, then, and the Nash equilibrium condition for user n is

$$a_n = \sum_{i=1}^{n} B_{ni} x_i^*.$$

By substituting the B_{ni} values, we obtain the equation in (5.14). □

For every vector $a \in \mathbf{R}_+^n$ with $a_n = t > 0$, the maximum possible aggregate surplus over all $x \in \mathbf{R}_+^n$ is the same. As such, the minimum possible ratio between the aggregate surplus at a Nash equilibrium of the production game and the maximum possible aggregate surplus can be determined by finding the minimum possible aggregate surplus under a Nash equilibrium of the production game over all vectors $a \in \mathbf{R}_+^n$ with $a_n = t$. This is done in the following lemma.

The maximum possible aggregate surplus over all vectors x and the minimum possible aggregate surplus under a Nash equilibrium both scale in proportion to t^2 as we vary t. As such, without loss of generality we assume that $t = 1$ in the following analysis. We denote the set of vectors a with $a_n = 1$ by $\mathcal{A} = \{a \in \mathbf{R}_+^n \mid a_1 \leq a_2 \leq \cdots \leq a_n = 1\}$. Extending our previous notation, we write $A_a(x)$ to denote the aggregate surplus of the system under the utility functions defined by the vector $a \in \mathbf{R}_+^n$ and the vector $x \in \mathbf{R}_+^n$ of requested quantities, and we denote the maximum possible aggregate surplus as

$$\mathrm{OPT}_a = \max_{x \in \mathbf{R}_+^n} A_a(x).$$

Lemma 5.10. *Fix the cost function $C(y) = y^2$, any number of users n, and any $\theta \in [0, 1]$.*

For any vector $a \in \mathcal{A}$, let $\mathcal{E}(a)$ denote the set of Nash equilibria of the production game under the θ-combination and the linear utility functions defined by the vector a. Then

$$\inf_{a \in \mathcal{A}} \inf_{x^* \in \mathcal{E}(a)} A_a(x^*) = \frac{1}{4\Gamma_\theta(n)}, \tag{5.19}$$

where

$$\Gamma_\theta(n) = \frac{(1+\theta)^2}{4} \sum_{i=1}^{n-1} \frac{1}{2\theta i + 1} + 1.$$

Proof. For any vector $a \in \mathcal{A}$ and Nash equilibrium vector $x^* \in \mathcal{E}(a)$, Lemma 5.9 implies that $x_1^* \leq x_2^* \leq \cdots \leq x_n^*$,

$$A_a(x^*) = \sum_{i=1}^{n} (2\theta(n-i)+1)(x_i^*)^2,$$

and

$$(1+\theta) \sum_{i=1}^{n-1} x_i^* + 2x_n^* = 1.$$

As such, a lower bound on the infimum in (5.19) is provided by the value of the convex program

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} (2\theta(n-i)+1)x_i^2 \\ \text{subject to} \quad & (1+\theta) \sum_{i=1}^{n-1} x_i + 2x_n = 1. \end{aligned} \tag{5.20}$$

We introduce a Lagrange multiplier λ for the constraint $1 - (1+\theta)\sum_{i=1}^{n-1} x_i - 2x_n = 0$. Then the Karush-Kuhn-Tucker (KKT) optimality conditions for the program in (5.20) are

$$\begin{gathered} 2(2\theta(n-i)+1)x_i - \lambda(1+\theta) = 0, \quad \forall i \in [n-1]; \\ 2x_n - 2\lambda = 0. \end{gathered}$$

Solving the KKT conditions for the x_i values yields

$$\begin{aligned} x_i &= \left(\frac{1+\theta}{2\theta(n-i)+1}\right)\frac{\lambda}{2}, \quad \forall i \in [n-1]; \\ x_n &= \lambda. \end{aligned}$$

Substituting these values into the equality constraint in (5.20), we obtain

$$\begin{aligned} 1 &= \lambda\left(\frac{(1+\theta)^2}{2}\sum_{i=1}^{n-1}\frac{1}{2\theta(n-i)+1}+2\right) \\ &= 2\lambda\left(\frac{(1+\theta)^2}{4}\sum_{i=1}^{n-1}\frac{1}{2\theta i+1}+1\right) \\ &= 2\lambda\Gamma_\theta(n), \end{aligned}$$

and thus

$$\lambda = \frac{1}{2\Gamma_\theta(n)}.$$

The value of the objective function in (5.20) for this vector x is

$$\begin{aligned} \sum_{i=1}^{n-1}(2\theta(n-i)+1)\left(\frac{1+\theta}{2\theta(n-i)+1}\right)^2\left(\frac{\lambda}{2}\right)^2+\lambda^2 &= \lambda^2\left(\frac{(1+\theta)^2}{4}\sum_{i=1}^{n-1}\frac{1}{2\theta(n-i)+1}+1\right) \\ &= \lambda^2\Gamma_\theta(n) \\ &= \frac{1}{4\Gamma_\theta(n)}. \end{aligned}$$

This quantity is a lower bound on the infimum in (5.19).

