CCIT Report #809 May 2012

# Global Estimation with Local Communication*

Ittay Eyal Idit Keidar Stacy Patterson Raphi Rom
Department of Electrical Engineering,
Technion — Israel Institute of Technology,
{ittay@tx, idish@ee, stacyp@ee, rom@ee}.technion.ac.il

**Abstract**

We present a distributed optimization algorithm for estimating a continuous function such as temperature or pollution over a geographic region, e.g., a road network. The estimate is generated from samples taken by sensors placed alongside roads or in cars driving along them. We employ piecewise estimation, that is, we divide the region into sectors and find an estimate for each sector, e.g., a polynomial or a line, so that their union is a continuous function that minimizes some global error function. The computation is distributed by designating a node (either virtual or physical) that is responsible for estimating the function in each sector. The estimate is then computed based on the samples taken in the sector and information from adjacent nodes.

The algorithm works in networks with bounded, yet unknown, latencies. It accommodates dynamic inputs (samples) and node arrivals and departures. Our algorithm converges to the global optimum with only local communication, using a novel, distributed implementation of coordinate ascent optimization.

## 1 Introduction

As we enter the era of ubiquitous sensing, we are able to monitor the environment with unprecedented resolution using large scale *sensor networks*. Each sensor measures some phenomenon, and communicates wirelessly with its neighbors.

To cope with vast amounts of data collected in a wide geographical area, we need to summarize it. However, it is infeasible to collect and analyze all the information at a central location [4]. Although sensors may be equipped with cellular and long distance communication modules, these are both expensive and energy intensive.

Moreover, forwarding all samples to a central location is impractical due to the large number of messages and the heavy load on nodes close to the center. These restrictions indicate a need for a distributed solution where the summary of the sensed data is generated within the network.

We propose a novel *distributed* approach for generating a compact estimate of a continuous physical phenomenon from *samples* measured throughout a region. Our solution is *selective* — each participant only learns of the estimate in its vicinity. We seek to estimate the phenomenon by a continuous function that is optimal in some sense with respect to the samples. Our approach is to divide the region into *sectors* and to obtain an estimate for each sector such that the union of these estimates is continuous over the region. Each sector is assigned a *node*, which may be either a physical station or a virtual node [9, 5, 7]. The division of the region and assignment of nodes can be done with known techniques [9, 5, 7, 14] and is outside the scope of this paper.

*This work was partially supported by: the Israeli Ministry of Trade and Labor Rescue Consortium, the Israeli Science Foundation, the Hasso-Plattner Institute for Software Systems Engineering, the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI), the Arlene & Arnold Goldstein Center at the Technion Autonomous Systems Program, a Technion fellowship, and an Andrew and Erna Finci Viterbi Post-Doc Fellowship. The authors thank Isaac Keslassy for his good advice.

The sensors in a sector take samples and report them to the sector's node. The goal is for each node to generate an estimate of its sector that minimizes some error function while maintaining the continuity requirement among sectors. This requirement means that the estimate in each sector depends not only on local samples but also on information from other sectors.

We consider a *dynamic* setting: As time passes, new samples are accumulated, and irrelevant ones are removed. Additionally, nodes may join or leave the system due to infrastructure changes or faults. Our goal is to converge to the optimal estimate once changes cease. Note that since the nodes do not know when changes stop, they have to make a best effort to converge at all times.

As an example scenario, we take *vehicular networks* (*VANET*s). Modern cars have dozens of sensors [8], which measure various conditions such as temperature, emissions, and traffic speed. In addition, roadside units (RSUs) performing similar measurements are spread alongside roads [4]. This sensed data is extremely useful for Intelligent Transport Systems (ITS) [23, 13, 10]. For example, systems such as OnStar[1], which assist drivers in planning their route, can utilize information on weather and traffic conditions to improve their services. Furthermore, infrastructure management and research can benefit immensely from statistics on phenomena such as temperature, humidity, and pollution.

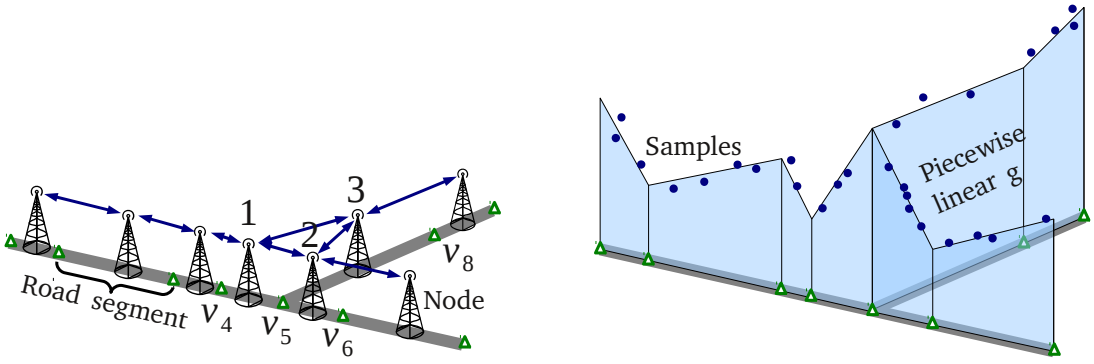
The cars serve as sensors in our algorithm, measuring values within sectors, and the RSUs function as the sectors' nodes. Short range communication with neighboring nodes can be done with VANet technologies which are gaining popularity [11].

