On Matrix Factorization and Scheduling for Finite-time Average-consensus

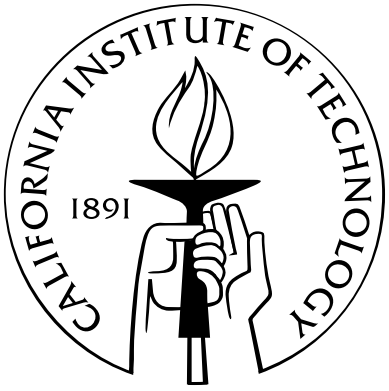
Thesis by

Chih-Kai Ko

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2010

(Defended January 5, 2010)

Acknowledgements

First and foremost, I would like to thank my advisor, Professor Leonard J. Schulman. This thesis would never have existed without his support, inspiration, guidance, and patience. To him, I offer my most sincere gratitude. In addition, I am grateful to my thesis committee members: Professors John Doyle, Tracy Ho, Steven Low, and Chris Umans for their helpful inputs and suggestions.

I am particularly indebted to my whole family: my dad, Jin-Wen Ko, my mom, Chu-Chih Li, and my sister, Mon-Lin Ko, for their love and encouragement; especially my parents, who have always been there to offer their guidance and support. I owe a lot to my dear wife, Xiaojie Gao, who has helped me in countless ways and provided constructive criticism and suggestions in research. I feel very fortunate to have met her at Caltech. I would also like to acknowledge my 2.5 week old[1] daughter, Kailin, whose arrival brought much joy and chaos into our lives[2].

I also want to acknowledge Sidharth Jaggi, my first year roommate at the Braun Graduate House in Caltech. As a more senior graduate student, Sidharth helped me tremendously in getting adjusted to the rigors of Caltech. He is a great friend, roommate, and mentor. His research guidance played a crucial role in my National Science Foundation Graduate Fellowship award. I would also like to acknowledge the National Science Foundation for providing me with financial support to pursue my graduate studies. Their support helped me continue with my Ph.D. studies.

I would like to thank my colleagues and friends whom I met at JPL, especially Scott Darden, Clement Lee, Andrew Gray, and Winston Kwong. They are truly

[1]Kailin was 2.5 weeks old at the time of the thesis defense.

[2]I am very glad that I listened to my wise advisor when he warned me against scheduling my thesis defense one week after my daughter's due date.

wonderful friends, and spending time with them helped relieve the stresses of Caltech life. I am happy to say that I now have a more definite response to their most frequent question: "Hey man, when are you going to graduate?" I would also like to thank my JPL supervisors, Clayton Okino and Norm Lay, for the opportunity and privilege to work at JPL. At JPL, I learned many new technologies and more importantly, I gained some great friends.

Lastly, I would like to give thanks to God. It is by the grace of God that I am able to pursue my graduate education at such a prestigious institution. I feel challenged and humbled everyday working with and learning from people who are much smarter than I am. I thank Him for the many blessings throughout the years and I give Him credit for my accomplishments.

Abstract

We study the problem of communication scheduling for finite-time average-consensus in arbitrary connected networks. Viewing this consensus problem as a factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$ by network-admissible families of matrices, we prove the existence of finite factorizations, provide scheduling algorithms for finite-time average consensus, and derive almost tight lower bounds on the size of the minimal factorization.

Contents

Chapter 1

Introduction

1.1 Consensus Problems

In a consensus problem, a group of agents (or network nodes) try to reach agreement on a certain quantity of interest that depends on their states [29]. Consensus problems arise in diverse areas such as oscillator synchronization [32, 31], flocking behavior of swarms [42], rendezvous problems [2, 18, 26], multi-sensor data fusion [35], multi-vehicle formation control [16], satellite alignment [3, 28], distributed computation [25], and many more. When the objective is to agree upon the average, it is an *average-consensus* problem. A motivating example (from [5]) is a network of temperature sensors needing to average their readings to combat fluctuations in ambient temperature and sensor variations.

Many efficient algorithms exist under various settings, *e.g.*, [13, 9, 5, 7, 22]. Although the majority of the proposed algorithms offer rapid convergence, in general, many cannot guarantee consensus in finite time. In this thesis, we study algorithms that achieve average-consensus in finite time for arbitrarily connected networks under various constraints. We view the network consensus problem as a network commodity redistribution problem. To make these ideas concrete, we begin with an example.

1.2 An Example with Milk and Cookies

Consider n children going to school, each equipped with a backpack of unit capacity. The children fill their backpacks with a combination of milk and cookies. If a child brings x units of milk ($0 \leq x \leq 1$), then the backpack also contains $1 - x$ units of cookies. At school, they sit in an assigned seating arrangement. Once seated, the children begin to exchange milk and cookies amongst themselves in hopes that everyone has the same share of milk and cookies at the end of the exchange process. The exchange process is subject to the following constraints:

- (Proximity Communication) Because we don't want children wandering in the classroom, each child can only exchange with someone seated in close proximity.
- (Serial Communication) Because we want to maintain order, at most one pair of children can exchange their goods at any given time.
- (Proportional Fairness) Because we want to maintain fairness, the exchanges must be proportionally fair. That is, if Alice gives Bob 10% of everything she has, then Bob must give Alice 10% of what he has.

Our role as the teacher is to devise a sequence of pairwise exchanges (*i.e.*, a schedule) so that at some finite time, say T, all the children have equal shares of milk and cookies. Furthermore, we require that our schedule achieve even distribution regardless of initial conditions. That is, our schedule can only depend on the seating configuration and not on the initial quantities of milk and cookies brought by the children.

Using symbols, we now make this problem more precise. Let $x_i(t)$ denote the amount of cookies possessed by child i at time t. Initially, at time $t = 0$, each i-th child brings $x_i(0)$ units of cookies and $1 - x_i(0)$ units of milk in his/her backpack where

$$0 \leq x_i(0) \leq 1.$$

We assume that the total quantity of milk and the total quantity of cookies are non-zero. That is, someone has brought some cookies to school and someone has brought some milk to school:

$$0 < \sum_{i=1}^{n} x_i(0) < n.$$

After following our schedule, we desire that at some finite time $T \geq 0$, we have for all i,

$$x_i(T) = \frac{1}{n} \sum_{j=1}^{n} x_j(0)$$

for all initial values of $x_j(0)$. That is, our schedule must lead to finite-time average-consensus regardless of what quantity of goods each child has brought to school.

The seating arrangement is represented by a connected undirected graph $G = (V, E)$ with n nodes where $V = \{1, 2, \ldots, n\}$. The i-th child is seated at node i. If $(i, j) \in E$, then we say that i and j are *neighbors* in G. The proximity constraint dictates that each child can only exchange goods directly with his/her neighbor in the graph. The fairness constraint requires that if neighbors i and j exchange their goods at time t then

$$\begin{aligned} x_i(t+1) &= \lambda_t\, x_i(t) + (1 - \lambda_t)\, x_j(t), \\ x_j(t+1) &= (1 - \lambda_t)\, x_i(t) + \lambda_t\, x_j(t). \end{aligned}$$

Here, λ_t ($0 \leq \lambda_t \leq 1$) represents the proportion of goods exchanged at time t. It shall be useful to view the exchange process as a weighted-averaging operation between two students.

1.3 Averaging in Wireless Sensor Networks

At this point, we can draw an analogy with wireless sensor networks. Instead of children and snacks, imagine a deployment of n wireless temperature sensors in

an area. These sensors measure the local temperature, establish a communication network, then run an averaging protocol to average their readings so that at some time T, we can query any sensor to get the exact global average temperature. In this setting, we run into similar constraints:

- (Proximity Communication) Sensors are typically power-constrained, so they can only communicate with other sensors in close proximity.
- (Serial Communication) To minimize communication interference, we may only want one pair of sensors to communicate at any given time.

The proportional fairness constraint from our classroom example does not have an exact analogy here. In a typical sensor network, the quantity that is exchanged is information. We can do much more with information than we can with a physical commodity: we can use source coding to decrease redundancy, channel coding to increase robustness, and network coding to increase throughput. In a commodity distribution network, we cannot do such coding operations on physical goods. We are often limited by local capacity constraints (*e.g.*, unit-capacity backpacks in the children example).

1.4 Basic Notation

Although most of the material in this thesis is self-contained, we will make use of elementary knowledge of graph theory, linear algebra, and algorithms. We refer readers unfamiliar with these areas to [14], [36], [10], as basic references to graph theory, linear algebra, and algorithms, respectively.

Before we venture into the problem formulation in its full generality, we need to establish some basic notations:

- $G = (V, E)$ denotes a connected undirected graph on n vertices with vertex set V and edge set $E \subseteq V \times V$. For convenience, we sometimes write $v \in G$ to denote a vertex v from the vertex set V of graph G.

- Given a graph G and two vertices $u, v \in G$, we write $\Delta(u, v)$ to denote the distance between nodes u and v. That is, $\Delta(u, v)$ is the length of the shortest simple path between u and v.

- Given a graph G, we define its *diameter*, D_G, as

$$D_G = \max_{u,v} \Delta(u, v),$$

its pairwise *total distance*, D_G^{total}, as

$$D_G^{\text{total}} = \frac{1}{2} \sum_{u \in G} \sum_{v \in G} \Delta(u, v),$$

its pairwise *average distance*, $\bar{D}_G$, as

$$\bar{D}_G = \frac{D_G^{\text{total}}}{\binom{n}{2}},$$

and its *median*, μ_G, as

$$\mu_G = \arg\min_{u \in G} \sum_{v \in G} \Delta(u, v).$$

When the underlying graph is clear from the context, we omit the subscript G and simply write D, D^{total}, $\bar{D}$, and μ to refer to its diameter, total distance, average distance, and median, respectively.

- A *tree* is a graph without cycles. That is, any two vertices in the tree are connected by exactly one simple path.

- A *spanning tree* $\mathcal{T}$ of G is a subgraph that is a tree and contains all vertices of G.

- $\mathbb{R}$,$\mathbb{Q}$, and $\mathbb{N}$ denote the set of real, rational, and natural numbers, respectively.

- Given functions f and g, we write $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0$.

- Given functions f and g, we write $f(n) = \Omega(g(n))$ if there exist positive constants c and n_0 such that $0 \leq c\, g(n) \leq f(n)$ for all $n \geq n_0$.
- Given functions f and g, we write $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
- Boldface $\mathbf{1} \in \mathbb{R}^n$ denotes a column vector of all 1's.
- $I \in \mathbb{R}^{n \times n}$ denotes the $n \times n$ identity matrix.
- $e_i \in \mathbb{R}^n$ is an unit vector with 1 in the i-th coordinate and 0's in other coordinates.
- Given a matrix M, M^{T} denotes its matrix transpose.
- For a sequence of T square matrices

$$\{W(0), W(1), W(2), \cdots, W(T-1)\},$$

where $W(t) \in \mathbb{R}^{n \times n}$ for $0 \leq t < T$, we write $\prod_{t=0}^{T-1} W(t)$ to denote the product

$$\prod_{t=0}^{T-1} W(t) = W(T-1) \cdot W(T-2) \cdots W(0).$$

We will also need the notion of G-admissible matrices:

Definition 1.1. *Given a graph $G = (V, E)$, we say a matrix $W \in \mathbb{R}^{n \times n}$ is G-*admissible *if*

1. (Conservation) *W is column stochastic: $W_{ij} \geq 0$ for all i, j; $\mathbf{1}^T W = \mathbf{1}^T$; and*
2. (Connectivity) *For $i \neq j$, $W_{ij} = 0$ if $(i, j) \notin E$.*

With slight abuse of notation, we write $W \in G$ if W is G-admissible; and $\mathcal{S} \subseteq G$ to denote that $\mathcal{S}$ is a subset of G-admissible matrices.

Loosely speaking, a G-admissible matrix performs averaging according to the topology specified by G. Notice that the proximity communication constraint is implicit in the definition.

1.5 Problem Statement

Given a connected undirected graph $G = (V, E)$, imagine vertices V as nodes in a network connected according to E. For each node $i \in V$, let $x_i(t)$ denote the value of node i at time step t. Define $x(t) = [x_1(t), x_2(t) \cdots, x_n(t)]^\mathsf{T}$. Given any set of initial values $x(0)$, we are interested in a finite sequence of weighted-averaging operations, $W(t) \in G$, that allows all nodes to reach average-consensus in finite time T, *i.e.*, $x(T) = \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}x(0)$. Expressed as a linear dynamical system, the values in our network evolve as

$$x(t+1) = W(t)\,x(t) \tag{1.1}$$

with $W(t) \in G$. The G-admissibility requirement limits our averaging operations to those that are consistent with the network topology. Ultimately, we desire a ***finite*** sequence of G-admissible matrices $W(0), \cdots, W(T-1)$ such that

$$x(T) = \prod_{t=0}^{T-1} W(t)\,x(0) = \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}x(0)$$

for all $x(0) \in \mathbb{R}^n$. The requirement that the sequence of G-admissible matrices results in average-consensus for all $x(0) \in \mathbb{R}^n$ suggests a matrix factorization perspective to finite-time average-consensus. This is made precise by the following lemma:

Lemma 1.2. *Let $W(0), W(1), \cdots, W(T-1) \in \mathbb{R}^{n\times n}$ be a finite sequence of T matrices, then $W(T-1)W(T-2)\cdots W(0) = \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$ iff*

$$W(T-1)W(T-2)\cdots W(0)\,x(0) = \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}x(0) \tag{1.2}$$

for all $x(0) \in \mathbb{R}^n$.

Proof. The "*only if*" direction is clear, so we show the "*if*" direction. Let e_i denote an unit-vector in $\mathbb{R}^n$ with 1 in the i-th coordinate. If we take $x(0) = e_i$, then equation

(1.2) becomes

$$\frac{1}{n}\mathbf{1} = W(T-1)W(T-2)\cdots W(0)\, e_i.$$

So the i-th column of the product $W(T-1)W(T-2)\cdots W(0)$ must be $\frac{1}{n}\mathbf{1}$ for any i and the lemma follows. □

To characterize the fastest consensus time (*i.e.*, minimal factorization) we define the following:

Definition 1.3. *Given a set of matrices $\mathcal{S} \subseteq \mathbb{R}^{n\times n}$, define*

$$T^*_{\mathcal{S}} \triangleq \min\left\{T : \exists W(t) \in \mathcal{S} \text{ with } \prod_{t=0}^{T-1} W(t) = \frac{1}{n}\mathbf{1}\mathbf{1}^T\right\}$$

when it exists. When such a finite factorization does not exist, we define $T^*_{\mathcal{S}} = \infty$. For convenience, we write T^*_G when $\mathcal{S}$ is the set of G-admissible matrices.