To obtain a corresponding upper bound on the infimum in (5.19), consider the vector $x \in \mathbf{R}^n_+$ obtained by solving the KKT conditions and imposing the equality constraint in (5.20). The components of x are

$$\begin{aligned} x_i &= \left(\frac{1+\theta}{2\theta(n-i)+1}\right)\frac{1}{4\Gamma_\theta(n)}, \quad \forall i \in [n-1]; \\ x_n &= \frac{1}{2\Gamma_\theta(n)}, \end{aligned}$$

and so $0 \le x_1 \le x_2 \le \cdots \le x_n$. Define a vector a as $a = Bx$, where B is the matrix

defined in the statement of Corollary 5.3 for the θ-combination. Corollary 5.3 implies that $a_1 \leq a_2 \leq \cdots \leq a_n$. Moreover, because x satisfies the equality constraint in (5.20), we have

$$\begin{aligned} a_n &= \sum_{i=1}^{n-1} B_{ni} x_i + B_{nn} x_n \\ &= (1+\theta) \sum_{i=1}^{n-1} x_i + 2x_n \\ &= 1. \end{aligned}$$

Finally, the vector x satisfies the Nash equilibrium conditions in (5.16), and so it is a Nash equilibrium for the production game under the utility functions defined by a. Since the aggregate surplus of the system under x is

$$A_a(x) = \frac{1}{4\Gamma_\theta(n)},$$

this quantity is also an upper bound on the infimum in (5.19). □

By combining Lemma 5.10 with our previous observations, we can determine the price of anarchy of each θ-combination under a quadratic cost function.

Theorem 5.11. *For the cost function $C(y) = y^2$, any number of users n, and any $\theta \in [0,1]$, the price of anarchy of the θ-combination is*

$$\frac{1}{\Gamma_\theta(n)}.$$

Proof. Lemma 5.8 implies that the price of anarchy in (5.5) can be expressed as

$$\inf_{U \in \mathcal{U}^n} \frac{\inf_{x^* \in \mathcal{E}(U)} A_U(x^*)}{\mathrm{OPT}_U} = \inf_{a \in \mathcal{A}} \frac{\inf_{x^* \in \mathcal{E}(a)} A_a(x^*)}{\mathrm{OPT}_a}.$$

For a vector $a \in \mathcal{A}$, we have $\mathrm{OPT}_a = 1/4$. By Lemma 5.10, then,

$$\begin{aligned} \inf_{U \in \mathcal{U}^n} \frac{\inf_{x^* \in \mathcal{E}(U)} A_U(x^*)}{\mathrm{OPT}_a} &= \frac{\frac{1}{4\Gamma_\theta(n)}}{\frac{1}{4}} \\ &= \frac{1}{\Gamma_\theta(n)}. \end{aligned}$$

□

When $\theta = 0$, $\Gamma_\theta(n) = (n+3)/4$, and so the price of anarchy of average cost pricing under a quadratic cost function is $4/(n+3)$. For any constant $\theta \in (0, 1]$, we have $\Gamma_\theta(n) \sim (1+\theta)^2 \ln n/(8\theta)$ as $n \to \infty$, which means that the price of anarchy of the θ-combination scales as $8\theta/((1+\theta)^2 \ln n)$ for large n. As the function $f(\theta) = \theta/(1+\theta)^2$ is increasing in the interval $\theta \in (0, 1]$, the asymptotic price of anarchy improves as θ increases. The best asymptotic price of anarchy under a quadratic cost function in this class of cost sharing methods is achieved at $\theta = 1$ by serial cost sharing, which has a price of anarchy of $2/(\ln n)$ for large n.