The problem of generating the estimate as described above is an optimization problem where the objective is the sum of the error functions of the sectors, subject to equality constraints that ensure the continuity of the estimate at the boundaries of the sectors. While distributed optimization algorithms have been proposed in previous works, all the algorithms of which we are aware either assume some kind of global information or global synchronization. In our problem setting, these assumptions translate to a requirement that every node knows the entire sector topology of the region or stores and updates all of the estimation variables. Neither of these requirements is feasible in a large network. We present an overview of related work in Section 2.

After formally detailing the model and problem definition in Section 3, we present, in Section 4, our distributed algorithm for finding the optimal piecewise linear estimate over a geographic region. While the continuity requirements of our optimization problem suggest a need for atomic communication within sizable groups of nodes, we show that, by transforming the problem to its dual form, it can be solved using only pairwise communication between neighbors. We then present a novel, distributed optimization algorithm based on the method of coordinate ascent. In general, coordinate ascent is not amenable to distribution, and a naive implementation requires global synchronization. We demonstrate how to decompose the problem so that the optimization process will converge using *local* steps and provide a distributed implementation of coordinate ascent that requires no global information or synchronization. In Section 5, we show how our algorithmic approach can also be applied to derive optimal estimates using higher-order polynomials, e.g. cubic splines. We prove the correctness of our algorithms in Section 6, with some formal details deferred to the appendices.

We believe that our approach to distributed, in-network optimization without global communication or synchronization can prove useful in many additional settings. Section 7 concludes the paper and touches on some directions for future research.

[1] http://onstar.com




(a) Roads and stations.

(b) Piecewise linear estimation.

Figure 1: Piecewise linear estimation in a road system. We see a junction of three roads. (a) The roads are divided into sectors (e.g., 1, 2 and 3), each with its node. The sectors meet at vertices (triangles). Neighboring nodes are connected by arrows. (b) The samples are shown as dots, and a curtain above the roads shows the continuous piecewise linear estimate $g$.

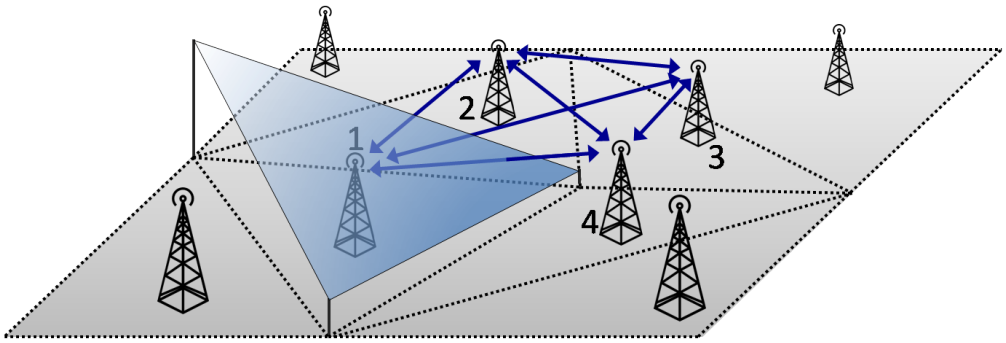



Figure 2: Piecewise linear estimation in a two dimensional region. The region is divided into sectors (e.g., 1–4), each with its node. Nodes that share a vertex are neighbors, and neighboring nodes are connected by arrows (we only show connections between nodes 1–4 for a single vertex to avoid clutter). The estimate for sector 1 is shown as a triangle above the sector.

## 2 Related Work

There has been a flurry of recent work on distributed methods for convex optimization. These works can be divided into two categories: averaging-based algorithms and sequential algorithms. The averaging-based approach builds on the framework proposed by Tsitsiklis et al. [22]. In these algorithms, every node stores a copy of all optimization variables, unlike in our case, where nodes only store variables relevant to their sectors. In every time step, a group of nodes averages their local variables and then performs an update step based on their local functions. This approach has been shown to converge to the optimal solution for unconstrained convex optimization problems with separable objectives under certain assumptions on the connectivity of the communication network over time [16, 20]. Averaging-based algorithms have also been proposed for constrained convex optimization problems where all constraints are known globally [19, 17] and where constraints are purely local [17]. An averaging-based algorithm would require that the continuity constraints of the entire network were known to every node. The per node storage requirement and the need for global information prohibits the application of these algorithms to our setting.

In the sequential distributed gradient method, there is a single set of optimization variables, and every node has its own objective function and constraints. The set of variables is passed from node to node, and it is updated by each node according to its local gradient information. If the sequence of updates meets certain requirements, the method converges to the optimal solution [12, 18]. As with averaging-based approaches, this method requires storage for all variables at every node, which is infeasible for large networks of the type we consider (e.g., a road network of an entire country.)