Thus, T^*_G is the minimum consensus time.

In this thesis, we shall address the following problems:

- *(Existence)* Given G, does there exist a sequence of $W(t)$'s such that T^*_G is finite?
- *(Algorithm)* How can we find a G-admissible factorization, if it exists?
- *(Minimality)* If it exists, what is the minimal of such factorization?

As it turns out, the set of G-admissible matrices may be too general in the context of network consensus problems. Some scenarios, such as the classroom example in Section 1.2, require additional constraints that further restrict the factors to certain subsets of G-admissible matrices. In addition to connectivity and conservation constraints imposed by G-admissibility, networked nodes may act synchronously or asynchronously, nodes may be power-constrained, and nodes may have different levels of knowledge or computation power. Each of these restrictions further constrain the factors of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$ to specific subsets of G-admissible matrices in the context of consensus problems.

Definition 1.4. *Given a graph G on n vertices and a set $\mathcal{S} \subseteq \mathbb{R}^{n\times n}$ of matrices, we define*

$$\mathcal{S} \cap G = \left\{W \in \mathbb{R}^{n\times n} : W \in \mathcal{S} \text{ and } W \in G\right\}.$$

Many existing consensus algorithms correspond to factoring $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$ using $\mathcal{S} \cap G$ with differing $\mathcal{S}$. For example, it may be desirable to factor $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$ using doubly stochastic matrices:

$$W(t) \in \{W : W\mathbf{1} = \mathbf{1} \text{ and } W \in G\}$$

so the average is a fixed point of iteration (1.1); *i.e.*, $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}x(0) = W(t) \cdot \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}x(0)$ for all t. In this thesis, we shall explore the following specific subsets of G-admissible matrices:

- (Pairwise Exchanges) Factorization using $W(t) \in G \cap P_1$ where

  $$P_1 \triangleq \{W : |\{i : W_{ii} \neq 1\}| \leq 2\}.$$

  That is, we restrict each averaging step to a pair of nodes in G. This is similar to the one-child-at-a-time serial communication constraint of Section 1.2.

- (Pairwise 50%-50% Exchanges) Factorization using $W(t) \in G \cap S_1'$ where

  $$S_1' \triangleq \left\{I - \frac{(e_i - e_j)(e_i - e_j)^\mathsf{T}}{2} : 0 \leq i, j < n\right\}.$$

  In addition to the proximity and serial communication constraints of Section 1.2, we enforce a 50%-50% exchange where each time two nodes average their values. This restriction also corresponds to the gossip-based asynchronous algorithms, *c.f.* [5].

- (Pairwise Symmetric Weighted Exchanges) As we will discover shortly, the set $G \cap S_1'$ often fails to admit finite-time average-consensus (*i.e.*, $T_{G\cap S_1'} = \infty$ for many graphs G) and we must look beyond $G \cap S_1'$ if we desire a finite factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$. Therefore, we consider the following generalization of

S_1':

$$S_1 \triangleq \left\{ I - \frac{(e_i - e_j)(e_i - e_j)^{\mathsf{T}}}{m} : 1 \leq m \in \mathbb{Q}; 0 \leq i, j < n \right\}.$$

The set S_1 allows pair-wise weighted-averages and notice that $S_1' \subset S_1$.

- (Symmetric Exchanges) Lastly, we consider

$$S \triangleq \left\{ W \in \mathbb{Q}^{n \times n} : W = W^{\mathsf{T}} \right\}.$$

Note that the matrices in S are doubly stochastic (*i.e.,* $\mathbf{1}^{\mathsf{T}} W = \mathbf{1}^{\mathsf{T}}$ and $W\mathbf{1} = \mathbf{1}$). The set S allows distribution of mass by symmetric weighted averages; yet disallows aggregation steps such as those matrices in the G-admissible set. Such operations are often impossible under typical network node assumptions (*i.e.,* topology awareness, computational limitations, distributed behavior... etc.) or capacity constraints.

1.6 Contributions

Our main contributions are as follows:

- We show that any connected graph admits a finite G-admissible factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}$and $T_G^* = \Theta(D)$, where D is the diameter of the graph.

- In the context of gossip-based asynchronous algorithms (*e.g.,* [5]), we provide necessary conditions for the finiteness of $T^*_{G \cap S_1'}$. We derive a lower bound on $T^*_{G \cap S_1'} = \Omega(n \log n)$ and show that it can be achieved when G is the boolean hypercube.

- When we are restricted to proportionally fair serial averaging (*i.e.,* one pair of nodes at a time), we exhibit a tree-based algorithm proving

$$T^*_{G \cap S_1} = O\left(n \log n + n \cdot \bar{D}_G\right).$$

- When G is a tree, our algorithm is provably optimal. That is,

$$T^*_{G\cap S_1} = \Omega\left(n\log n + n\cdot \bar{D}_G\right).$$

- More surprisingly, when G is a general graph, our tree-based algorithm is off by, at most, a polylog factor:

$$T^*_{G\cap S_1} = \Omega\left(n\log n + n\log^{-2}(n)\cdot \bar{D}_G\right).$$

  Interestingly, this result implies that using the entire network for averaging can yield, at most, a modest increase in performance. It is an open question whether this $O(log^2 n)$ gap can be eliminated.

Since our results are of a centralized nature, our consensus algorithm runtime lower bounds the runtime of any distributed finite-time average-consensus algorithm.

1.7 Organization

The rest of the thesis is organized as follows: In Chapter 2, we review the relevant existing literature. In Chapter 3, we show that a G-admissible factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$ always exists and that $T^*_G = \Theta(D)$, where D is the diameter of G. In Chapters 4, 5, and 6, we study factorization by subsets of G-admissible matrices, $G\cap S_1'$, $G\cap S_1$, and $G\cap S$ respectively. Finally, we close with potential research directions and concluding remarks in Chapter 7.

Chapter 2

Related Work

We introduced the finite-time average-consensus problem as a matrix factorization problem to emphasize that we require an *exact* average in *finite* time. Much of the existing work analyzes asymptotic properties of $\prod_t W(t)$ as $t \to \infty$; see, for example, [5, 29, 16]. If we relax our exactness requirement and allow a randomized choice of product matrices, one can define the ϵ-average time of distribution $\mathcal{D}$ [5] as

$$T_{\text{ave}}(\epsilon, \mathcal{D}) \triangleq \sup_{x(0)} \inf \left\{ \mathbf{P}_{\mathcal{D}} \left(\frac{\|x(t) - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}\|}{\|x(0)\|} \geq \epsilon \right) \leq \epsilon \right\},$$

where $\epsilon > 0$ and $\mathcal{D}$ is a probability distribution on the set of G-admissible matrices, and $W(t)$ are drawn independently from $\mathcal{D}$. The choice of $\mathcal{D}$ reflects the behavior of different distributed consensus algorithms. For a trivial $\mathcal{D}$, *e.g.*, pick a $W \in G$ with $W\mathbf{1} = \mathbf{1}$ and let $W(t) = W$ for all t; the ϵ-average time is governed by the second largest eigenvalue of W [29, 16]. Optimization of $T_{\text{ave}}(\epsilon, W)$ over W can be written as a semidefinite program (SDP) [43], hence solved efficiently numerically. Tight bounds on $T_{\text{ave}}(\epsilon, \mathcal{D})$, when $\mathcal{D}$ corresponds to synchronous and asynchronous distributed gossip algorithms, can be found in [5]. For a more comprehensive and detailed overview of convergence behavior of consensus-type problems, we refer the reader to [5, 29, 16] and the references within.

Although exponentially-fast convergence is sufficient in many cases, it is sometimes desirable to achieve exact convergence in finite-time. A number of authors have studied finite-time consensus in the framework of continuous-time systems:

Cortés [11] employed nonsmooth gradient flows to design gradient-based coordination algorithms that achieve average-consensus in finite-time. Using finite-time semi-stability theory, Hui et al [17] designed finite-time consensus algorithms for a class of thermodynamically motivated dynamic network. Wang and Xiao [41] used finite-time Lyapunov functions to derive finite-time guarantees of specific coordination protocols.

In the discrete-time setting, Sundaram and Hadjicostis [38, 37] studied the finite-time consensus problem for discrete-time systems. By allowing sufficient computation power and memory for the network nodes, [38] showed that nodes in certain linear time-invariant systems can compute their averages after a finite number of linear iterations. The basic idea is that of observability from control theory. Given enough time, the nodes will have observed enough to reconstruct the initial state of the system. At which time, they can compute the correct average. Kingston and Beard [20] studied average-consensus problems in networks with switching topologies. Using a special consensus protocol, they showed that if the topology switches to a fully connected graph, then finite-time average-consensus is possible.

As consensus problems have and continue to receive wide interest, researchers have considered many model variations. Some popular variations include: quantization [4, 19, 23, 24, 8, 6], switched topologies [30, 27, 40, 44], time delay [30, 33, 39, 45], and routing and node mobility [15, 34, 45].

Our work is most closely related to [22] and [21]. In [22], Ko and Shi examined link scheduling on the complete graph to achieve finite-time average-consensus. They provided necessary and sufficient conditions for finite-time consensus and computed the minimum consensus time on the boolean hypercube. In [21], Ko and Gao introduced the matrix factorization perspective to consensus problems. They provided algorithms for finite-time average consensus and showed worst case graph examples which matched the algorithm runtime. This thesis provides several tight bounds for general graphs and thus subsumes [22] and [21].

Chapter 3

G-admissible Factorization

3.1 Pairwise Exchanges

We begin our study with factorization by $(G \cap P_1)$ matrices, instead of the G-admissible factorization, because $(G \cap P_1)$ factorization is more easily understood and its algorithm is more straightforward. Since $(G \cap P_1) \subseteq G$, the existence of a $(G \cap P_1)$ implies the existence of a G-admissible factorization.

Recall the definition of the set P_1:

$$P_1 \triangleq \{W : 1 \leq |\{i : W_{ii} \neq 1\}| \leq 2\}.$$

The set $G \cap P_1$ restricts our averaging operation to one pair of nodes at any given time. This is similar to the one-child-at-a-time serial communication constraint of Section 1.2.

To prove the existence of a finite $(G \cap P_1)$-admissible factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$, consider Algorithm 3.1. The basic idea of this algorithm is simple: nodes pass all their goods to one fixed "aggregator" node. The aggregator node then propagates the appropriate amount of goods back into the network so that everyone has an equal amount at the end. More specifically, we start with a spanning tree of G and arbitrarily designate a node as the root. After designating the root, we keep track of the number of descendants of each node (including itself) in the n-dimensional vector d. Starting from the leaves of the spanning tree, the algorithm traverses up

towards the root. Along the way, each node gives all its goods onto its parent and then removes itself. This process terminates when a only single vertex remains. At this point, the remaining node contains the sum of all initial node values. The second part of the algorithm (Algorithm 3.2) traverses back down the tree while re-distributing the values to achieve average-consensus at termination. The vector d allows us to propagate an appropriate amount of goods to each child node in order to achieve consensus.

Algorithm 3.1: GATHER-PROPAGATE

Input: Graph G, initial values x
Output: $x \leftarrow \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}x$

$d \leftarrow$ vector of 1's indexed by $V(G)$
$T \leftarrow$ a spanning tree of G
while T *is not a single vertex* **do**
  Pick a leaf $v \in V(T)$
  Let $e = (u, v)$ be the edge attaching v to T
  $\begin{bmatrix} x_u \\ x_v \end{bmatrix} \leftarrow \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_u \\ x_v \end{bmatrix}$
  $d_u \leftarrow d_u + d_v$
  $T \leftarrow (V \backslash \{v\}, E \backslash \{e\})$
end
Let $u \leftarrow$ remaining vertex of T
PROPAGATE(T, u, d) `// See Algorithm 3.2`

Algorithm 3.2: PROPAGATE

Input: T, u, d
Output: $x \leftarrow \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}x$

foreach *neighbor* v *of* u **do**
  $\begin{bmatrix} x_u \\ x_v \end{bmatrix} \leftarrow \frac{1}{d_u} \begin{bmatrix} d_u - d_v \\ d_v \end{bmatrix} x_u$
  $E \leftarrow E \backslash \{(u, v)\}$
  PROPAGATE(T, v, d)
  $d_u = d_u - d_v$
end

To translate Algorithm 3.1 into a $(G \cap P_1)$-admissible factorization, notice that

line 6 corresponds to a $(G \cap P_1)$-admissible matrix W with

$$W_{ij} = \begin{cases} 1 & \text{if } i = j \neq v, \\ 1 & \text{if } i = u \text{ and } j = v, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

Similarly, line 2 of Algorithm 3.2 corresponds to a $(G \cap P_1)$-admissible matrix W with

$$W_{ij} = \begin{cases} (d_u - d_v)/d_u & \text{if } i = j = u, \\ d_v/d_u & \text{if } i = v \text{ and } j = u, \\ 1 & \text{if } i = j \neq u, \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

Thus, one can construct a finite factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$using $2(n-1)$ $(G \cap P_1)$-admissible matrices: $n-1$ matrices of type (3.1) followed by $n-1$ matrices of type (3.2). Summarizing into a theorem:

Theorem 3.1. *For any connected graph $G = (V, E)$ on n vertices, there exists a finite $(G \cap P_1)$-admissible factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$. Furthermore, $T^*_{G\cap P_1} \leq 2(n-1)$ and Algorithm 3.1 exhibits such a factorization.*

To see that our upper bound is tight (up to constants), we consider a connected graph on n vertices: let $G = (V, E)$ with $V = \{0, \cdots, n-1\}$. Fix the initial values $x(0)$ as

$$x_i(0) = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Since all of the mass is contained in node-0, we require at least $n-1$ averaging operations to distribute mass to other nodes, because each operation (*i.e.,* multiplication by a matrix in $(G \cap P_1)$) can only propagate goods by one additional node. Thus, $T^*_{G\cap P_1} = \Omega(n)$. Summarizing into a theorem:

Theorem 3.2.

$$T^*_{G\cap P_1} = \Theta(n)$$

3.2 G-admissible Factorization

The existence of a finite G-admissible factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$ is implied by the existence of a finite $(G \cap P_1)$ factorization (see Theorem 3.1). Unlike a $(G \cap P_1)$-factorization, a G-admissible factorization allows one to use several edges at each consensus step. We can rewrite Algorithm 3.1, as in Algorithm 3.3, and combine several $G \cap P_1$ matrices into a G-admissible matrix to reduce the complexity of the factorization.