These qualitative comparisons between different θ-combinations are illustrated in Figure 5.2, which shows the price of anarchy of the θ-combination for several values of θ. For small values of n, serial cost sharing has a worse price of anarchy than average cost pricing, but as n increases the price of anarchy of serial cost sharing becomes the best among these cost sharing methods. Note also that for the larger values of n, the gap in the price of anarchy between average cost pricing and the (1/4)-combination is much larger than the gap between the (1/4)-combination and serial cost sharing.

5.4 Limitations of Analysis

Under a quadratic cost function, the θ-combinations share the property that the partial derivatives $\xi_i'(x_i; x_{-i})$ can be expressed as linear equations as in Corollary 5.3. In this sense, the matrix B defined in Corollary 5.3 for any $\theta \in [0, 1]$ characterizes these cost sharing methods. A natural question is to what extent the method of analysis applied here to the θ-combinations for quadratic cost functions extends to other cost sharing methods for which the partial derivatives are linear equations, but the matrix analogous to B does not have the same structure as in the case of the θ-combinations.

As in Corollary 5.3, for a vector $x \in \mathbf{R}_+^n$, let π be an ordering of x and let z be the ordered version of x. For a cost sharing method ξ, define the vector $p \in \mathbf{R}^n$ by $p_{\pi(i)} = \xi_i'(x_i; x_{-i})$. Suppose that ξ satisfies the property that under the quadratic cost function $C(y) = y^2$, $p = Bz$ for some $n \times n$ matrix B of constants.

Consider a vector $x \in \mathbf{R}_+^n$ such that, for some $i \in [n]$, $x_i > 0$, and $x_j = 0$ for all $j \neq i$. Since the largest requested quantity is that of user i, the partial derivative of the cost

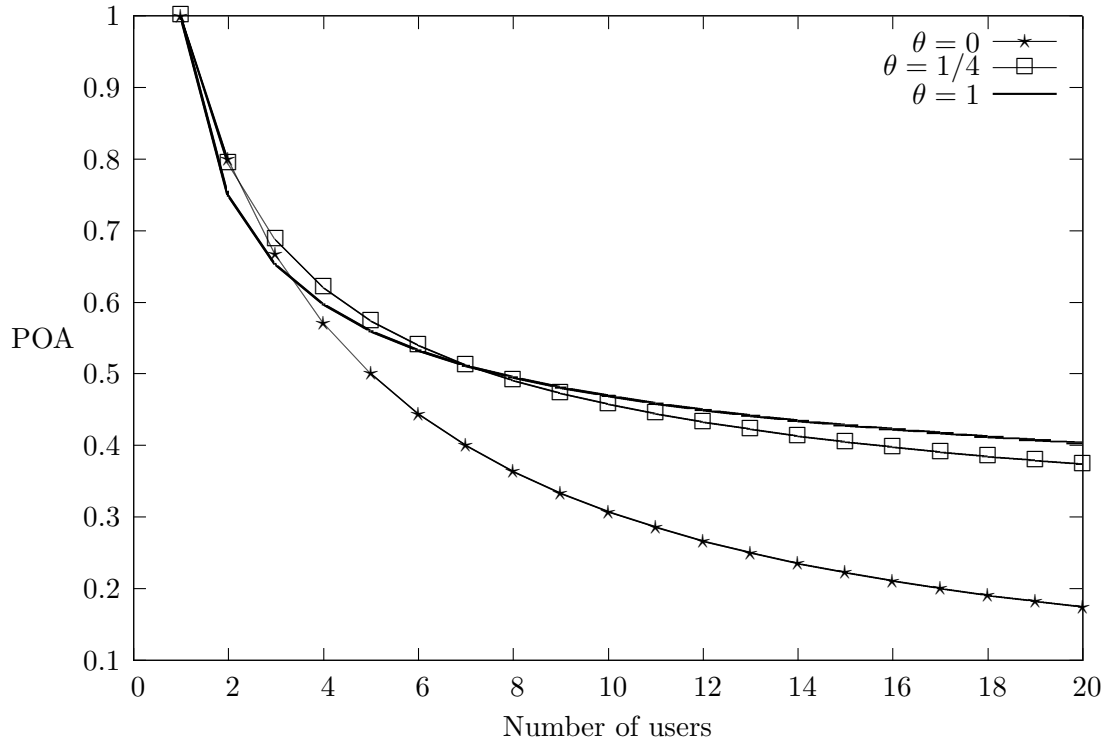

Figure 5.2: The price of anarchy (POA) of the θ-combination for $n \leq 20$ and $\theta = 0$ (average cost pricing), $\theta = 1/4$, and $\theta = 1$ (serial cost sharing).