In this work, we present a distributed coordinate descent algorithm for convex optimization. Coordinate descent is an iterative method where, in each step, a single variable is updated. The algorithm converges to the optimal solution only if the order in which the variables are updated obeys special properties. A few recent works have proposed parallel implementations of coordinate descent for solving large optimization problems [6, 3]. While the update operations are distributed, these implementations still require that updates are executed in globally specified order, thus requiring centralized coordination. In contrast, our distributed coordinate descent algorithm simulates a centralized algorithm but requires no global information or coordination.

# 3 Model and Problem Formulation

## 3.1 Model Definition

We consider a finite region that is divided into a fixed set of non-overlapping sectors. In a road network, a sector is a segment of a road (with two vertices), as shown in Figure 1a. In a two-dimensional region, a sector is a triangle (with three vertices), as shown in Figure 2. Each sector has a unique ID, and the IDs are totally ordered. Each vertex also has a unique ID. The set of vertices for sector $i$ is denoted $\mathcal{V}_i$.

Each sector has a node that is responsible for processing and communication. The node may be part of a physical infrastructure, for example, an RSU with storage and compute resources, or it may be a virtual node comprised of a dynamic set (possibly a singleton) of mobile agents. At any time $t$, a node may be either *active* or *inactive*. Nodes either operate correctly, or stop, and their failures can be accurately detected. A node is referred to by the ID of its sector. If node $i$ (in sector $i$) fails, it may be replaced by another node, and this node will also have ID $i$. Each sector has at most one active node at any time. The set of nodes that are active at time $t$ is denoted $\mathcal{N}^t$. If a sector is associated with an active node, we say that the sector is *active*. Otherwise, the sector is *inactive*.

The set of nodes whose sectors share vertex $v$ at time $t$ are called the *members* of $v$, denoted $M^t(v)$. We say that nodes $i$ and $j$ are *neighbors* if there exists a vertex $v$ such that both $i$ and $j$ are members of $v$. Neighboring nodes are connected with bidirectional, reliable FIFO links. Examples of the communication links between nodes are shown as arrows between nodes in Figures 1a and 2. Node $i$ may send a message to its neighbor $j$ by placing a message $msg$ on the appropriate link with $\texttt{send}_i(j, msg)$, and $j$ reacts to the receipt of the message with $\texttt{recv}_j(i, msg)$.

Node $i$ maintains a dynamic set of *samples* that have been taken in its sector. The set of all samples at time $t$ is denoted $\mathcal{S}^t$. Each sample is a tuple $(x, y, z)$ where $(x, y)$ are the global coordinates at which the sample was taken, and $z$ is the value of $f$ sampled at location $(x, y)$. Node $i$ obtains a new sample in its sector at position $(x, y)$ with value $z$ with $\texttt{addSample}_i(x, y, z)$ and removes a sample from its sector (e.g., since it became outdated) with $\texttt{delSample}_i(x, y, z)$. Note that a value change at position $(x, y)$ from $z_0$ to $z_1$ is equivalent to $\texttt{delSample}_i(x, y, z_0)$ followed by $\texttt{addSample}_i(x, y, z_1)$.

When a node is added, its sample set is empty. Following the addition, it is notified of all its neighbors, and they all get notifications of it. Node $i$ is notified of a neighbor $j$ on vertex $v$

with $\texttt{addNeighbor}_i(v, j)$ and notified of its removal with $\texttt{delNeighbor}_i(v, j)$. Notification of the addition or removal of neighbors arrives at a node within a bounded, but unknown, time after the event. When the system starts, each node is notified of its neighbors.

An *execution* is a sequence of events. Each event is the response of a node to some external trigger, e.g., the arrival of a message or a notification of a new observation or neighbor. An event has an input (the message or notification) and an optional output (e.g., outgoing messages). The duration of an event is zero.

**Definition 1** (Global Stabilization Time)**.** *The* ***global stabilization time (GST)*** *is the earliest time after which the following properties hold: (1) no nodes are informed of neighbor changes, (2) samples are neither added nor deleted, and (3) the message latency of all outstanding and future messages is bounded between $\delta$ and $\Delta$. Nodes do not know GST.*

We note that eventual synchrony is necessary because each node must eventually know the identities of its living neighbors in order to ensure that the estimate is continuous at the boundaries of the sectors.

Since after GST, the sample sets and node sets are static, we omit the superscript $t$ when referencing these sets in that context.

## 3.2 Problem Formulation

Let $f$ be a real-valued, continuous, unknown function defined over the region. The objective is to learn a continuous, piecewise estimate $g$ of the function $f$ based on the samples. Each piece $g_i$ is a real-valued function defined over a single active sector $i$. The function $g$ is the union of these pieces. We address the estimation problem for two classes of continuous piecewise functions, continuous piecewise linear functions and cubic splines. The details of each class of estimation problem are presented below.