Algorithm 3.3: GATHER-PROPAGATE

Input: Graph G, initial values x
Output: $x \leftarrow \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}x$

$d \leftarrow$ vector of 1's indexed by $V(G)$
$\mathcal{T} \leftarrow$ a spanning tree of G with root r arbitrarily picked
foreach $v \in V(\mathcal{T})$ **do**
  $l_v \leftarrow \Delta(r, v)$ *i.e.,* the distance from r to v
end
for $\alpha \leftarrow \max_v l_v$ **to** *1* **do**
  foreach v *such that* $l_v = \alpha$ **do**
    v gives all its value onto its parent u, *i.e.,* , $\begin{bmatrix} x_u \\ x_v \end{bmatrix} \leftarrow \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_u \\ x_v \end{bmatrix}$
    $d_u \leftarrow d_u + d_v$
  end
end
for $\alpha \leftarrow 0$ **to** $\max_v l_v - 1$ **do**
  foreach u *such that* $l_u = \alpha$ **do**
    $\{v_1, \cdots, v_\beta\} \leftarrow$ set of children of u

$$\begin{bmatrix} x_u \\ x_{v_1} \\ x_{v_2} \\ \vdots \\ x_{v_\beta} \end{bmatrix} \leftarrow \frac{1}{d_u} \begin{bmatrix} d_u - d_{v_1} - d_{v_2} - \cdots - d_{v_\beta} \\ d_{v_1} \\ d_{v_2} \\ \vdots \\ d_{v_\beta} \end{bmatrix} x_u$$

  end
end

Algorithm 3.3 starts from vertices farthest from the root of a spanning tree of G and traverses upwards to give all goods to the root. The second part of the algorithm traverses back down the tree while re-distributing the values to achieve average-consensus at termination. The main difference from Algorithm 3.1 is that all nodes at each level simultaneously transfer their goods to their parents. In the propagation stage, a node propagates the appropriate value to all its children at once.

To translate Algorithm 3.3 into a G-admissible factorization, notice that the computations of x updates in each *for* loop (line 6) can be translated into a single G-admissible matrix, W, with

$$W_{ij} = \begin{cases} 1 & \text{if } i = j \notin V_\alpha, \\ 1 & \text{if } j \in V_\alpha \text{ and } i \text{ is } j\text{'s parent}, \\ 0 & \text{otherwise,} \end{cases} \tag{3.3}$$

where $V_\alpha = \{v : l_v = \alpha\}$.

Similarly, updates in each **for** loop of line 15 corresponds to a G-admissible matrix W with

$$W_{ij} = \begin{cases} (d_j - \sum_{v:\, v \text{ is a child of } j} d_v)/d_j & \text{if } i = j \in V_\alpha, \\ d_i/d_j & \text{if } j \in V_\alpha \text{ and } i \text{ is a child of } j, \\ 1 & \text{if } i = j \notin V_\alpha, \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

where $V_\alpha = \{v : l_v = \alpha\}$.

The number of *for* iterations at line 6 needed for root r to gather all initial node values is $\max_{v \in V} \Delta(r, v)$. The number of **for** iterations at line 12 needed for re-distributing the values is also $\max_{v \in V} \Delta(r, v)$. It is straight forward to construct a finite factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$ using $2 \max_{v \in V} \Delta(r, v)$ G-admissible matrices: $\max_{v \in V} \Delta(r, v)$ matrices of type (3.3) followed by $\max_{v \in V} \Delta(r, v)$ matrices of type

(3.4). Summarizing into a theorem:

Theorem 3.3. *For any connected graph $G = (V, E)$ on n vertices with diameter D, there exists a finite G-admissible factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^T$. Furthermore, $T_G^* \leq 2D$ and Algorithm 3.3 exhibit such a factorization.*

Proof. This follows from the above discussion and the fact that diameter $D = \max\{\Delta(i, j) : i, j \in V\}$. □

To see that our upper bound is tight (up to constants), we consider a connected graph $G = (V, E)$ on n vertices with diameter D. Assume that the vertex pair (i, j) has maximal distance, *i.e.*, $\Delta(i, j) = D$. Fix the initial values $x(0)$ as

$$x_v(0) = \begin{cases} 1 & \text{if } v = i \\ 0 & \text{otherwise.} \end{cases}$$

Since all of the mass is contained in node-i, we require at least D averaging operations to distribute mass to other nodes, because each operation (*i.e.*, multiplication by a G-admissible matrix) can only propagate goods by distance 1. Thus, $T_G^* = \Omega(D)$ and

Theorem 3.4. *For any connected graph $G = (V, E)$ with diameter D, there exists a finite G-admissible factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^T$. Furthermore,*

$$T_G^* = \Theta(D).$$

Chapter 4

Factorization under Additional Constraints - Pair-wise Averages

In terms of network consensus, allowing factorization by G-admissible matrices may be too strong of a requirement. Often times, communication constraints inherent in the network may restrict the types of G-admissible matrices that we are allowed to use. For example, gossip-based asynchronous consensus algorithms [5] correspond to factorization using $W(t)$'s from $G \cap S_1'$ where

$$S_1' \triangleq \left\{ I - \frac{(e_i - e_j)(e_i - e_j)^{\mathsf{T}}}{2} : 0 \leq i, j < n \right\}.$$

Each matrix in $G \cap S_1'$ corresponds to the averaging of two neighboring node values. Boyd et al [5] studies the ϵ-average time of system (1.1) when the $W(t)$'s are drawn independently and uniformly at random from $G \cap S_1'$. In terms of finite-time consensus, we will show that $T^*_{G \cap S_1'} \geq (n \log n)/2$ using a potential function argument. But first, we require a brief information theory interlude.

4.1 Information Theory

Let $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ and $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ be n-dimensional probability vectors. Let $H(\mathbf{p}) = -\sum_i p_i \log p_i$ denote the binary entropy function. Unless otherwise specified, all $\log$'s are base-2 and we adopt the convention that $0 \log 0 = 0$.

Because $H(\cdot)$ is concave (see Theorem 2.7.3 in [12]), for $0 \leq \lambda \leq 1$,

$$H\left(\lambda\,\mathbf{p} + (1-\lambda)\,\mathbf{q}\right) \geq \lambda\,H(\mathbf{p}) + (1-\lambda)\,H(\mathbf{q})$$

by Jensen's Inequality. Therefore if we replace both $\mathbf{p}$ and $\mathbf{q}$ by their average, the total entropy does not decrease:

$$\Delta H \triangleq H(\lambda\,\mathbf{p} + (1-\lambda)\,\mathbf{q}) + H((1-\lambda)\mathbf{p} + \lambda\mathbf{q}) - H(\mathbf{p}) - H(\mathbf{q}) \geq 0.$$

Let $D(\mathbf{p}\,\|\,\mathbf{q}) = \sum_i p_i \log(p_i/q_i)$ denote the Kullback-Leibler divergence between $\mathbf{p}$ and $\mathbf{q}$. We have

Lemma 4.1.

$$D(\mathbf{p}\,||\,\lambda\,\mathbf{p} + (1-\lambda)\,\mathbf{q}) \leq -\log\lambda.$$

Proof.

$$\begin{aligned}
D(\mathbf{p}\,||\,\lambda\,\mathbf{p} + (1-\lambda)\,\mathbf{q}) &= \sum_i p_i\,\log\frac{p_i}{\lambda\,p_i + (1-\lambda)\,q_i} \\
&= \sum_i p_i\,\log\frac{\lambda^{-1}\,\lambda\,p_i}{\lambda\,p_i + (1-\lambda)\,q_i} \\
&= \sum_i p_i\left(\log\lambda^{-1} + \log\frac{\lambda\,p_i}{\lambda\,p_i + (1-\lambda)\,q_i}\right) \\
&\leq \sum_i p_i \log\lambda^{-1} \qquad (4.1) \\
&= -\log\lambda,
\end{aligned}$$

where the inequality (4.1) is because $\lambda\,p_i \leq \lambda\,p_i + (1-\lambda)\,q_i$ so $\log\frac{\lambda\,p_i}{\lambda\,p_i+(1-\lambda)\,q_i} \leq 0$. □

Now, we can upper bound the increase in entropy due to averaging:

$$\begin{aligned}
\Delta H &= H(\lambda\,\mathbf{p} + (1-\lambda)\,\mathbf{q}) + H((1-\lambda)\mathbf{p} + \lambda\mathbf{q}) - H(\mathbf{p}) - H(\mathbf{q}) \\
&= H(\lambda\,\mathbf{p} + (1-\lambda)\,\mathbf{q}) - (\lambda H(\mathbf{p}) + (1-\lambda)H(\mathbf{q})) + \\
&\quad H((1-\lambda)\mathbf{p} + \lambda\mathbf{q}) - ((1-\lambda)H(\mathbf{p}) + \lambda H(\mathbf{q})) \\
&= \lambda\, D(\mathbf{p}||\lambda\,\mathbf{p} + (1-\lambda)\,\mathbf{q}) + (1-\lambda)\, D(\mathbf{q}||\lambda\,\mathbf{p} + (1-\lambda)\,\mathbf{q}) + \\
&\quad (1-\lambda)\, D(\mathbf{p}||(1-\lambda)\,\mathbf{p} + \lambda\,\mathbf{q}) + \lambda\, D(\mathbf{q}||(1-\lambda)\,\mathbf{p} + \lambda\,\mathbf{q}) \\
&\leq 2\,[-\lambda\log\lambda - (1-\lambda)\log(1-\lambda)] \qquad (4.2) \\
&= 2\,H(\lambda) \\
&\leq 2, \qquad (4.3)
\end{aligned}$$

where inequality (4.2) is due to Lemma 4.1 and inequality (4.3) is because the binary entropy function is upper bounded by 1 [12]. Stated as a lemma:

Lemma 4.2. *The change in total entropy, ΔH, due to the averaging of two probability vectors is bounded by*

$$0 \leq \Delta H \leq 2.$$

We remark that when $\lambda = 1/2$,

$$\Delta H = 2 \cdot \mathrm{JS}(\mathbf{p}, \mathbf{q}) = \mathrm{D}\left(\mathbf{p} \,\middle\|\, \frac{\mathbf{p}+\mathbf{q}}{2}\right) + \mathrm{D}\left(\mathbf{q} \,\middle\|\, \frac{\mathbf{p}+\mathbf{q}}{2}\right),$$

where $\mathrm{JS}(\mathbf{p}, \mathbf{q})$ is the Jensen-Shannon divergence, a symmetrized version of the Kullbeck-Leibler divergence. With an upper bound on the entropy change, we can derive a lower bound on the necessary consensus time for any graph.

Lemma 4.3. *For any connected graph G with n vertices, $T^*_{G\cap S'_1} \geq (n\log n)/2$.*

Proof. Recall that $\mathbf{x}(t) = [x_0(t)\, x_1(t)\, \cdots\, x_{n-1}(t)]^{\mathsf{T}} \in \mathbb{R}^n$ denotes the node values at time t. For each node i, we can express its value at time t as

$$x_i(t) = \mathbf{p}_i(t)^{\mathsf{T}}\mathbf{x}(0),$$

where $\mathbf{p}_i(t)$ is a n-dimensional probability vector. Intuitively, $\mathbf{p}_i(t)$ represents the weighted contributions of $\mathbf{x}(0)$. Initially, for all i,

$$x_i(0) = \mathbf{p}_i(0)^\mathsf{T}\mathbf{x}(0) = \mathbf{e}_i^\mathsf{T}\mathbf{x}(0),$$

where $\mathbf{e}_i$ is the i-th column of the $n \times n$ identity matrix. When consensus is reached at some time, say T,

$$x_i(T) = \mathbf{p}_i(T)^\mathsf{T}\mathbf{x}(0) = \frac{1}{n}\mathbf{1}^\mathsf{T}\mathbf{x}(0)$$

for all i. Define $\phi_i(t) \triangleq H(\mathbf{p}_i(t))$ and $\phi(t) \triangleq \sum_{i=1}^n \phi_i(t)$ so that

$$\phi(T) = n\, H\left(n^{-1}\mathbf{1}\right) = n \log n.$$

Note that $\phi_i(0) = 0$. Since each averaging operation increases the total entropy by at most 2 (Lemma 4.2), we need at least $(n \log n)/2$ such operations to reach $\phi(T)$. Therefore, $T^*_{G \cap S_1'} \geq (n \log n)/2$. □

4.2 Necessary Condition for Finite-time Consensus

Now that we've established a lower bound on consensus time, the question regarding the existence of a finite factorization still remains. As it turns out, restricting the nodes to pairwise averaging prevents the possibility of finite-time consensus in many graphs.

Lemma 4.4. *If the number of vertices n is not a power of 2, then one cannot achieve finite time consensus with $G \cap S_1'$.*

Proof. By contradiction, suppose that finite time consensus is possible. Consider initial node values:

$$x_i(0) = \begin{cases} n & \text{if } i = 0 \\ 0 & \text{otherwise.} \end{cases}$$

At any time $t > 0$, the values of each node is in the form of $n\,a/2^b$ for some $a, b \in \mathbb{Z}^+ \cup \{0\}$. At consensus time T, we have $x_i(T) = 1$ so $n\,a/2^b = 1$ for some $a, b \in$

$\mathbb{Z}^+ \cup \{0\}$. This means

$$n\,a = 2^b$$

which implies that n is a power of 2, a contradiction. □

4.3 Consensus on the Boolean Hypercube

It is natural to wonder whether there are graphs that achieve the lower bound of Lemma 4.3. As it turns out, the boolean hypercube is optimal for finite-time average-consensus. A boolean hypercube is a graph on $n = 2^m$ vertices for some $m \in \mathbb{N}$. Its vertex set is the set of 2^m binary strings of length m. An edge exists between two vertices if the Hamming distance between the vertices is one (*i.e.*, if the two m-bit strings differ by only one bit). It is not difficult to see that performing pairwise averaging along every edge of the hypercube leads to finite time consensus. Since a hypercube with 2^m vertices has $m2^{m-1}$ edges, the lower bound of Lemma 4.3 is achieved.

For a more formal presentation, consider Algorithm 4.1. The "$\oplus$" in the algorithm denotes bit-wise XOR. The overall consensus time of this strategy is

$$(n \log n)/2 = m\,2^{m-1}$$

as the outer **for** loop executes $\log n$ times, the inner loop executes $n/2$ times, and the set of operations in the inner loop corresponds to a single matrix in $(G \cap S_1')$. The correctness of Algorithm 4.1 follows from recognizing that it is essentially a divide and conquer algorithm: dividing a size-n hypercube into two size-$n/2$ hypercubes, performing consensus on both halves, and then averaging between them. Summarizing everything:

Theorem 4.5. *Given a connected graph G on n vertices, finite factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^T$ with $G \cap S_1'$ is possible only if $n = 2^m$ for some $m \in \mathbb{N}$. Furthermore,*

$$T^*_{G\cap S_1'} \geq m\, 2^{m-1}$$

and equality is achieved when G is the boolean m-hypercube.