share of user i with respect to x_i is

$$\xi_i'(x_i; x_{-i}) = B_{nn} x_i. \tag{5.21}$$

By the subset coverage property of a cost sharing method, the cost share of user i when all other users are requesting the quantity zero must be $\xi_i(x) = C(x_i)$. This implies that the partial derivative $\xi_i'(x_i; x_{-i})$ must be $\xi_i'(x_i; x_{-i}) = C'(x_i)$, which for a quadratic cost function is $\xi_i'(x_i; x_{-i}) = 2x_i$. From the equation in (5.21), then, we conclude that $B_{nn} = 2$.

One of the key steps in the analysis of the price of anarchy of a θ-combination is the observation that the matrix $D = (1/2)(B + B^T) - E$ is diagonal. This property simplifies the objective function in the convex program in (5.20), and enables the solution of the KKT conditions in closed form. The matrix D is diagonal if and only if, for all $k, \ell \in [n]$ such that $k \neq \ell$, $B_{k\ell} + B_{\ell k} = 2$. As extending our approach beyond matrices B for which

the corresponding matrix D is not diagonal appears to be a challenge, our analysis is effectively limited to such matrices. In the following, then, we assume that $B_{k\ell} + B_{\ell k} = 2$ for all $k \neq \ell$.

Consider now a vector $x \in \mathbf{R}_+^n$ such that all of the entries of x are distinct, with the exception of two users $i_1 \neq i_2$ such that $x_{i_1} = x_{i_2}$. Let $k = |L_{i_1}(x)| + 1$. There are two possible orderings of x, π_1 and π_2, which satisfy $\pi_1(j) = \pi_2(j)$ for all $j \notin \{i_1, i_2\}$, and $\{\pi_1(i_1), \pi_1(i_2)\} = \{\pi_2(i_1), \pi_2(i_2)\} = \{k, k+1\}$. A cost sharing method that is symmetric in the partial derivatives $\xi_i'(x_i; x_{-i})$ as well as the cost shares $\xi_i(x)$ would satisfy $p_k = p_{k+1}$. This implies that

$$\begin{aligned}
0 &= p_{k+1} - p_k \\
&= \sum_{\ell=1}^{n} B_{(k+1)\ell} z_\ell - \sum_{\ell=1}^{n} B_{k\ell} z_\ell \\
&= \sum_{\ell=1}^{k-1} (B_{(k+1)\ell} - B_{k\ell}) z_\ell + \sum_{\ell=k+2}^{n} (B_{(k+1)\ell} - B_{k\ell}) z_\ell \\
&\quad + (B_{(k+1)k} - B_{kk} + B_{(k+1)(k+1)} - B_{k(k+1)}) z_k, \qquad (5.22)
\end{aligned}$$

because $z_k = z_{k+1}$.

Now, if the entries in the B matrix are constants, in order for the equation in (5.22) to hold as the z_ℓ values vary by infinitesimal amounts, we must have $B_{(k+1)\ell} - B_{k\ell} = 0$ for all $\ell < k$ and $\ell > k+1$. Since this constraint holds for any $k \in [n]$, by applying the additional property that $B_{k\ell} + B_{\ell k} = 2$ for all $k \neq \ell$, we can conclude that there exists some constant α such that $B_{\ell k} = \alpha$ and $B_{k\ell} = 2 - \alpha$ for all $k < \ell$. The equation in (5.22) now simplifies to

$$0 = (B_{kk} - B_{(k+1)(k+1)} + 2(1-\alpha)) z_k.$$

Because this equation must continue to hold as z_k varies, it implies that $B_{kk} - B_{(k+1)(k+1)} + 2(1-\alpha) = 0$. Combining this with the fact that $B_{nn} = 2$, we obtain $B_{kk} = 2(1 + (\alpha - 1)(n-k))$ for all $k \in [n]$.