### 3.2.1 Estimation with Continuous Piecewise Linear Function

For estimation where each piece $g_i$ is a linear function, we consider two settings for the optimization problem, estimation over a road network and estimation over a two-dimensional region. In the case of a road network, each $g_i$ is a one-dimensional linear function defined over a road segment. An illustration of such a function is shown in Figure 1b; the estimate $g$ is drawn as a curtain above the roads. Let $u$ and $v$ be the IDs of the vertices of segment $i$. The function $g_i$ is parameterized by its values at its vertices, denoted $\theta_u$ and $\theta_v$, and is given by,

$$g_i(x, y; \theta_u, \theta_v) \triangleq \left(1 - \frac{loc_i(x, y)}{d_i}\right) \theta_u + \left(\frac{loc_i(x, y)}{d_i}\right) \theta_v \ ,$$

where $d_i$ is the length of segment $i$, and $loc_i(x, y)$ is a function that, for a location $(x, y)$ on segment $i$, returns the distance along the road from the vertex with the minimum ID.

In the two-dimensional region, each $g_i$ is a two-dimensional linear function defined over a triangle. Let $u, v$, and $w$ be the IDs of the vertices of triangle $i$, and let the vertex locations be $(u_x, u_y), (v_x, v_y)$, and $(w_x, w_y)$. Each $g_i$ is parameterized by its value at the vertices, denoted $\theta_u$, $\theta_v$, and $\theta_w$,

$$\begin{aligned} g_i(x, y; \theta_u; \theta_v, \theta_w) \quad = \quad & \frac{1}{a}\left((x - v_x)(w_y - v_y) + (y - v_y)(v_x - w_x)\right)\theta_u \\ & + \ \frac{1}{a}\left((x - w_x)(u_y - w_y) + (y - w_y)(w_x - u_x)\right)\theta_v \\ & + \ \frac{1}{a}\left((x - u_x)(v_y - u_y) + (y - u_y)(u_x - v_x)\right)\theta_w, \end{aligned}$$

where

$$a = u_x(w_y - v_y) \ + \ v_x(u_y - w_y) \ + \ w_x(v_y - u_y). \tag{1}$$

An example of such a $g_i$ is shown above sector 1 in Figure 2.

In the distributed setting, each node $i$ has a variable $\theta_{v,i}$ for each of its vertices. Let $\Theta$ denote the set of all $\theta$ variables and let $\Theta_i$ denote the variables for node $i$. The goal is for each node to generate an estimate of its sector that is optimal with respect to some error function. For example, the least squares error for an estimate $g_i$ is,

$$C_i(\Theta_i; \mathcal{S}_i^t) \stackrel{\Delta}{=} \sum_{(x,y,z)\in\mathcal{S}_i^t} \left(g_i(x, y; \Theta_i) - z\right)^2.$$

The error function for the entire estimate $g$ is the sum of error functions for each sector,

$$C(\Theta; \mathcal{S}^t) \stackrel{\Delta}{=} \sum_{i\in\mathcal{N}^t} C_i(\Theta_i; \mathcal{S}_i^t)$$

In addition to minimizing the global error, we also require that the estimate is continuous at the sector boundaries. This requirement means that nodes much collaborate to generate the estimate; a node's estimate will be affected not only by its neighbors, but also by nodes that are far away. The estimation problem of learning the function $g$ after GST can be formulated as a convex optimization problem with linear constraints that capture the continuity requirements,

$$\underset{\Theta}{\text{minimize}} \quad \sum_{i\in\mathcal{N}} C_i(\Theta_i; \mathcal{S}_i) \tag{2}$$

$$\text{subject to} \quad \theta_{v,i} = \theta_{v,j}, \quad \text{for } v \in \mathcal{V}, \quad i, j \in M(v), i \neq j. \tag{3}$$

The constraints (3) state that every pair of nodes in $M(v)$ has the same value for $\theta$ for vertex $v$. By transitivity, this implies that all nodes in $M(v)$ have the same value for $\theta$ for $v$.

### 3.2.2 Estimation with a Cubic Spline

We also consider the problem of estimation over a road network where each piece $g_i$ is a cubic polynomial, and the resulting global estimate is a cubic spline. The piecewise functions not only have the same function value at shared vertices, but also have the same values for the first and second derivatives at these vertices.

The function $g_i$ for each segment is parameterized by the values of $g_i$ at the endpoints, $\theta_u$ and $\theta_v$, and the values of the second derivative of $g_i$ at the endpoints, denoted $\phi_u$ and $\phi_v$,

$$\begin{aligned} g_i(x, y; \theta_u, \theta_v, \phi_u, \phi_v) \ = \ & \left(1 - \frac{loc_i(x,y)}{d_i}\right)\theta_u + \left(\frac{loc_i(x,y)}{d_i}\right)\theta_v + \\ & \left(-\frac{1}{6d_i}(loc_i(x,y))^3 + \frac{1}{2}(loc_i(x,y))^2 \ - \ \frac{1}{3}d_i loc_i(x,y)\right)\phi_u + \\ & \left(\frac{1}{6d_i}(loc_i(x,y))^3 - \frac{1}{6}d_i loc_i(x,y)\right)\phi_v. \end{aligned}$$

and the corresponding optimization problem optimizes over both the values of each $g_i$ at its endpoints, denoted $\Theta_i$ and the values of the second derivative of each $g_i$ at its endpoints, denoted