Algorithm 4.1: SINGLEEDGECONSENSUS

Input: $\{x_1, x_2, \ldots, x_n\}$
Output: For all i, $x_i = n^{-1}\sum_{j=1}^n x_j$

```
for i = 0 to log n − 1 do
    foreach a, b ∈ {0, 1, ..., n − 1} such that a ⊕ b = 2^i do
        M = (x_a + x_b)/2
        x_a = M
        x_b = M
    end
end
```

The boolean hypercube is one of a few graphs that allow for finite-time average-consensus with $G \cap S_1'$. Furthermore, the lower bound of Lemma 4.3 shows that the hypercube structure is optimal in terms of average-consensus time.

Given the negative result of Lemma and the fact that a boolean hypercube has 2^m vertices, it is natural to wonder whether all graphs of size $n = 2^m$ admit finite-time average-consensus. This turns out to be false when we examine a path of length $n = 2^m$. If we initialize the "left-most" node with 1 and the rest of the nodes with 0, *i.e.*,

$$x(0) = [1, 0, \ldots, 0]^\mathsf{T},$$

then it's easy to see that if $x_{i-1}(t) \neq 0, x_i(t) \neq 0$, and $x_{i+1}(t) \neq 0$ then either

$$\begin{aligned} x_{i-1}(t) &\leq x_i(t) < x_{i+1}(t), \text{ or} \\ x_{i-1}(t) &< x_i(t) \leq x_{i+1}(t) \end{aligned}$$

since all masses must flow from "left" to "right." Therefore, there's no way to

achieve an even distribution in finite time on the path with pairwise 50%-50% averages.

Chapter 5

Factorization under Additional Constraints: Pair-wise Symmetric Weighted Averages

We saw the previous section, *i.e.,* Lemma 4.3, that for arbitrary G, the set $G \cap S_1'$ is too restricting for finite-time consensus. Therefore, we must look beyond $G \cap S_1'$ if we desire a finite factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$.

Consider the following generalization of S_1':

$$S_1 \triangleq \left\{ I - \frac{(e_i - e_j)(e_i - e_j)^\mathsf{T}}{m} : 1 \leq m \in \mathbb{Q}; 0 \leq i, j < n \right\}.$$

Notice that $S_1' \subset S_1$ and that the matrices in set S_1 allow pair-wise weighted-averages. To show that finite-time average-consensus is possible using only pair-wise weighted averages at each step (*i.e.,* $T^*_{G \cap S_1} < \infty$), we present Algorithm 5.1. The algorithm first constructs a spanning tree $\mathcal{T}$ of G. After picking an arbitrary leaf node, say v, as the tree's root, the algorithm performs reverse depth-first traversal of $\mathcal{T}$ (*c.f.* Algorithm 5.2) while propagating the appropriate amount of goods upward toward v. When the process is complete, v contains the average amount of goods of all nodes in the tree and can thus be removed from future consideration. At this point, another leaf node is designated as the root and the process repeats until all vertices have been examined. At which time, all nodes will have reached average-consensus.

It is straight forward to construct a sequence of $(G \cap S_1)$ matrices from Line 6 of Algorithm 5.2. Its runtime is $O(n^2)$ since depth-first traversal takes time $O(n)$ (see §22.3 of [10]) and we perform $n-1$ such traversals.

Algorithm 5.1: CONSENSUS

Input: Graph G, initial values x
Output: $x \leftarrow \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}x$

1 $\mathcal{T} \leftarrow$ a spanning tree of G
2 $d \leftarrow$ vector indexed by $V(\mathcal{T})$
3 **while** *T is not a single vertex* **do**
4 Initialize d to all 1's
5 Pick a node $v \in V(\mathcal{T})$ such that degree(v)=1
6 Designate v as the root of $\mathcal{T}$
7 DFS($\mathcal{T}, v, x, d$) `// See Algorithm 5.2`
8 $\mathcal{T} \leftarrow \mathcal{T} \backslash \{v\}$
9 **end**

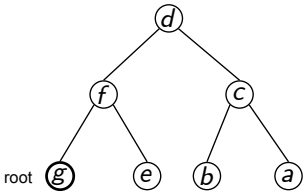
Algorithm 5.2: DFS

Input: Tree $\mathcal{T}$, vertex v,vectors x, d
Output: $x_v \leftarrow |\mathcal{T}|^{-1} \sum_{u \in \mathcal{T}} x_u$

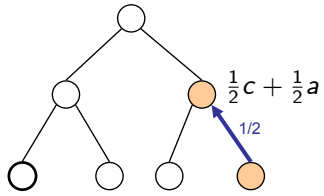
1 **if** *v has no children* **then**
2 **return**
3 **else**
4 **foreach** *child u of v* **do**
5 DFS($\mathcal{T}, u, x, d$)
6 $\begin{bmatrix} x_v \\ x_u \end{bmatrix} \leftarrow \begin{bmatrix} \frac{d_v}{d_v+d_u} & \frac{d_u}{d_v+d_u} \\ \frac{d_u}{d_v+d_u} & \frac{d_v}{d_v+d_u} \end{bmatrix} \begin{bmatrix} x_v \\ x_u \end{bmatrix}$
7 $d_v \leftarrow d_v + d_u$
8 **end**
9 **end**

To better understand Algorithm 5.2, we illustrate its steps on a complete binary tree in Figure 5.1. The letters inside the nodes of the figure denote their initial values. The blue arrows and weights denote the flow of values. For example, in Figure 5.1(c), the orange node is giving the yellow node 1/3 of its value. Colored nodes denote the active nodes in the algorithm. A yellow coloring means that the node value is the average of its subtree: that it contains the average of itself and all

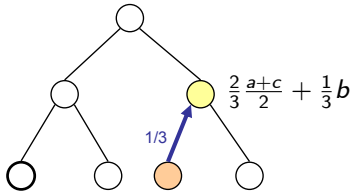
its descendants.



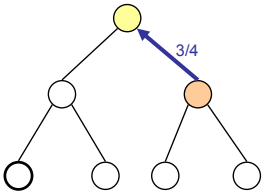
(a) Picking a leaf node as root



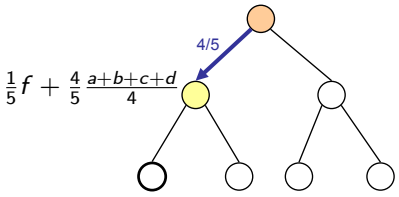
(b) Averaging toward the root



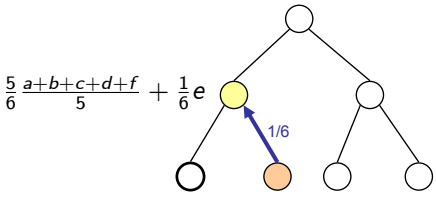
(c) Averaging toward the root



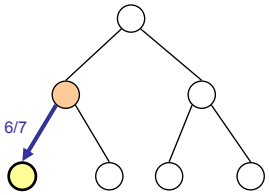
(d) Averaging toward the root



(e) Averaging toward the root



(f) Averaging toward the root



(g) Root node has the global average

Figure 5.1: Sample execution the Algorithm 5.2.

Algorithm 5.1 serves as a simple proof that $T^*_{G\cap S_1} < \infty$. However, it produces a length $O(n^2)$ factorization from $(G \cap S_1)$, which we know is not always optimal. The hypercube example from Section 4.3 demonstrated that some graphs admit a factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$ using only $O(n \log n)$ matrices from $(G \cap S_1)$. This motivates us to develop a better algorithm. But first, let's explore what basic matrix theory can tell us about consensus algorithms.

5.1 Matrix Insights

Many of our finite factorization results have been derived constructively from consensus algorithms. We now examine what basic matrix theory can tell us about the algorithmic structure. Let us consider factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}$ with $W(t) \in S_1 \cap G$. Except for matrices in $S_1' \cap S_1$, all of the matrices in S_1 are non-singular. Thus, for

$$\det \prod_{t=0}^{T-1} W(t) = \det \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}$$

we must have $W(t) \in S_1'$ for at least one t. In fact,

Theorem 5.1. *If a finite sequence of T matrices $W(0), \cdots, W(T-1)$ satisfy*

$$\prod_{t=0}^{T-1} W(t) = \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}$$

with $W(t) \in S_1 \cap G$, then, there exists a sequence of $n-1$ indices $\mathcal{I} = \{t_1, t_2, \cdots, t_{n-1}\} \subseteq \{0, 1, \cdots, T-1\}$ such that $W(t_i) \in S_1' \cap G$ for all $t_i \in \mathcal{I}$.

Proof. First notice that $\operatorname{rank} \mathrm{A} = \mathrm{n} - 1$ for $A \in S_1'$, $\operatorname{rank} \mathrm{B} = \mathrm{n}$ for $B \in S_1 \backslash S_1'$, and $\operatorname{rank} \mathbf{1}\mathbf{1}^{\mathsf{T}} = 1$. Since multiplication by a rank-$(n-1)$ matrix can decrease the rank of a matrix by at most one, we need $n-1$ such matrices to reach a rank of one. □

Corollary 5.2. *For any connected graph G with n vertices, $T^*_{G \cap S_1} = \Omega(n)$.*

Compared with the $\Omega(n \log n)$ bound derived using information theory, this lower bound based on elementary matrix theory is very loose. Nevertheless, it is interesting to note its ramifications on the structure of consensus algorithms.

5.2 Consensus Algorithms for Trees

The hypercube example of Section 4.3 taught us that certain graph structures allow for fast consensus. This is a good motivation for a fast consensus algorithm. That is, given a graph G, we shall look for certain subgraphs that allow for fast

finite-time average-consensus. Since the matrices in $(G \cap S_1)$ allow for "swaps" (*i.e.,* two neighbors in G can completely exchange their values), we will use swap operations to transfer the values to the "fast" parts of the graph for fast averaging. Let us first restrict our attention to graphs that are trees: for the remainder of this section, all graphs will be trees and G denotes a tree. For clarity and conciseness of presentation, we shall also assume that $|V(G)| = n = 2^k$.

Given a graph G with $n = 2^k$ vertices, define a sequence of $k-1$ graphs as follows:

$$G_i = G_{i+1} \cup G'_{i+1}, \quad G_0 \triangleq G,$$

where G_{i+1} is a connected subgraph of G_i with $|V(G_{i+1})| = \frac{|V(G_i)|}{2}$ and $G_i \cap G'_i = \varnothing$. Notice that $|V(G_i)| = 2^{k-i}$. Along with each partition, we define a bijection

$$g_i : V(G_i) \to V(G'_i).$$

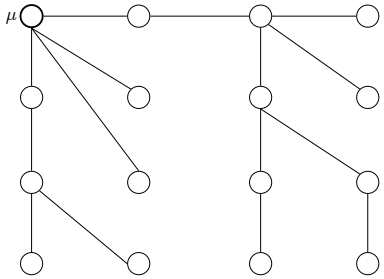
For suitably chosen partitions and bijections, a consensus algorithm on G can be described recursively as follows:

Algorithm 5.3: TREE-CONSENSUS

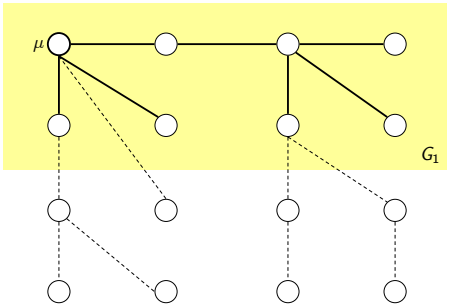
Input: Tree G_{i-1}, bijection g_i
Output: Finite-time average-consensus is achieved on G_{i-1}

if G_{i-1} *has only 2 nodes* **then**
    Perform pairwise averaging
    return
else
    TREE-CONSENSUS(G_i) `// Perform averaging on` G_i
    Swap values in G_i and G'_i
    `// This takes` $\sum_{v \in G_i} \Delta(v, g_i(v))$ `operations.`
    TREE-CONSENSUS(G_i)
    `// Perform averaging on` G_i `with swapped values`
    Swap values again while simultaneously averaging.
    `// This takes` $\sum_{v \in G_i} \Delta(v, g_i(v))$ `operations.`
end

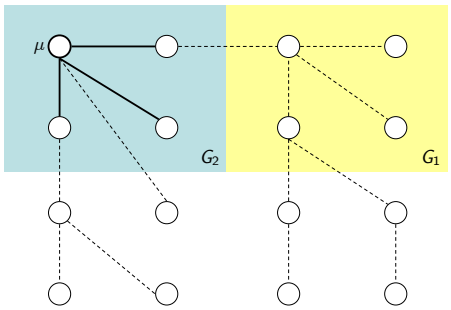
An illustration of the recursive partitioning process is shown in Figure 5.2.



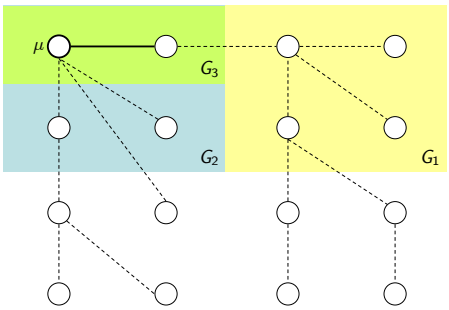
(a) A sample tree, G, with median μ.



(b) G partitioned into G_1 and G_1'. G_1 highlighted in yellow.



(c) G_1 partitioned into G_2 and G_2'. G_2 highlighted in blue.



(d) G_2 partitioned into G_3 and G_3'. G_3 highlighted in green.

Figure 5.2: Recursive partitioning process of Algorithm 5.3

The total number of operations in Algorithm 5.3 is:

$$\frac{n}{2} + \sum_{i=1}^{k-1} 2^i \sum_{v \in G_i} \Delta(v, g_i(v)).$$

The double summation only counts the number of swap operations (this includes the simultaneous swap-out and average). The $n/2$ term comes from the averaging operations on G_{k-1} for the $n/2$ pair of vertices that get swapped in. With suitable partitioning and bijections, we can get a desirable runtime (*i.e.,* a short factorization of $\frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$). Before proceeding, we need a couple definitions:

Definition 5.3. *For $a, b \in G$, let $a \rightsquigarrow b$ denote the path from a to b in G. For $c \in G$, define*

$$\Delta(a \rightsquigarrow b, c) = \min_{x \in a \rightsquigarrow b} \Delta(x, c).$$

That is $\Delta(a \rightsquigarrow b, c)$ is the distance of node c to the path $a \rightsquigarrow b$.