For any user $i \in [n]$ and a fixed vector $x \in \mathbf{R}_+^n$, let $z \in \mathbf{R}_+^{n-1}$ be the ordered version of the vector x_{-i} containing the quantities requested by the users other than i. Suppose that $0 < z_1 < z_2 < \cdots < z_{n-1}$, and consider the partial derivative $\xi_i'(y; x_{-i})$ as x remains fixed and the quantity y requested by i varies. When $z_{k-1} < y < z_k$ for $k \in [n-1]$, where

z_0 is defined to be zero, we have

$$\xi_i'(y; x_{-i}) = \alpha \sum_{\ell=1}^{k-1} z_\ell + B_{kk} y + (2-\alpha) \sum_{\ell=k}^{n-1} z_\ell.$$

If $B_{kk} < 0$, then increasing y by an infinitesimal amount while leaving x fixed would decrease this partial derivative. As a result, the function $\xi_i(y; x_{-i})$ would not be convex in y for every fixed x, and we would not be able to apply Rosen's Theorem 5.4 to guarantee the existence of a Nash equilibrium in the production game. To ensure that $B_{kk} \geq 0$ for all k and n, then, we impose the constraint $\alpha \geq 1$.

On the other hand, if $\alpha > 2$, then $B_{k\ell} = 2 - \alpha < 0$ for all $k < \ell$. In this case, the partial derivative $\xi_i'(y; x_{-i})$ for $y = 0$ would be

$$\begin{aligned} \xi_i'(y; x_{-i}) &= (2-\alpha) \sum_{\ell=1}^{n-1} z_\ell \\ &< 0, \end{aligned}$$

and so the cost share of user i for requesting a small quantity $x_i > 0$ would be smaller than the cost share for requesting $x_i = 0$. By the subset coverage property of a cost sharing method, however, any user requesting the quantity zero must be assigned a cost share of zero, and any user requesting a positive quantity under a quadratic cost function must be assigned a positive cost share. Thus, we also require that $\alpha \leq 2$.

Note now that the class of B matrices to which our analysis applies, through the transformation $\alpha = 1 + \theta$, is in one-to-one correpondence with the θ-combinations. In this sense, the θ-combinations, each of which is an interpolation between average cost pricing and serial cost sharing, comprise the exact class of cost sharing methods for which our approach determines the price of anarchy under a quadratic cost function.

Chapter 6

Conclusion

In this thesis, we have studied the equilibria of several distributed systems. We have taken two distinct perspectives on the role of a system designer. In the first, the designer can specify the behavior of the agents in the system, and the goal of the designer is to develop a distributed algorithm for a computational problem. For the problems of information dissemination, computation of separable functions, and convex optimization, we have shown that, if the agents execute certain simple algorithms, the system will converge over time to an equilibrium in which each node has an approximate solution.

The second perspective is that the users in the system behave independently and selfishly, and the goal of a system designer is to specify the operation of the system so that this competitive behavior leads to an equilibrium in which no user can benefit by taking a different action. Furthermore, it is natural for the designer to try to ensure that the equilibrium is not much worse than an optimal solution in terms of the aggregate welfare of all the users. In the production game in which a cost sharing method is used to allocate the cost of production of a good to the users that receive the good, we have studied a class of cost sharing methods that interpolates between the well-known methods of average cost pricing and serial cost sharing. We determined the price of anarchy of each cost sharing method in the class when the cost function is quadratic.

There are opportunities for improving our understanding of the work in this thesis. In particular, it would be useful to have lower bounds on the convergence times of the distributed algorithms. We suspect that our upper bounds on convergence times are loose in many cases, but proving general lower bounds that depend on the structure of the

communication graph seems to be a challenge.

On the topic of cost sharing methods in the production game, a natural question is whether our analysis can be effective for other cost functions. While the method of analysis is general in the sense that it could be applied to other cost functions, the task of solving the convex program used in the analysis becomes more difficult when the cost function is not quadratic. In general, given any cost function, one would want to know the cost sharing method that achieves the best price of anarchy for that cost function. Answering this question appears to require significant new techniques.

Bibliography

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

[2] M. C. Azizoğlu and Ö. Eğecioğlu. The isoperimetric number of d-dimensional k-ary arrays. *International Journal of Foundations of Computer Science*, 10(3):289–300, 1999.

[3] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proceedings of RANDOM 2002*, pages 1–10, 2002.

[4] Y. Bartal, J. W. Byers, and D. Raz. Fast, distributed approximation algorithms for positive linear programming with applications to flow control. *SIAM Journal on Computing*, 33(6):1261–1279, 2004.

[5] N. Berger, C. Borgs, J. T. Chayes, and A. Saberi. On the spread of viruses on the Internet. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 301–310, 2005.

[6] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[7] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, June 2006.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[9] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997.

[10] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of the 20th International Conference on Data Engineering*, pages 449–460, 2004.