$\Phi_i$,

$$\underset{\Theta,\Phi}{\text{minimize}} \quad \sum_{i\in\mathcal{N}} C_i(\Theta_i, \Phi_i; \mathcal{S}_i) \tag{4}$$

$$\text{subject to} \quad \theta_{v,i} = \theta_{v,j} \quad \text{for } v \in \mathcal{V}, \quad i,j \in M(v), i \neq j \tag{5}$$

$$g_i'(v_x, v_y; \Theta_i, \Phi_i) = g_i'(v_x, v_y; \Theta_i; \Phi_i) \quad \text{for } v \in \mathcal{V}, \quad i,j \in M(v), i \neq j \tag{6}$$

$$\phi_{v,i} = \phi_{v,j} \quad \text{for } v \in \mathcal{V}, \quad i,j \in M(v), i \neq j. \tag{7}$$

As before, $\Theta$ is the set of all $\theta$ variables, and $\Phi$ is the set of all $\phi$ variables.

### 3.3 Summary

In summary, our goal is to design a distributed algorithm for finding the values of the parameters $\Theta$ (and $\Phi$ if relevant) that solve the optimization problem defined above. Each node $i$ knows only its own sample set $\mathcal{S}_i$ and communicates only with its neighbors, as explained in Section 3.1. Each node $i$ is responsible for obtaining an estimate of its sector by learning values for $\Theta_i$ (and $\Phi_i$ if relevant). After GST, these estimates must converge to globally optimal estimate.

## 4 Algorithm for Piecewise Linear Estimation

In this section, we present our distributed algorithm for generating the optimal continuous piecewise linear estimate defined in Section 3.2.1. In the optimization problem (2 - 3), all members of a vertex must agree on the value of $\theta$ for that vertex. Therefore, a distributed algorithm that addresses this problem directly requires expensive coordination among all nodes that share a vertex (An example of such an algorithm is presented in Appendex E). We eliminate the need for vertex-wise coordination by instead solving the dual problem. Our algorithm is based on the coordinate ascent method for nonlinear optimization [15, 1]. We briefly review this method in Section 4.1. We present the transition to the dual form of the problem in Section 4.2, and we present our distributed algorithm in Section 4.3. In Section 6, we formally prove the correctness of our algorithm.

### 4.1 Preliminaries — Coordinate Ascent Method

Let $h(x_1, \ldots, x_m)$ be a function that is strictly convex in each $x_i$ when the other variables, $x_j, j \neq i$ are held constant, and let $h$ have continuous first partial derivatives. We consider the unconstrained optimization problem,

$$\underset{x\in\mathbb{R}^m}{\text{maximize}} \quad h(x_1, x_2, \ldots, z_x). \tag{8}$$

The optimization problem can be solved using the coordinate ascent method, which is executed as follows. Let $x(k) = [x_1(k), \ldots, x_m(k)]$ be the vector of the values of the $x_i$'s in iteration $k$. The algorithm begins with an initial $x(1)$. In each step $k$, a coordinate $i$ is selected, and $x(k)$ is updated by finding its maximum when all other values of x(k) are fixed. The update step is,

$$\overline{x}_i = \arg\max_{\xi} h(x_1(k), \ldots x_{i-1}(k), \xi, x_{i+1}(k), \ldots, x_m(k)) \tag{9}$$

$$x(k+1) = [x_1(k), \ldots, x_{i-1}(k), \overline{x}_i, x_{i+1}(k), \ldots, x_m(k)]. \tag{10}$$

As shown, $x(k+1)$ is generated by replacing component $i$ of $x(k)$ with $\overline{x}_i$. We note that it is possible that an update step may not result in any change to $x$, i.e. $x(k+1) = x(k)$, if the value of selected coordinate is already optimal with respect to the rest of $x(k)$.

It has been shown that, in order to guarantee that the algorithm above converges to the $x$ that optimizes problem (8), it is necessary that the order in which the coordinates are updated satisfies certain properties. One order policy that guarantees convergence is the *essentially cyclic rule* [1, 21], which states that there exists a constant integer $T$, such that every coordinate $j \in \{1, \ldots, m\}$ is chosen at least once between the $r^{th}$ iteration and the $(r + T - 1)^{th}$ iteration, for all $r$.

### 4.2 Formulating the Dual Problem

Given a constrained optimization problem, the dual problem is formed by incorporating the constraints into the objective function. To form the dual for problem (2 - 3), we first define the Lagrangian,

$$\mathcal{L}(\Theta, \Lambda; \mathcal{S}) = \sum_{i \in \mathcal{N}} C_i(\Theta_i; \mathcal{S}_i) + \sum_{v \in \mathcal{V}} \sum_{\substack{i,j \in M(v) \\ i \neq j}} \lambda_{i,j}^v \left(\theta_{v,i} - \theta_{v,j}\right). \tag{11}$$