Lemma 5.4. *If $\Delta(\mu \rightsquigarrow v, u) \leq 1$ for all $u \in V(G_i)$ and $v = g_i(u)$, then*

$$\sum_{i=1}^{k-1} 2^i \sum_{v \in G_i} \Delta(v, g_i(v)) = O(n(\bar{D}_G + \log n)).$$

Proof. For $u \in V(G_i)$, $\Delta(\mu \rightsquigarrow g_i(u), u) \leq 1$ implies

$$\Delta(u, g_i(u)) \leq \Delta(\mu, g_i(u)) - \Delta(\mu, u) + 2.$$

Now we take the summation over G_i

$$\begin{aligned}
\sum_{u \in G_i} \Delta(u, g_i(u)) &\leq \sum_{u \in G_i} (\Delta(\mu, g_i(u)) - \Delta(\mu, u) + 2) \\
&= \sum_{u \in G_i} (\Delta(\mu, g_i(u)) + \Delta(\mu, u) - 2\Delta(\mu, u) + 2) \\
&= \sum_{v \in G_{i-1}} \Delta(\mu, v) - 2 \sum_{u \in G_i} \Delta(\mu, u) + 2\,|V(G_i)| \\
&= \sum_{v \in G_{i-1}} \Delta(\mu, v) - 2 \sum_{u \in G_i} \Delta(\mu, u) + 2^{k-i+1} \\
&= D_{i-1}(\mu) - 2D_i(\mu) + 2^{k-i+1},
\end{aligned}$$

where $D_i(\mu) = \sum_{u \in G_i} \Delta(\mu, u)$. Now,

$$\begin{aligned}
\sum_{i=1}^{k-1} 2^i \sum_{u \in G_i} \Delta(u, g_i(v)) &\leq \sum_{i=1}^{k-1} 2^i \left(D_{i-1}(\mu) - 2D_i(\mu) + 2^{k-i+1} \right) \\
&= \sum_{i=1}^{k-1} \left(2^i D_{i-1}(\mu) - 2^{i+1} D_i(\mu) + 2^{k+1} \right) \\
&= \sum_{i=0}^{k-2} 2^{i+1} D_i(\mu) - \sum_{i=1}^{k-1} 2^{i+1} D_i(\mu) + (k-1)2^{k+1} \\
&= 2D_0(\mu) - 2^k D_{k-1}(\mu) + (k-1)2^{k+1} \\
&= O(n \cdot \bar{D}_G) - O(n) + O(n \log n).
\end{aligned}$$

To see that $D_0(\mu) = O(n \cdot \bar{D}_G)$, observe that

$$\begin{aligned}
D_G^{\text{total}} &= \frac{1}{2} \sum_{u \in G} \sum_{v \in G} \Delta(u, v) \\
&\geq \frac{1}{2} \sum_{u \in G} \sum_{v \in G} \Delta(\mu, v) \\
&= \frac{n}{2} \sum_{v \in G} \Delta(\mu, v) = \frac{n}{2} D_0(\mu).
\end{aligned}$$

Now just look at the definition of average distance:

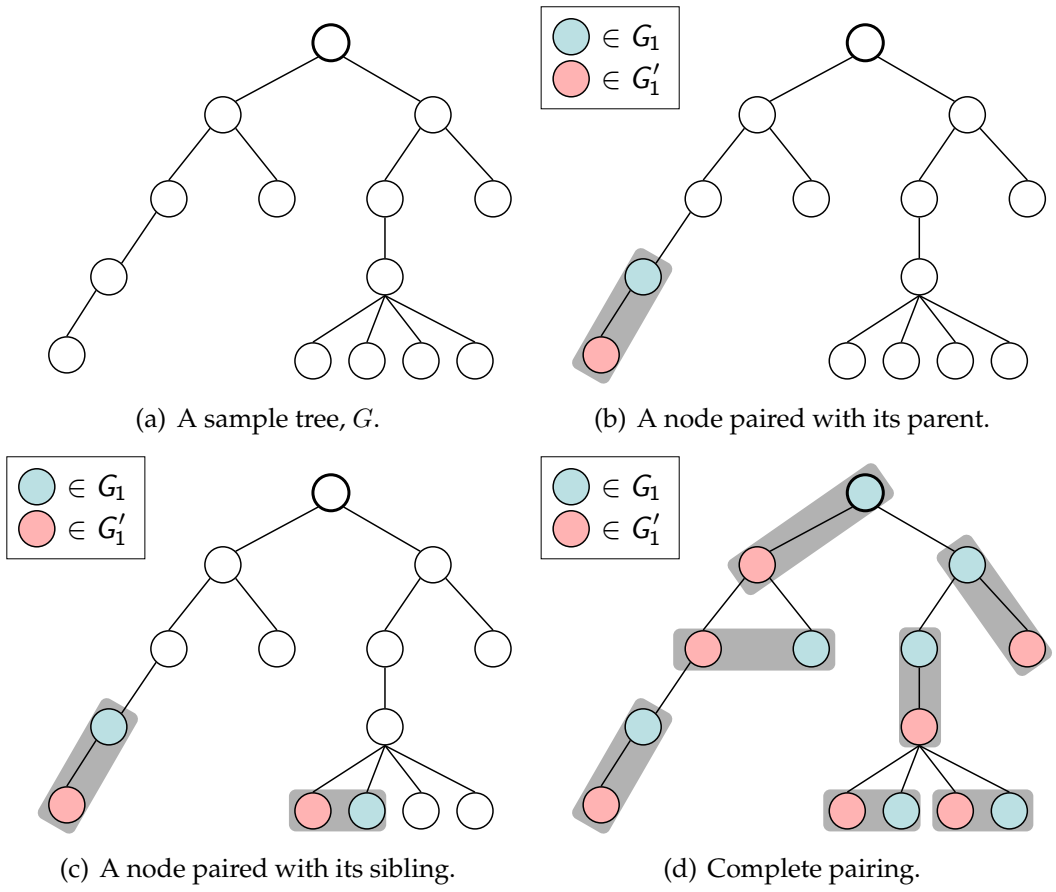
$$\bar{D}_G = \frac{D_G^{\text{total}}}{\binom{n}{2}} \geq \frac{D_0(\mu)}{n-1} \geq \frac{D_0(x)}{n}.$$

□

Using Algorithm 5.4, we show how to partition and pair the G_i-G_i' vertices to satisfy the condition in Lemma 5.4.

As an illustration, we demonstrate the pairing process on a sample tree in Figure 5.3.

Since each node is paired with its sibling or its parent, the output of Algorithm 5.4 easily satisfies Lemma 5.4 condition. Even though the pairings produced by Algorithm 5.4 promote efficient G_1-G_1' exchange, they cannot be used directly be-



(a) A sample tree, G.

(b) A node paired with its parent.

(c) A node paired with its sibling.

(d) Complete pairing.

Figure 5.3: Pair assignment process of Algorithm 5.4.

Algorithm 5.4: INITIAL PAIRING

Input: Tree G rooted at x
Output: A partition $G_1 \cup G_1' = G$ and mapping $g : G_1 \to G_1'$ satisfying Lemma 5.4

```
1  if |G| is odd then
2  |  Get rid of a leaf by DFS-like averaging.
3  end
4  i ← deepest level (descendants farthest from root x)  repeat
5  |  foreach unpaired node v at level i do
6  |  |  if v has an unpaired sibling, say u then
7  |  |  |  Pair (u, v) together.
8  |  |  |  The vertex closest to x is assigned to G_i and the other to G_i'
9  |  |  else if v has no unpaired sibling then
10 |  |  |  u ← Parent of v
11 |  |  |  Pair (u, v) together.
12 |  |  |  u is assigned to G_i and v to G_i'
13 |  |  end
14 |  end
15 |  Decrease level: i ← i − 1
16 until i = 0 or all nodes are paired
```

cause G_1 might be disconnected and averaging on G_1 may take a long time. We need to fix the pairings with Algorithm 5.5:

An illustration of the fixing process is shown in Figure 5.4.

To see the correctness of Algorithm 5.5, first note that:

Lemma 5.5. *Each node's membership in G_1 or G_1' is modified by the algorithm at most once.*

Proof. It is clear that each G_1' node on level i is enters the **foreach** loop exactly once: the algorithm changes the pairing so that v belongs in G_1.

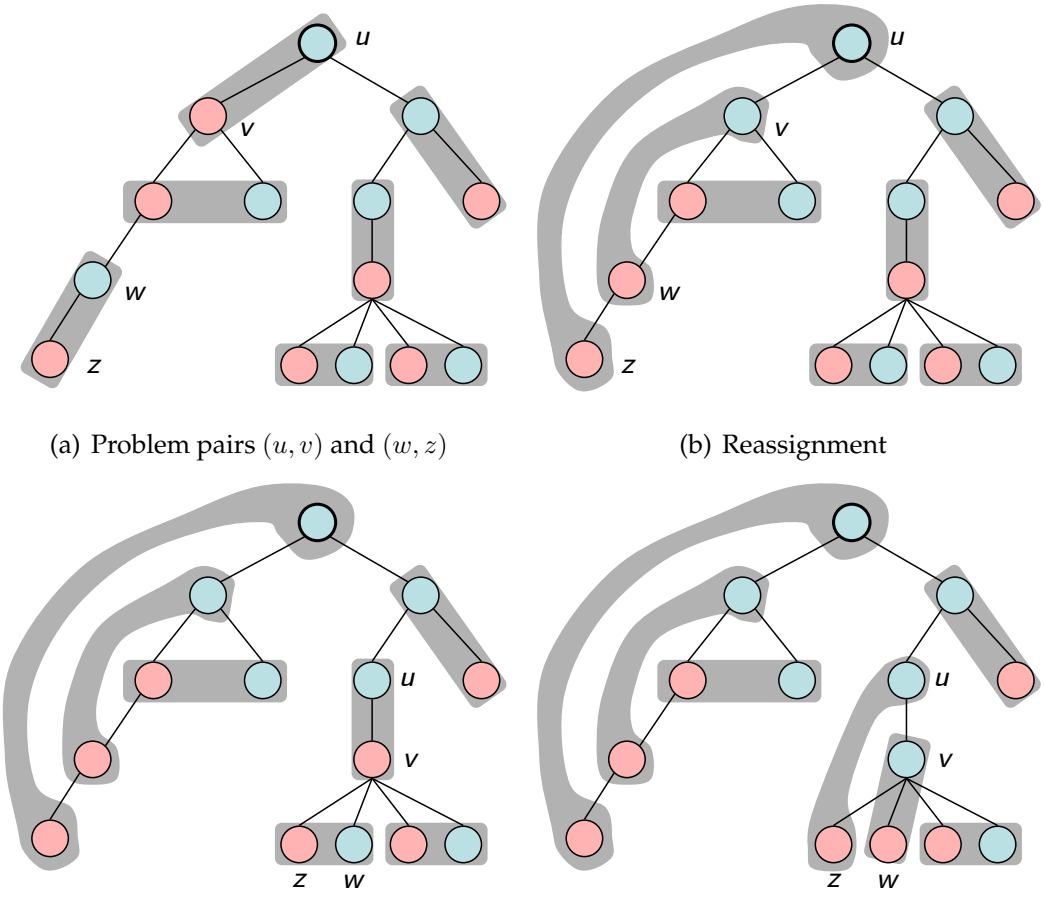
Now we consider a node in the tree that played the role of w (line 5) at some iteration of the algorithm. Line 7 of the algorithm would have changed w so it now belongs to G_1'. This means that w and all its descendants are in G_1' (since w was chosen as the farthest G_1 node before the membership change). So w will never enter the **foreach** loop again. □

In terms of problematic pairings addressed by Line 3 of Algorithm 5.5, there are only 4 cases to consider:

Algorithm 5.5: FIX PAIRING

Input: Tree G rooted at x with $|G|$=even
Output: A partition $G_1 \cup G_1' = G$ and mapping $g : G_1 \to G_1'$ satisfying Lemma 5.4 where G_1 is connected subgraph of G

$i \leftarrow 1$ (start with nodes in G_1' that are closest to x)
repeat
  foreach *node* $v \in G_1'$ *on level* i *that has a descendent* $w \in G_1$ **do**
    $u \leftarrow g^{-1}(v)$
    $w \leftarrow$ farthest descendent $\in G_1$.
    $z \leftarrow g(w)$
    Pair u with w, put w in G_1' (Remove w from G_1).
    Pair v with z, put v in G_1 (Remove v from G_1').
  end
  Increase level: $i \leftarrow i + 1$
until *(i = depth of G) or (G_1 is connected)*



(a) Problem pairs (u, v) and (w, z)

(b) Reassignment

(c) Problem pairs (u, v) and (w, z)

(d) Fixing complete. Blue nodes (*i.e.,* nodes in G_1) are now connected.

Figure 5.4: Pair fixing process of Algorithm 5.5.