[11] S. Deb and M. Médard. Algebraic gossip: A network coding approach to optimal multiple rumor mongering. In *Proceedings of the 42nd Annual Allerton Conference on Communication, Control, and Computing*, 2004.

[12] S. Deb, M. Médard, and C. Choute. Algebraic gossip: A network coding approach to optimal multiple rumor mongering. *IEEE Transactions on Information Theory*, 52(6):2486–2507, June 2006.

[13] A. Dembo and O. Zeitouni. *Large Deviations Techniques and Applications*. Springer, second edition, 1998.

[14] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

[15] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[16] A. M. Frieze and G. R. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10:57–77, 1985.

[17] R. G. Gallager. A minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communications*, COM-25(l):73–85, 1977.

[18] A. Ganesh, L. Massoulié, and D. Towsley. The effect of network topology on the spread of epidemics. In *Proceedings of IEEE INFOCOM 2005*, pages 1455–1466, 2005.

[19] N. Garg and N. Young. On-line end-to-end congestion control. In *Proceedings of the 43rd Annual Symposium on Foundations of Computer Science*, pages 303–312, 2002.

[20] R. Johari and J. N. Tsitsiklis. Efficiency loss in a network resource allocation game. *Mathematics of Operation Research*, 29(3):407–435, August 2004.

[21] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 565–574, 2000.

[22] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252, March 1998.

[23] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 482–491, 2003.

[24] D. Kempe and J. Kleinberg. Protocols and impossibility results for gossip-based communication mechanisms. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 471–480, 2002.

[25] D. Kempe, J. Kleinberg, and A. Demers. Spatial gossip and resource location protocols. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 163–172, 2001.

[26] D. Kempe and F. McSherry. A decentralized algorithm for spectral analysis. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 561–568, 2004.

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

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

[29] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 448–457, 1993.

[30] E. Modiano, D. Shah, and G. Zussman. Maximizing throughput in wireless networks via gossiping. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, pages 27–38, 2006.

[31] D. Mosk-Aoyama, T. Roughgarden, and D. Shah. Fully distributed algorithms for convex optimization problems. In *Proceedings of the 21st International Symposium on Distributed Computing*, pages 492–493, 2007.

[32] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing*, pages 113–122, 2006.

[33] D. Mosk-Aoyama and D. Shah. Information dissemination via network coding. In *Proceedings of the 2006 IEEE International Symposium on Information Theory*, pages 1748–1752, 2006.

[34] D. Mosk-Aoyama and D. Shah. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, July 2008.

[35] H. Moulin. The price of anarchy of serial, average and incremental cost sharing. *Economic Theory*, 36(3):379–405, September 2008.

[36] H. Moulin and S. Shenker. Serial cost sharing. *Econometrica*, 60(5):1009–1037, September 1992.

[37] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 250–262, 2004.

[38] M. J. Neely. Distributed and secure computation of convex programs over a network of connected processors. In *Proceedings of the DCDIS 4th International Conference on Engineering Applications and Computational Algorithms*, pages 498–503, 2005.

[39] C. Papadimitriou and M. Yannakakis. Linear programming without the matrix. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 121–129, 1993.

[40] B. Pittel. On spreading a rumor. *SIAM Journal of Applied Mathematics*, 47(1):213–223, 1987.

[41] Y. Rabani, A. Sinclair, and R. Wanka. Local divergence of Markov chains and the analysis of iterative load-balancing schemes. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 694–703, 1998.

[42] R. Ravi. Rapid rumor ramification: Approximating the minimum broadcast time. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 202–213, 1994.

[43] O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, 155(1):157–187, January 2002.

[44] R. T. Rockafellar. *Network Flows and Monotropic Optimization*. Wiley-Interscience, 1984. Republished by Athena Scientific, 1998.

[45] J. B. Rosen. Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica*, 33(3):520–534, July 1965.

[46] S. J. Shenker. Making greed work in networks: A game-theoretic analysis of switch service disciplines. *IEEE/ACM Transactions on Networking*, 3(6):819–831, December 1995.

[47] A. Sinclair. *Algorithms for Random Generation and Counting: A Markov Chain Approach*. Birkhäuser, Boston, 1993.

[48] R. Srikant. *The Mathematics of Internet Congestion Control*. Birkhäuser, 2004.

[49] J. N. Tsitsiklis. *Problems in Decentralized Decision Making and Computation*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1984.

[50] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, 1986.