Here, each equality $\theta_{v,i} = \theta_{v,j}$ constraint in (3) is assigned a Lagrange multiplier $\lambda_{i,j}^v \in \mathbb{R}$. Let $\Lambda$ denote the set of all Lagrange multipliers, and let $\Lambda_i$ denote the set of Lagrange multipliers associated with a constraint involving $\theta_i$. We note that a $\lambda_{i,j}^v$ is an element of both $\Lambda_i$ and $\Lambda_j$. The dual function is defined as follows,

$$q(\Lambda; \mathcal{S}) \triangleq \inf_{\Theta} \ \mathcal{L}(\Theta, \Lambda; \mathcal{S}). \tag{12}$$

In our case, equation (12) can be expressed as a sum over the nodes in $\mathcal{N}$,

$$q(\Lambda; \mathcal{S}) = \sum_{i \in \mathcal{N}} q_i(\Lambda_i; \mathcal{S}_i),$$

The function $q_i$ depends only on information local to node $i$, i.e. $\mathcal{S}_i$ and the location of of the vertices of sector $i$. In the examples described in Section 3.2, $q_i$ is a quadratic minimization problem (over $\Theta_i$) and thus can be solved analytically. The full expression for $q_i$ is given in Appendix A.1.

The dual problem is an unconstrained convex optimization problem over $\Lambda$,

$$\underset{\Lambda}{\text{maximize}} \ \ q(\Lambda; \mathcal{S}) = \sum_{i \in \mathcal{N}} q_i(\Lambda_i; \mathcal{S}_i). \tag{13}$$

Let $\hat{\Lambda}$ be the argument that maximizes $q$. For a square error minimization of the form (2-3), strong duality holds (see [2]). Therefore, the solution to (13) gives the solution to the primal problem,

$$\hat{\Theta} = \arg\min_{\Theta} \ \mathcal{L}(\Theta, \hat{\Lambda}; \mathcal{S}).$$

The dual problem (13) is an unconstrained convex optimization problem that is strictly convex in each $\lambda_{i,j}^v$ when the other values in $\Lambda$ are held constant. Therefore, we can solve this problem using the method of coordinate ascent. In the next section, we present a novel, distributed implementation of the coordinate ascent algorithm that solves the dual problem.

### 4.3 Distributed Coordinate Ascent Algorithm

We present our distributed coordinate ascent algorithm to solve Problem (13), where the coordinates are the variables in $\Lambda$. As stated above, a Lagrange multiplier, $\lambda_{i,j}^v \in \Lambda$, appears in two sets, $\Lambda_i$ and $\Lambda_j$. In our algorithm, nodes $i$ and $j$ both store a copy of $\lambda_{i,j}^v$; we refer to $\lambda_{i,j}^v$ as a *shared variable*. Nodes $i$ and $j$ collaborate to perform the update step for coordinate $\lambda_{i,j}^v$ and update their copies.

In the remainder of this section, we describe how we distribute the coordinate ascent algorithm. First, we detail what information is exchanged in order to perform a distributed coordinate ascent update step. We then explain how we perform such steps atomically in the face of partial synchrony, and we show how we ensure that the order of update steps follows the essentially cyclic rule. Finally we show how the distributed algorithm accommodates dynamics, namely the addition and removal of nodes and samples. The pseudocode appears in Appendix C.

**Coordinate update step** For an update step for coordinate $\lambda_{i,j}^v$, its new value $\gamma$ depends only on the dual functions for nodes $i$ and $j$,

$$\gamma = \arg\max_{\lambda_{i,j}^v} q(\Lambda; \mathcal{S}) = \arg\max_{\lambda_{i,j}^v} \left( q_i(\Lambda_i; \mathcal{S}_i) + q_j(\Lambda_j; \mathcal{S}_j) \right).$$

The value of $\gamma$ is the root of the equation,

$$\frac{\partial}{\partial \lambda_{i,j}^v} (q_i + q_j) = \frac{\partial}{\partial \lambda_{i,j}^v} q_i + \frac{\partial}{\partial \lambda_{i,j}^v} q_j = 0.$$

To find this value, each node sends information about its partial derivative to the other. We show that this information can be encapsulated in two coefficients $\alpha_i^v$ and $\beta_i^v$ (see Appendix A.2). Each node then independently computes $\gamma$ as follows,

$$\gamma = -(\beta_i^v + \beta_j^v)/(\alpha_i^v + \alpha_j^v), \tag{14}$$

and updates its copy of $\lambda_{i,j}^v$.

The values of the $\alpha$ and $\beta$ coefficients for a given node are determined by the node's samples and the values of its other shared variables, and the shared variables depend on the values of variables shared with other nodes, which in turn, depend on additional shared variables. For the coordinate update step to be performed correctly, both nodes involved in the update must compute their coefficients using a consistent shared state. We proceed to explain how we ensure that each coordinate is updated atomically.

**Implementing atomic updates** We enforce the update atomicity by assigning a leader and a follower to each shared variable; the node with the smaller ID is the leader. Note that each node may act as a follower for some variables and as a leader for others. Whenever a node decides it is necessary to update a variable, either when another shared variable changes in the course of convergence or due to dynamics (see below), it sends a NOTIFY message to the leader of that variable (either its neighbor or itself). The leader processes one notification at a time. It initiates the update step by sending an UPDATE message containing its coefficients to the respective follower, and it waits for a response. While it is waiting, it does not process any UPDATE messages for variables for which it is not the leader. When a follower receives coefficients from a leader, it sends its coefficients to the leader, and it updates its copy of $\lambda_{i,j}^v$. Note that the follower does not block during the update exchange. When the leader receives the coefficients from the follower, it updates its copy of $\lambda_{i,j}^v$. After the update, the leader is free to deal with its other notifications and updates.