- u-v, w-z (see Figure 5.5(a))
- u-v, $w \wedge z$ (see Figure 5.5(b))
- $u \wedge v$, w-z (see Figure 5.5(c))
- $u \wedge v$, $w \wedge z$ (see Figure 5.5(d)),

where u-v denotes that nodes u and v are neighbors (more precisely, they have a parent-child relationship); $u \wedge v$ denotes that nodes u and v are siblings that share the same parent node. These cases are illustrated in Figure 5.5. It is easy to see that each of these four cases are correctly handled by lines 7 and 8 of Algorithm 5.5 thus:

Theorem 5.6 (Upper bound). *For any connected graph G with n vertices, $T^*_{G \cap S_1} = O(n(\bar{D}_G + \log n))$.*

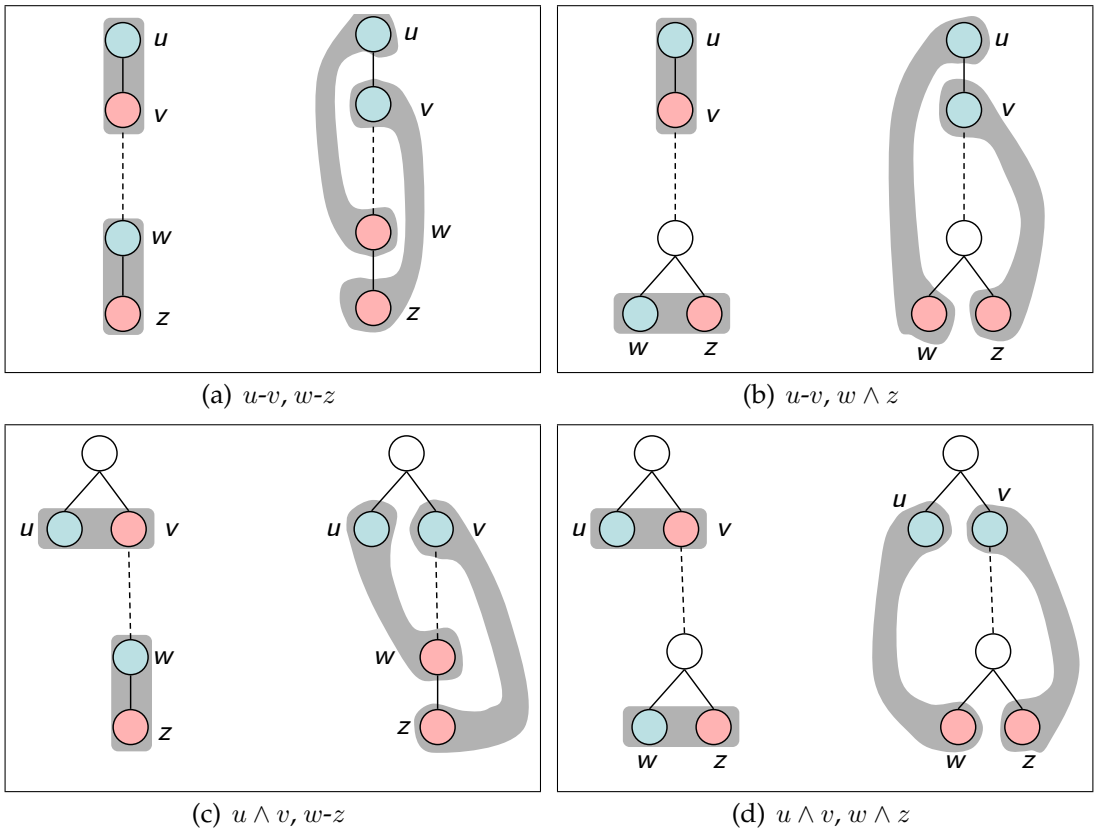


Figure 5.5: Four problematic pairing configurations and their fixes.

5.3 Lower bound on Trees

To see that the algorithms presented in the previous section are optimal, we now derive lower bounds on the consensus time.

Definition 5.7. *Given a tree G and an edge $e = (l, r)$ in G, define*

$$m(e) \triangleq \min\{|L_e|, |R_e|\},$$

where

$$L_e \triangleq \{v \in G : \Delta(v, l) < \Delta(v, r)\}, \quad R_e \triangleq \{v \in G : \Delta(v, l) > \Delta(v, r)\}.$$

In other words, $m(e)$ is the minimum number of nodes on the "left" and "right" side of edge e.

Lemma 5.8. *For any tree G on n vertices,*

$$T^*_{G\cap S_1} = \Omega\left(\sum_{e\in E(G)} m(e)\right).$$

Proof. Given an edge $e = (l, r)$ in G, consider the initial condition

$$x_i(0) = \begin{cases} 0 & \text{if } i \in L_e\} \\ 1 & \text{if } i \in R_e\}. \end{cases}$$

The total mass is $|R_e|$ and average-consensus is achieved when $x(T) = \frac{|R_e|}{n}\mathbf{1}$ for some T. In this proof, it is useful to view each matrix in $G \cap S_1$ as a "use" of a particular edge in $E(G)$. Using a mass-balancing flow argument, we show that $m(e)/2$ is a lower bound on the number of times edge e must be used.

In aggregate, the nodes in L_e require $|L_e|/n$ fraction of the total mass. Since the total mass is $|R_e|$ and each use of an edge carries at most 1 unit of mass, we know that edge e must be used at least

$$\left\lceil \frac{|L_e||R_e|}{n} \right\rceil \geq \frac{|L_e||R_e|}{n} = \frac{m(e)(n - m(e))}{n} \geq \frac{m(e)}{2}$$

times. Since a consensus algorithm must achieve average-consensus for all initial distributions, we must have

$$T^*_{G\cap S_1} \geq \sum_{e\in E(G)} \frac{m(e)}{2}$$

as a lower bound on its consensus time.

To see that each edge can carry a flow of at most 1, observe that matrices in S_1 correspond to a convex combination of a pair of node values. Since initial values are $x_i(0) \in \{0,1\}$, any sequence of convex combinations must keep the values in the closed interval $[0,1]$, *i.e.*, $0 \leq x_i(t) \leq 1$ for all t. □

One can relate this lower bound to the graph average distance by

Lemma 5.9.

$$\sum_{e\in E(G)} m(e) \leq \bar{D}_G \cdot (n-1) \leq 2 \sum_{e\in E(G)} m(e).$$

Proof.

$$\begin{aligned} D_G^{\text{total}} &= \sum_{i\in V} \sum_{j\in V, j\neq i} \Delta(i,j) \\ &= \sum_{e\in E} |L_e| \cdot |R_e| \\ &= \sum_{e\in E} m(e) \cdot (n - m(e)). \end{aligned}$$

For the "$\leq$", notice that

$$\sum_{e\in E} m(e)\cdot(n-m(e)) = n\sum_{e\in E} m(e) - \sum_{e\in E} m^2(e) \leq n\sum_{e\in E} m(e).$$

Since total distance is just average distance times $\binom{n}{2}$, we have

$$\begin{aligned} \bar{D}_G \cdot \frac{n(n-1)}{2} &\leq n\sum_{e\in E} m(e) \\ \bar{D}_G \cdot \frac{(n-1)}{2} &\leq \sum_{e\in E} m(e). \end{aligned}$$

For the "$\geq$", we note that $m(e) \leq n/2$ so that $n - m(e) \geq n/2$.

$$\sum_{e \in E} m(e) \cdot (n - m(e)) \geq \frac{n}{2} \sum_{e \in E} m(e)$$

which means

$$\begin{aligned} \bar{D}_G \cdot \frac{n(n-1)}{2} &\geq \frac{n}{2} \sum_{e \in E} m(e) \\ \bar{D}_G \cdot (n-1) &\geq \sum_{e \in E} m(e). \end{aligned}$$

□

Thus, we can restate Lemma as,

Lemma 5.10. *For any connected graph G with n vertices, $T^*_{G \cap S_1} = \Omega\left(n \cdot \text{avg dist}\right)$.*

This lower bound is based on the structure of the underlying graph. We can use the same argument as in Lemma 4.3 to obtain an information-theoretic lower bound:

Lemma 5.11. *For any connected graph G with n vertices, $T^*_{G \cap S_1} = \Omega\left(n \log n\right)$.*

Combining Lemma 5.10 and Lemma 5.11, we have

Theorem 5.12 (Lower bound)**.** *For any connected graph G with n vertices, $T^*_{G \cap S_1} = \Omega\left(n(\bar{D}_G + \log n)\right)$.*

5.4 Consensus Time on Trees

The lower bound of Theorem 5.12 matches the complexity of the recursive finite-time average-consensus algorithm developed in Section 5.2 so the problem of scheduling for finite-time average-consensus on trees subject to the constraints of S_1 is solved:

Theorem 5.13.

$$T^*_{G \cap S_1} = \Theta\left(n(\bar{D}_G + \log n)\right)$$

5.5 General Graphs

For general graphs that are not trees, the upper bound from Section 5.2 still holds. That is, given a graph G, we can always construct a spanning tree and run the tree-based algorithm of Section 5.2. The question remains on how much do we lose by considering only a spanning tree and not the entire graph. In general, graphs can have very rich structure and high connectivity compared to its spanning tree. Surprisingly, we only lose a $\log^2 n$ factor when considering only the spanning tree. As a caveat, one needs to be careful when choosing the spanning tree as certain trees (such as the path on n vertices) do not admit fast consensus. We need to choose a spanning tree that preserves the average distance (up to a constant).

5.5.1 Graph Lower bound

Recall the lower bound technique for trees used in Lemma 5.3 of Section 5.3 where we picked edges e from the graph G and argued that each edge must support of flow of $m(e)$ units. In trees, edges are precisely the bottlenecks because removing an edge disconnects the tree into two trees. In graphs, however, there may be many paths between a given pair of nodes. Removing an edge may not be enough to disconnect the graph so if we want to proceed with the same mass flow argument, we need to consider graph cuts.

Definition 5.14. *A cut $C = (S, T)$ of a graph $G = (V, E)$ is a partition of the vertex set V where*

$$S \cup T = V, \quad S \cap T = \emptyset.$$

The cut-set of a cut C is the set

$$\{(u, v) \in E : u \in S, v \in T.$$

We say two cuts are edge-disjoint *if their cut-sets do not share any edges.*

Given a cut $C = (S, T)$, we can define

$$m(C) = \min\{|S|, |T|\}$$

to refer to the smaller side of the cut. That is, the side with the fewest vertices. If we have a sequence $C_1, C_2, \cdots, C_k$ of edge-disjoint cuts, then a lower bound on $T^*_{G \cap S_1}$ is

$$T^*_{G \cap S_1} = \Omega\left(\sum_{i=1}^{k} m(C_i)\right)$$

by using a similar mass flow argument as in Lemma 5.3.

It is clear that if we can find a suitable set of edge disjoint cuts, then we have a good lower bound. The trouble is that collections of good (*i.e.,* maximal) edge-disjoint cuts are hard to find. If we are careful in accounting the amount of goods that flow across each edge, then we don't need edge-disjoint cuts and we can use the solution to the following integer program as a lowerbound:

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}^{\mathsf{T}} x \\ \text{s.t.} \quad & A\,x \geq m \\ & 0 \leq x_e < n, \forall e \in E \\ & x_e \in \mathbb{Z}, \forall e \in E. \end{aligned}$$

Here, $x \in \mathbb{R}^{|E|}$ is a vector indexed by the edges of the graph. For $e \in E$, x_e indicates how many times edge e is used in the consensus protocol. The matrix A is a cut-edge incidence matrix of G. It is a "tall" matrix of size $O(2^n) \times |E|$. The rows of A are indexed by cuts of G and columns of A are indexed by edges of G. Given a cut C of G and an edge $e \in E$, $A_{C,e} = 1$ if e is in the cut-set of C and 0 otherwise. The vector m is a vector indexed by the cuts of G; it is of size $O(2^n)$. Given a cut C, $m_C = m(C)$. The program solves for the minimum number of edge uses across all edges subject to a mass flow constraint.

Although the solution to the program is a valid upper bound, integer programs are generally hard to solve. Even if we relax the integer constraint and turn it into a linear program, we still have to deal with the exponential number of constraints in the program. This motivates us to seek alternative methods of lower bounding the consensus-time on general graphs.

5.6 Metric Space Embeddings

The key idea in our lower bound techniques was our search for graph cuts which are representative of the bottlenecks in the graph. If we can find a maximal set of edge-disjoint paths, then we can get a "good" lower bound. As maximal edge-disjoint paths can be hard to find, we use some tools from metric space embedding to map our graph into a different space where the edge-disjoint cuts are easier to find. This allows us to find a "good-enough" collection of edge-disjoint cuts that provide a "good-enough" lower bound. To be more precise, we need to introduce some tools from metric space embedding:

Theorem 5.15. *[Abraham, Bartal, Neiman [1]] For any $1 \leq p \leq \infty$, every n-point metric space embeds in L_p with distortion $O(\log n)$ in dimension $O(\log n)$.*

The theorem implies that there exists a mapping $f : V(G) \to \mathbb{R}^{O(\log n)}$ with distortion

$$\Delta(u,v) \leq \|f(u) - f(v)\| \leq O(\log n) \cdot \Delta(u,v), \quad \forall u, v \in G, \tag{5.1}$$

where $\|f(u)\| = \sum_j |f_j(u)|$ denotes the L_1-norm. Let i denote the "heaviest" coordinate in terms of total pairwise distance:

$$i = \arg\max_j \sum_{u,v \in G : u \neq v} |f_j(u) - f_j(v)|.$$

This means that

$$\sum_{u \neq v} \frac{\|f(u) - f(v)\|}{O(\log n)} \leq \sum_{u \neq v} |f_i(u) - f_i(v)| \tag{5.2}$$

because

$$\sum_{u \neq v} \|f(u) - f(v)\| \;=\; \sum_{u \neq v} \sum_{j} |f_j(u) - f_j(v)| \leq O(\log n) \sum_{u \neq v} |f_i(u) - f_i(v)|.$$

We also know that

$$|f_i(u) - f_i(v)| \leq \|f(u) - f(v)\|, \quad \forall u, v \in G. \tag{5.3}$$

Combining the first inequality of (5.1) with (5.2), we see that

$$\sum_{u \neq v} \frac{\Delta(u,v)}{O(\log n)} \leq \sum_{u \neq v} |f_i(u) - f_i(v)|. \tag{5.4}$$

Combining the second inequality of (5.1) with (5.3), we see that

$$|f_i(u) - f_i(v)| \leq O(\log n)\, \Delta(u,v), \quad \forall u, v \in G. \tag{5.5}$$

Thus, on average, edges in G are "stretched" by at most $O(\log n)$ under f_i. This property allows us to use f_i to embed the graph G onto $\mathbb{R}$ and look for edge-disjoint cuts on the embedded line. Starting from the left-most point (*i.e.,* $\min_u f_i(u)$), consider partitioning the embedded points on $\mathbb{R}$ using a sequence of cuts spaced $O(\log n)$ apart. These partitions correspond to a sequence of edge-disjoint cuts $\{C_k\}$ in G: if not, then some edge in G must have been stretched longer than $O(\log n)$ by f_i which contradicts (5.5).

Pick a point x that has $n/2$ points to its left: $|\{v : f_i(v) \leq f_i(x)\}| = n/2$. We have

$$\begin{aligned}
\sum_k m(C_k) &\geq \sum_{v \in G} \left\lfloor \frac{|f_i(v) - f_i(x)|}{O(\log n)} \right\rfloor && (5.6)\\
&\geq \sum_{v \in G} \left(\frac{|f_i(v) - f_i(x)|}{O(\log n)} - 1 \right) \\
&= \sum_{v \in G} \left(\frac{|f_i(v) - f_i(x)|}{O(\log n)} \right) - n \\
&\geq \sum_{u \neq v} \left(\frac{|f_i(u) - f_i(v)|}{n\, O(\log n)} \right) - n && (5.7)\\
&\geq \sum_{u \neq v} \left(\frac{\Delta(u,v)}{n\, O(\log^2 n)} \right) - n && (5.8)\\
&= \Omega\left(\frac{n \cdot \bar{D}_G}{\log^2 n} \right), && (5.9)
\end{aligned}$$

where inequality (5.6) is because in the summation, each point v contributes its distance to x divided by $O(\log n)$; inequality (5.7) is because

$$\sum_{u \neq v} |f_i(u) - f_i(v)| \leq \sum_{u \neq v} |f_i(u) - f_i(x)| + |f_i(v) - f_i(x)| = 2(n-1) \sum_{v \in G} |f_i(v) - f_i(x)|;$$

inequality (5.8) is because of (5.4). The final inequality follows from the definition of $\bar{D}_G$.

Since $T^*_{G \cap S_1} = \Omega(\sum_k m(C_k))$ for any sequence $\{C_k\}$ of edge disjoint cuts and the information-theoretic lower bound of $\Omega(n \log n)$ remains valid for all graphs, we conclude that

Theorem 5.16. *For any graph G,*

$$T^*_{G \cap S_1} = \Omega\left(n \left(\frac{\bar{D}_G}{\log^2 n} + \log n \right) \right).$$

Interpreting this result in light of the tree-based consensus algorithm in Section 5.2, we see that taking into account all edges of the graph offer at most an $O(\log^2 n)$ speedup over our spanning tree-based algorithm. This is quite surprising as the number of edges between a graph and its spanning tree can differ by as much as a factor $O(n)$.

Chapter 6

Factorization under Additional Constraints - Parallel Symmetric Weighted Averages

Instead of allowing only one pair of neighbors to exchange values at any given time, we now explore the possibility of parallel communications. Consider the set

$$S \triangleq \left\{ W \in \mathbb{Q}^{n \times n} : W = W^{\mathsf{T}} \right\}.$$

Note that the matrices in S are doubly stochastic (*i.e.*, $\mathbf{1}^{\mathsf{T}} W = \mathbf{1}^{\mathsf{T}}$ and $W\mathbf{1} = \mathbf{1}$). The motivation for S is to allow parallel distribution of mass by symmetric weighted averages; yet disallow drastic aggregation steps such as line 6 of Algorithm 3.1 where nodes transfer all their mass to another. Such operations are often undesirable due to trust considerations: why should one node send everything to another node in hopes of getting his/her fair share in the future. Furthermore, node capacity constraints often disallow such aggregation by any individual node. In the example of Section 1.2, each child has a backpack of unit capacity. Therefore, they are unable to store goods in excess of one unit. We want to consider the possibility of parallel communications to investigate the speed up between parallel communication and serial communication in the context of finite-time average-consensus problems. We will see shortly that by using $G \cap S$ instead of $G \cap S_1$, the consensus time is improved to $\Theta(n)$.

6.1 Upper bound: $T^*_{G\cap S} = O(n)$

As $S_1 \subset S$, we can still use Algorithm 5.1 to achieve consensus. But instead using the reverse depth-first traversal in Algorithm 5.2, we modify it slightly (see Algorithm 6.1) to use fewer matrices (*i.e.,* Algorithm 5.1 using the improved DFS of Algorithm 6.1 is faster than Algorithm 5.1 using the DFS of Algorithm 5.2). Algorithm 5.1 with improved DFS (Algorithm 6.1) can be further improved by using a pipelined architecture to yield Algorithm 6.2 which allows factoring using only $O(n)$ matrices.

Algorithm 6.1: DFS-IMPROVED

Input: Tree T, vertex v,vectors x, d
Output: $x_v \leftarrow |T|^{-1} \sum_{u\in T} x_u$

1 **if** v *has no children* **then**
2 | **return**
3 **else**
4 | **foreach** *child* u *of* v **do**
5 | | DFS-IMPROVED(T, u, x, d)
6 | **end**
7 | $\{u_1, \cdots, u_\ell\} \leftarrow$ set of children of v
8 | $D \leftarrow d_v + \sum_{j=1}^{\ell} d_{u_j}$
9 | $\begin{bmatrix} x_v \\ x_{u_1} \\ x_{u_2} \\ \vdots \\ x_{u_\ell} \end{bmatrix} \leftarrow \begin{bmatrix} \frac{d_v}{D} & \frac{d_{u_1}}{D} & \frac{d_{u_2}}{D} & \cdots & \frac{d_{u_\ell}}{D} \\ \frac{d_{u_1}}{D} & \frac{D-d_{u_1}}{D} & 0 & \cdots & 0 \\ \frac{d_{u_2}}{D} & 0 & \frac{D-d_{u_2}}{D} & & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ \frac{d_{u_\ell}}{D} & 0 & \cdots & 0 & \frac{D-d_{u_\ell}}{D} \end{bmatrix} \begin{bmatrix} x_v \\ x_{u_1} \\ x_{u_2} \\ \vdots \\ x_{u_\ell} \end{bmatrix}$
10 | $d_v \leftarrow D$
11 **end**

Intuitively speaking, Algorithm 6.2 implements a pipelined version of Algorithm 5.1. We employ parallel consensus steps when they do not interfere with each other. For clarity, we use a simple example to illustrate the pipelined algorithm. Consider G as the path with $V = \{1, 2, 3, 4, 5\}$ and $E = \{a, b, c, d\}$ as shown in Figure 6.1.

First, let us run Algorithm 5.1 on our simple example. Suppose that line 5 of

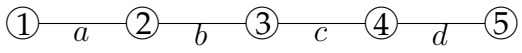


Figure 6.1: A line graph.

Algorithm 5.1 examines nodes $5, 4, 3, 2, 1$ in that order, then the sequence of pairwise weighted averages corresponds to the following sequence of edges:

	a	b	c	d						⑤
					a	b	c			④
								a	b	③
									a	①&②
Time:	1	2	3	4	5	6	7	8	9	10

Here, time runs left-to-right and each time column enumerates all edges used during that time slot. With Algorithm 5.1, each time slot only utilizes one edge and we need 10 edges. The right-most annotation indicates that the sequence of edges allowed a node to obtain the correct average. For example, after the edge sequence a, b, c, d in time steps 1-4, node ⑤ will have the correct global average. Since edges a and c are vertex-disjoint, the averaging on a will not effect the values of nodes in edge c. We can thus perform some averages in parallel and implement a pipelined architecture:

	a	b	c	d				⑤
			a	b	c			④
					a	b		③
							a	①&②
Time:	1	2	3	4	5	6	7	

Pipelining allows us to use multiple edges per time step (*e.g.,* At time 4, edges b and d are used in parallel. At time 5, edges c and a are used in parallel). The parallel edges can be incorporated into a single matrix leading to savings in terms of the number of matrices used. Once again, the right-most labels annotate the epochs

dedicated to each node obtaining the global average (*e.g.*, after edge sequence a, b, c in time 3-5, node ④ obtains the global average). This pipelined architecture is the key innovation of Algorithm 6.2.

Algorithm 6.2: CONSENSUS

Input: Graph G, initial values x
Output: $x \leftarrow \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}x$

$T \leftarrow$ a spanning tree of G
$r_1, r_2, \cdots, r_n \leftarrow$ a postordering of $V(T)$ by depth-first search
$d \leftarrow$ vector of 1's indexed by $V(T)$
$\phi \leftarrow$ vector of 0's indexed by $V(T)$
foreach $v \in V(T)$ **do**
  if degree(v)=1 **then** $\phi_v = 1$
end
$i \leftarrow 1$
Let r_i be the root of T
$\phi_{r_i} = 0$
while $|V(T)| > 1$ **do**
  Consensus-While Loop (Algorithm 6.3)
  if $\phi_{r_i} = 1$ **then**
    let $r_i \sim u_1 \sim \cdots \sim u_m \sim r_{i+1}$ be the path from r_i to r_{i+1} in T
    foreach $0 < j \leq m$ **do**
      $d_{u_j} \leftarrow 1$
      $\phi_{u_j} \leftarrow 0$
    end
    if degree(u_1)=2 **then**
      $\phi_{u_1} \leftarrow 1$
    end
    $T \leftarrow T \backslash \{r_i\}$
    $i \leftarrow i + 1$
    Let r_i be the root of T
    $\phi_{r_i} = 0$
  end
end

Now we examine the inner workings of Algorithm 6.2. We begin by establishing a postordering $(r_1, r_2, \cdots, r_n)$ of vertices by a depth-first search from an arbitrary vertex. During the execution process, we keep track of ϕ, an indicator of

Algorithm 6.3: CONSENSUS-WHILE LOOP

$d' \leftarrow d$;
$\phi' \leftarrow \phi$;
foreach $v \in V(T)$ **do**
  if $(\phi_v = 0)$ *and* $(\forall \,\text{child}\, u \,\text{of v}, \phi_{\text{u}} = 1)$ **then**
    $\{u_1, \cdots, u_\ell\} \leftarrow$ children of v;
    $D \leftarrow d_v + \sum_{j=1}^{\ell} d_{u_j}$;
    $$\begin{bmatrix} x_v \\ x_{u_1} \\ \vdots \\ x_{u_\ell} \end{bmatrix} \leftarrow \begin{bmatrix} \frac{d_v}{D} & \frac{d_{u_1}}{D} & \cdots & \frac{d_{u_\ell}}{D} \\ \frac{d_{u_1}}{D} & \frac{D-d_{u_1}}{D} & & 0 \\ \vdots & & \ddots & \\ \frac{d_{u_\ell}}{D} & 0 & & \frac{D-d_{u_\ell}}{D} \end{bmatrix} \begin{bmatrix} x_v \\ x_{u_1} \\ \vdots \\ x_{u_\ell} \end{bmatrix};$$
    $d'_v \leftarrow D$;
    $\phi'_v \leftarrow 1$;
    foreach *child* u *of* v **do**
      if $d_u \neq 1$ **then**
        $d'_u \leftarrow 1$;
        $\phi'_u \leftarrow 0$;
      end
    end
  end
end
$d \leftarrow d'$;
$\phi \leftarrow \phi'$;

whether a vertex's value is the average of its descendants:

$$\phi_v = \begin{cases} 1 & \text{if } x_v = \frac{1}{|\text{decedents(v)}|+1}\left(x_v + \sum_{u \in \text{decedents(v)}} x_u\right) \\ 0 & \text{otherwise.} \end{cases}$$

The computations of x-updates in each **while** loop (line 11 of Algorithm 6.2) can be translated into a single matrix in $G \cap S$ as each vertex appears at most once in line 7 of Algorithm 6.3 for each iteration of the **while** loop (*i.e.,* all operations inside line 3's **foreach** loop of Algorithm 6.3 can be combined into one matrix). The number of *while* iterations needed for r_1 to achieve the average is at most $n-1$. After r_i reaches the average, the number of *while* loops needed for r_{i+1} to achieve the average is at most the length of the path from r_i to r_{i+1}. Since the sequence $r_1, r_2, \cdots, r_n$ is a postordering of $V(T)$ by depth-first search,

$$\sum_{i=1}^{n-1} (\text{length of path from } r_i \text{ to } r_{i+1}) \leq 2n.$$

Therefore, the number of matrices in this factorization is $O(n)$.

For a rigorous argument of the correctness and consensus time of Algorithm 6.2, we rewrite it as more abstract pseudo code in Algorithm 6.4. To prove the correctness of Algorithm 6.4, we need definitions of "ready" nodes and "progress" nodes:

Definition 6.1. *Node v is a* progress node *if it is ready* (i.e., $\phi_v = 1$) *and the path from root to v contains no other ready nodes.*

We will show, through a series of arguments, that there is a wave of progress nodes moving toward each root node and that the total time for all waves to complete is $O(n)$.

Lemma 6.2. *Let δ_i denote the maximal distance from root r_i to a farthest progress node. After the "**foreach**" loop, δ_i decreases by 1.*

Proof. We shall proceed inductively. Initially, before the first execution of the loop, all progress nodes are leaves. Let S denote the set of leaf-nodes whose distances

Algorithm 6.4: PIPELINED

```
1  r_1, r_2, ··· , r_n is a postodering by depth-first search.
2  Start with r_1 as the root
3  repeat
4      foreach node v whose children are ready do
5          Average v with its children
6          Set φ_v = 1 (v is ready)
7          (book-keeping) reset all non-leaf children to not ready
8      end
9      (*)
10     if Root r_i is ready (i.e., φ_{r_i} = 1) then
11         Reset nodes along path r_i ⇝ r_{i+1}
12         Remove old root r_i
13         (book-keeping) if node connected to r_i becomes leaf then set it to
           ready
14         Make r_{i+1} the new root
15         (**)
16     end
17 until no more nodes left
```

from the root are maximal. The parents of nodes in S will become progress nodes after (*). Assume the contrary: that there exists a $v \in S$ who has an unready sibling $u \notin S$, $\phi_u = 0$. Since leaves are always ready, u cannot be a leaf. If u is not a leaf, it must have a descendant w that is a leaf and $\phi_w = 1$. The distance from the root to w is strictly greater than the distance from root to v, contradicting the maximality of v. Thus, no such siblings exist for all nodes in S. Parents of nodes in S will become progress nodes at (*) and δ_i decreases by 1.

Notice that the advancement of progress nodes depends only on other progress nodes. So we can discard all descendants of progress nodes from our consideration. Effectively, all progress nodes can be seen as "leaf-nodes" and the previous argument applies for each iteration. □

Definition 6.3. *A node v is* good *if*

1. *v is a leaf, or*
2. *v is not ready but all its children are ready and good, or*
3. *v is ready and all its children are good.*

Lemma 6.4. *Once a node becomes good, it will never become non-good.*

Proof. We will use backward induction on the distance from the root to the node. The base cases are the leaf nodes, which are always good by definition. Assume the claim is true for all nodes at a distance t from the root. Now consider a good node v at distance $t-1$ from the root. We show that after the iteration of the algorithm, v remains good. There are three cases to consider:

1. (v is a leaf) A leaf is always good and ready.
2. (v is not ready but all its children are ready and good) After one iteration of the algorithm, v becomes ready and all its children remain good by induction hypothesis.
3. (v is ready and all its children are good) There are two subcases to consider:
   - (After an iteration, v remains ready) v remains good since all its children remain good by induction hypothesis.
   - (After an iteration, v becomes not ready) We need to show that v's children are ready and good. Since v's children were good by assumption, we shall show that they are ready. Consider the children of v that are not ready (if there are no such children then we are done). These children are good by induction hypothesis and therefore their children (i.e. v's grandchildren) must be ready. Thus after an iteration, the unready children of v will become ready.

□

Lemma 6.5. *At (**), all nodes at distance 1 away from $r_i \rightsquigarrow r_{i+1}$ path are good.*

Proof. By the time r_i becomes ready, all remaining nodes will be good. The bookkeeping resets only affect the nodes on the path $r_i \rightsquigarrow r_{i+1}$ so the nodes attached to this path (at distance 1 away) are good. □

Lemma 6.6. *At (**), $\delta_{i+1} \leq \Delta(r_i, r_{i+1})$.*

Proof. First, observe that a good node that is not ready will become ready within one iteration. Thus, within one iteration, all nodes attached to the $r_i \rightsquigarrow r_{i+1}$ will become ready.