**Ensuring essentially cyclic update order** In order to enforce the essentially cyclic scheduling policy, while it waits, the leader queues all incoming NOTIFY and UPDATE messages from other neighbors. After the update is completed, it handles the queued messages in order. It first deals with all variables of which it is a follower (the UPDATE messages), since these do not require blocking. It then processes the next notification. This internal scheduling ensures that the update frequency ratio between different shared variables is bounded, thus obtaining an essentially cyclic update rule.

**Dealing with dynamics** Shared variables may have to be updated for several reasons. As explained above, in the course of convergence, the update of a shared variable may trigger the update of another shared variable belonging to the same node. In addition, dynamic system behavior may also trigger such updates. Whenever a node's sample set $\mathcal{S}_i^t$ changes, all of its shared variables require an update. Additionally, changes to a node's neighbor set entail a change of its shared variable set (addition or removal of a $\lambda$), which, in turn, may require updating all of its shared variables. In all these cases, the node can independently detect a change in $\alpha_i^v$ and $\beta_i^v$ for a shared variable $\lambda_{i,j}^v$. When this happens, node $i$ notifies the leader for the $\lambda_{i,j}^v$ that an update step is needed.

Whenever one of its shared variables are updated, a node may have to update other variables, as explained above. If so, it notifies the leader of all such variables. At most one NOTIFYmessage is sent for a variable per update of that variable. A leader sends its coefficients only in response to a notification on the relevant shared variable, and a follower responds with its coefficients only after receiving the leader's coefficients. Therefore (after GST) when the system reaches the optimal solution to problem (13), after a finite period of time no additional NOTIFY or UPDATE messages are sent, and the system achieves quiescence.

# 5 Algorithm for Estimation with Cubic Splines

The distributed algorithm for optimal estimation with cubic splines (as defined in Section 3.2.2) is nearly identical to the algorithm presented in the previous section. The two points of difference are the dual problem formulation and the information exchange for the coordinate update step. We explain these two points below.

## 5.1 Forming the Dual Problem

We begin by defining the dual problem for the optimization problem stated in equations (4) - (6). The Lagrangian for this problem is

$$\begin{aligned}\mathcal{L}(\Theta,\Lambda;\mathcal{S}) \;=\; & \sum_{i\in\mathcal{N}} C_i(\Theta_i;\mathcal{S}_i) \;+\; \sum_{v\in\mathcal{V}} \sum_{\substack{i,j\in M(v)\\ i\neq j}} \lambda_{i,j}^{v,1}\left(\theta_{v,i}-\theta_{v,j}\right) \;+\\ & \sum_{v\in\mathcal{V}} \sum_{\substack{i,j\in M(v)\\ i\neq j}} \lambda_{i,j}^{v,2}\left(g_i'(v_x,v_y;\Theta_i,\Phi_i)-g_j'(v_x,v_y;\Theta_j,\Phi_j)\right) \;+\\ & \sum_{v\in\mathcal{V}} \sum_{\substack{i,j\in M(v)\\ i\neq j}} \lambda_{i,j}^{v,3}\left(\phi_{v,i}-\phi_{v,j}\right)\end{aligned}$$

As in the previous section, each equality $\theta_{v,i}=\theta_{v,j}$ constraint in (5) is assigned a Lagrange multiplier $\lambda_{i,j}^{v,1}\in\mathbb{R}$. Each equality constraint involving the first derivative of $g_i$ (Equation (6)) is assigned a Lagrange multiplier $\lambda_{i,j}^{v,2}\in\mathbb{R}$, and each equality constraint involving the second derivative of $g_i$ (Equation (7)) is assigned a Lagrange multiplier $\lambda_{i,j}^{v,3}\in\mathbb{R}$. For sector $i$ with vertices $u$ and $v$, the value of $g_i'$ at vertex $v$ is given by,

$$g_i'(v_x,v_y;\Theta_i,\Phi_i)=c_{v,i}^{\theta_u}\theta_{u,i}+c_{v,i}^{\theta_v}\theta_{v,i}+c_{v,i}^{\phi_u}\phi_{u,i}+c_{v,i}^{\phi_v}\phi_{v,i}, \tag{15}$$

where $c_{v,i}$ are coefficients that depends on which vertex is considered; i.e., the function $g_i'$ evaluated at vertex $v$ has different coefficients than the same $g_i'$ evaluated at vertex $u$. As shown by (15), each equality constraint in (6) involves all of node $i$'s variables, $\theta_{u,i}$, $\theta_{v,i}$, $\phi_{u,i}$ and $\phi_{v,i}$.