Let $r_i \to u_1 \to u_2 \to \cdots u_m \to r_{i+1}$ be the path from r_i to r_{i+1}. There are two cases to analyze:

- (After removing r_i, u_1 becomes a leaf) Since u_1 became a leaf, it will be set as ready. Let's consider some possible cases for farthest progress nodes:
  - (u_1 or children of u_2 that are not on the $r_i \rightsquigarrow r_{i+1}$ path) Distances from r_{i+1} to these nodes are strictly less than $\Delta(r_i, r_{i+1})$.
  - (grand children of u_2 that are not on the $r_i \rightsquigarrow r_{i+1}$ path) Distances from r_{i+1} to these nodes are equal to $\Delta(r_i, r_{i+1})$.

  Since all nodes attached to the $r_i \rightsquigarrow r_{i+1}$ path are good, these are the only candidates for farthest progress nodes.

- (After removing r_i, u_1 is not a leaf) Let's consider the state of things at (*) when $\phi_{u_1} = 1$. The process of making ϕ_{r_i} ready will have made $\phi_{u_1} = 0$. But since u_1 remains good, we know that the children of u_1 are all ready. With respect to the new root r_{i+1}, these children u_1 are progress nodes at a maximal distance away. This maximal distance is the $\Delta(r_i, r_{i+1})$.

□

Theorem 6.7. *[Upper bound] For any connected graph G with n vertices,*

$$T^*_{G \cap S} = O(n).$$

Proof. The time it takes to complete r_1 is at most n. The time it takes to complete r_{i+1} is $\delta_{i+1} \leq \Delta(r_i, r_{i+1})$. So time to total completion is at most

$$n + \sum_{i=2}^{n} \Delta(r_i, r_{i+1}) \leq n + 2(n-1) \leq 3n.$$

Because the r_i's are a postordering by depth-first search, the summation $\sum_{i=2}^{n} \Delta(r_i, r_{i+1})$ is analogous to tracing an outline around the entire tree. A tree on n vertices has $n-1$ edges, an outline of the tree traces each edge twice so that's where the $2(n-1)$ term comes in. □

6.2 Lower bound: $T^*_{G \cap S} = \Omega(n)$

To show that the lower bound of $T^*_{G \cap S}$ is $\Omega(n)$, we use a similar mass flow argument as Lemma 5.3 in Section 5.3. To make the argument, we need to first identify some bottle-neck nodes in our tree:

Lemma 6.8. *Given a tree $\mathcal{T}$, there exists a node v and corresponding sets A and B such that*

1. $\mathcal{T} \backslash \{v\} = A \cup B$ *with* $A \cap B = \varnothing$,
2. $v \in a \rightsquigarrow b$ *for all* $a \in A$ *and* $b \in B$,
3. $\min\{|A|, |B|\} \geq \lfloor \rho n \rfloor$ *for* $0 < \rho < 1/2$.

Proof. Assume for contradiction that all nodes possess a neighbor whose subtree contains $\geq \lceil (1-\rho)n \rceil$ nodes. Pick an arbitrary node u_1 and let u_2 be the neighbor whose subtree contains $\geq \lceil (1-\rho)n \rceil$ nodes. By assumption, u_2 must have a neighbor u_3 whose subtree contains $\geq \lceil (1-\rho)n \rceil$ nodes. Now $u_3 \neq u_1$ because u_2 was picked as the "heavy" neighbor of u_1 so there aren't enough nodes on u_1's side. Proceed similarly until we get a sequence of $k = \lceil (1-\rho)n \rceil$ nodes: $u_1, u_2, \cdots, u_k$. By assumption, u_k has a neighbor $u_{k+1} \neq u_{k-1}$ whose subtree contains $\geq \lceil (1-\rho)n \rceil$ nodes. But there are $\geq \lceil (1-\rho)n \rceil$ nodes not in this subtree (*e.g.*, $u_1, u_2, \cdots, u_k$), leading to a contradiction. □

Pick a node v such that $V\backslash\{v\} = A \cup B$ with $\min\{|A|, |B|\} \geq n/4$ as in the above lemma. Initialize all nodes in A with weight 1 and all nodes in B with weight 0 so that the total mass in the system is $\geq n/4$. Because all the mass are on the A side of v, we must move a mass of $\geq n/8$ across v in order to reach average consensus. Since each use of an edge adjacent to v can move mass at most 1 (matrices in $G \cap S$ are doubly stochastic), we must use them at least $\geq n/8 = \Omega(n)$ times. Together with the upper bound of Theorem 6.7, we see that

Theorem 6.9. *For any connected graph G with n vertices,*

$$T^*_{G\cap S} = \Theta(n).$$

Chapter 7

Discussion and Extensions

We close with some discussions on future research directions:

- There is a gap of $O(\log^2 n)$ between the graph and tree lower bounds. One of the $O(\log n)$ factors is due to the distortion in the embedding process when we embed the graph into the L_1 space in order to easily find edge-disjoint cuts. The other $O(\log n)$ factor is an artifact of our projection after the embedding. The embedding used in Theorem 5.15 due to [1] is tight in general since metric spaces induced by expander graphs require $\Omega(\log n)$ distortion and $\Omega(\log n)$ dimension (see Theorem 4 of [1]). It is possible that the $O(\log^2 n)$ gap is not intrinsic to our consensus problem and there are other approaches to arrive at a graph lower bound which is consistent with the tree lower bound. Alternatively, we can hope for better algorithms that achieve the current lower bound.

- All of the algorithms given thus far are of a centralized nature. We assume that the scheduler has access to the graph and is able to construct a schedule ahead of time. It would be very interesting to investigate distributed algorithms that result in finite-time average-consensus under similar node constraints.

- In our analysis, we have assumed that network nodes are homogeneous and capable of performing only weighted average operations. If nodes are inhomogeneous (*e.g.,* a network of mobile phones and base stations) then their

ability to compute weighted averages may differ. It is interesting to consider the implications of inhomogeneous networks on finite-time average-consensus.

- If nodes communicate wirelessly using directional antennas, then their topology is represented by a directed graph. Also, if we think of a commodity distribution network, it is conceivable that some networks allow unidirectional flow of goods in certain portions of the network. Hence, the analysis of G-admissible factorizations of $\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}$, when G is a directed graph, is a natural extension.

- We studied the case of average-consensus where nodes tend to a $n^{-1}\mathbf{1}$ distribution. Often times there is a need to arrive at a distribution other than $n^{-1}\mathbf{1}$. For example, consider the emerging smart grid. Different times and different regions have varying demands on electricity. We would like to rapidly redistribute the resources in the network to match the changing demand profile. Phrased in our problem setting, we would be exploring G-admissible factorizations of general left-stochastic matrices.

Bibliography

[1] I. Abraham, Y. Bartal, and O. Neiman. Advances in metric embedding theory. In *Proceedings of the 38th ACM Symposium on Theory of Computing*, 2006.

[2] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *Robotics and Automation, IEEE Transactions on*, 15(5):818–828, Oct 1999.

[3] T. Arai, E. Pagello, and L. E. Parker. Guest Editorial Advances in Multirobot Systems. *IEEE Transactions on Robotics and Automation*, 18(5):655 – 661, 2002.

[4] T. C. Aysal, M. Coates, and M. Rabbat. Distributed average consensus using probabilistic quantization. In *SSP '07: Proceedings of the 2007 IEEE/SP 14th Workshop on Statistical Signal Processing*, pages 640–644, Washington, DC, USA, 2007. IEEE Computer Society.

[5] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE/ACM Transactions on Networking*, 14(SI):2508–2530, 2006.

[6] R. Carli, F. Bullo, and S. Zampieri. Quantized average consensus via dynamic coding/decoding schemes. *International Journal of Robust and Nonlinear Control*, 20:156–175, 2010.

[7] R. Carli, F. Fagnani, A. Speranzon, and S. Zampieri. Communication constraints in coordinated consensus problems. American Control Conference, June 2006.

[8] R. Carli, P. Frasca, F. Fagnani, and S. Zampieri. Gossip consensus algorithms via quantized communication. *Automatica*, 46:70–80, 2010.

[9] J.-Y. Chen, G. Pandurangan, and D. Xu. Robust computation of aggregates in wireless sensor networks: distributed randomized algorithms and analysis. *Fourth International Conference on Information Processing in Sensor Networks, 2005.*, pages 348–355, April 2005.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[11] J. Cortes. Finite-time convergent gradient flows with applications to network consensus. *Automatica*, 42(11):1993 – 2000, 2006.

[12] T. M. Cover and J. A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.

[13] A. Dobra D. Kempe and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 03)*, volume 8, page 482, 2003.

[14] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, third edition, 2005.

[15] A.D.G. Dimakis, A.D. Sarwate, and M. Wainwright. Geographic gossip : Efficient averaging for sensor networks. *IEEE Transactions on Signal Processing*, 56(3):1205–1216, March 2008.

[16] J. A. Fax and R. M. Murray. Information Flow and Cooperative Control of Vehicle Formations. *IEEE Trans. on Automatic Control*, 49(9):1465–1476, September 2004.

[17] Q. Hui, W.M. Haddad, and S.P. Bhat. Finite-time semistability theory with applications to consensus protocols in dynamical networks. In *Proceedings of the American Control Conference*, July 2007.

[18] A. S. Morse J. Lin and B. D. O. Anderson. The multi-agent rendezvous problem. *42nd IEEE Conference on Decision and Control, 2003*, page 1508, Dec. 2003.

[19] A. Kashyap, T. Başar, and R. Srikant. Quantized consensus. *Automatica*, 43(7):1192–1203, 2007.

[20] D. B. Kingston and R. W. Beard. Discrete-time average-consensus under switching network topologies. In *Proceedings of the 2006 American Control Conference*, 2006.

[21] C.-K. Ko and X. Gao. On matrix factorization and finite time average consensus. In *Proceedings of the 48th IEEE Conference on Decision and Control*, December 2009.

[22] C.-K. Ko and L. Shi. Scheduling for finite time consensus. In *Proceedings of the American Control Conference*, June 2009.

[23] J. Lavaei and R. M. Murray. On quantized consensus by means of gossip algorithm: part i: convergence proof. In *ACC'09: Proceedings of the 2009 conference on American Control Conference*, pages 394–401, Piscataway, NJ, USA, 2009. IEEE Press.

[24] J. Lavaei and R. M. Murray. On quantized consensus by means of gossip algorithm: part ii: convergence time. In *ACC'09: Proceedings of the 2009 conference on American Control Conference*, pages 2958–2965, Piscataway, NJ, USA, 2009. IEEE Press.

[25] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[26] S. Martínez, J. Cortés, and F. Bullo. Motion coordination with distributed information. *IEEE Control Systems Magazine*, 27(4):75–88, 2007.

[27] L. Moreau. Consensus seeking in multiagent systems using dynamically changing interconnection topologies. *IEEE Transactions on Automatic Control*, 50(2), 2005.

[28] P. Ogren, E. Fiorelli, and N. E. Leonard. Cooperative Control of Mobile Sensor Networks: Adaptive Gradient Climbing in a Distributed Environment. *IEEE Trans. on Automatic Control*, 49(8):1292–1302, August 2004.

[29] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and Cooperation in Networked Multi-Agent Systems. *Proceedings of the IEEE, Special Issue on Networked Control Systems*, 95(1):215–233, 2007.

[30] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 43:1520–1533, 2004.

[31] A. Papachristodoulou and A. Jadbabaie. Synchronization in oscillator networks: Switching topologies and non-homogeneous delays. *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pages 5692–5697, Dec. 2005.

[32] V.M. Preciado and G.C. Verghese. Synchronization in generalized erdos-renyi networks of nonlinear oscillators. *44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05.*, pages 4628–4633, Dec. 2005.

[33] N. N. Sadeghzadeh and A. Afshar. New approaches for distributed sensor networks consensus in the presence of communication time delay. In *CCDC'09: Proceedings of the 21st annual international conference on Chinese control and decision conference*, pages 3675–3680, Piscataway, NJ, USA, 2009. IEEE Press.

[34] A. D. Sarwate and A. G. Dimakis. The impact of mobility on gossip algorithms. In *Proceedings of the 28th Annual International Conference on Computer Communications (INFOCOM)*, Rio de Janeiro, Brazil, April 2009.

[35] D. P. Spanos, R. Olfati-Saber, and R. M. Murray. Distributed Sensor Fusion Using Dynamic Consensus. In *IFAC World Congress*, 2005.

[36] G. Strang. *Linear algebra and its applications*. Harcourt Brace Jovanovich College Publishers, third edition, 1988.

[37] S. Sundaram and C. N. Hadjicostis. Distributed consensus and linear functional calculation in networks: an observability perspective. In *The 6th International Conference on Information Processing in Sensor Networks*, 2007.

[38] S. Sundaram and C. N. Hadjicostis. Finite-Time Distributed Consensus in Graphs with Time-Invariant Topologies. In *The 26th American Control Conference, New York, NY*, 2007.

[39] Y.-P. Tian and C.-L. Liu. Robust consensus of multi-agent systems with diverse input delays and asymmetric interconnection perturbations. *Automatica*, 45(5):1347–1353, 2009.

[40] S. Vanka, V. Gupta, and M. Haenggi. On consensus over stochastically switching directed topologies. In *ACC'09: Proceedings of the 2009 conference on American Control Conference*, pages 4531–4536, Piscataway, NJ, USA, 2009. IEEE Press.

[41] L. Wang and F. Xiao. Finite-time consensus problems for networks of dynamic agents. *http://arxiv.org/pdf/math/0701724*.

[42] W. Xi, X. Tan, and J. S. Baras. A Stochastic Algorithm for Self-Organization of Autonomous Swarms. In *IEEE Conference on Decision and Control*, 2005.

[43] L. Xiao and S. Boyd. Fast Linear Iterations for Distributed Averaging. *Systems and Control Letters*, 53(1):65–78, September 2004.

[44] G. Xie, H. Liu, L. Wang, and Y. Jia. Consensus in networked multi-agent systems via sampled control: switching topology case. In *ACC'09: Proceedings of the 2009 conference on American Control Conference*, pages 4525–4530, Piscataway, NJ, USA, 2009. IEEE Press.

[45] H.-Y. Yang and S.-Y. Zhang. Consensus of multi-agent moving systems with heterogeneous communication delays. *Int. J. Syst., Control Commun.*, 1(4):426–436, 2009.