The dual function is

$$q(\Lambda;\mathcal{S}) \triangleq \inf_{\Theta;\Phi} \mathcal{L}(\Theta,\Phi,\Lambda;\mathcal{S}), \tag{16}$$

and as before this function can be expressed as a sum over the nodes in $\mathcal{N}$,

$$q(\Lambda;\mathcal{S}) = \sum_{i\in\mathcal{N}} q_i(\Lambda_i;\mathcal{S}_i).$$

The optimization problem in (16) can be solved analytically using information local to sector $i$; the solution is given in Appendix B. The resulting dual problem is of the same form as (13).

Strong duality also holds for the problem (4) - (6). Therefore, the argument $\hat{\Lambda}$ that solves the dual problem gives the solution to the primal problem,

$$\hat{\Theta},\hat{\Phi} = \arg\min_{\Theta,\Phi} \mathcal{L}(\Theta,\Phi,\hat{\Lambda};\mathcal{S}).$$

### 5.2 Distributed Block Coordinate Ascent Algorithm

As in the algorithm for piecewise linear estimation, we use a distributed coordinate ascent algorithm to solve the dual problem. In the piecewise linear estimation setting, for each vertex $v$, there is a shared variable $\lambda_{i,j}^v$ for every pair of nodes $i,j \in M(v)$. In the case of cubic splines, every pair of nodes $i,j \in M(v)$ has two shared variables, one for the constraint in (5) and one for the constraint in (6). We denote these shared variables by $\lambda_{i,j}^{v,1}$ and $\lambda_{i,j}^{v,2}$ respectively. Rather than optimizing for one shared variable at a time, our algorithm optimizes for both shared variables in a single update step. This approach is known as *block coordinate ascent* [21]. We now detail the pairwise information exchange in our distributed block coordinate ascent algorithm.

**Block Coordinate Update Step** To execute an update step between nodes $i$ and $j$ on vertex $v$, the nodes must collaborate to find $\gamma = [\gamma_1\ \gamma_2\ \gamma_3]^\mathsf{T}$ such that

$$\gamma = \arg\min_{\lambda_{i,j}^{v,1},\lambda_{i,j}^{v,2},\lambda_{i,j}^{v,3}} q(\Lambda;\mathcal{S}) = \arg\min_{\lambda_{i,j}^{v,1},\lambda_{i,j}^{v,2},\lambda_{i,j}^{v,3}} \left(q_i(\Lambda_i;\mathcal{S}_i) + q_j(\Lambda_j;\mathcal{S}_j)\right).$$

The vector $\gamma$ is the solution to the equation

$$\frac{\partial}{\partial\lambda_{i,j}^v}(q_i+q_j) = \frac{\partial}{\partial\lambda_{i,j}^v}q_i + \frac{\partial}{\partial\lambda_{i,j}^v}q_j = 0.$$

Here we use $\lambda_{i,j}^v$ to denote the vector $[\lambda_{i,j}^{v,1}\ \lambda_{i,j}^{v,2}\ \lambda_{i,j}^{v,3}]^\mathsf{T}$. To find $\gamma$, each node sends information about its $q_i$ to the other. This information for node $i$ can be encapsulated in four coefficients, $\alpha_i^{v,1}$, $\alpha_i^{v,2}$, $\alpha_i^{v,3}$, and $\alpha_i^{v,4}$ (see Appendix B). After receiving coefficients from its neighbor, each node independently computes $\gamma$. The node then updates its copy of the shared variables $\lambda_{i,j}^{v,1}$, $\lambda_{i,j}^{v,2}$, and $\lambda_{i,j}^{v,3}$ with the values $\gamma$.

## 6 Convergence

In this section, we prove the correctness of the distributed estimation algorithms presented in Sections 4 and 5. For clarity, we formulate the proof in the context of the distributed linear estimation algorithm only. We note that the distributed spline estimation algorithm is identical to the distributed linear estimation algorithm with respect to the elements of our proof. Therefore, all convergence results apply directly to this algorithm as well.

In order to prove that the algorithm in Section 4.3 converges to the optimum, we first show that it *simulates* the centralized coordinate ascent algorithm (as described in Section 4.1). We then prove that the order of coordinate updates follows the essentially cyclic rule. From these, we derive the convergence proof.

**Simulation** The state of the centralized system consists of the Lagrange multipliers $\Lambda$. In each step of the centralized algorithm, one Lagrange multiplier is chosen and updated, with the others held constant, moving the system into a new state.

We define a mapping $\mathcal{F}$ between the state of the distributed algorithm and the state of the centralized one. For a given state of the distributed algorithm at time $t$, $\mathcal{F}$ returns a state where the value of each multiplier $\lambda$ is its value at the node with the higher ID of the two nodes sharing it (follower). Under this mapping, the distributed algorithm simulates the centralized one, as stated in the following lemma. We defer the proof to Appendix D.1.

**Lemma 1** (Simulation)**.** *After GST, the distributed algorithm simulates the centralized coordinate ascent algorithm, under the mapping $\mathcal{F}$.*

**Essentially Cyclic Rule** We consider the *pairwise steps* of our algorithm, where a follower node sends coefficients to its neighbor. Lemma 2 states that there exists a bound $T$ such that in every step sequence of length $T$ after GST, each shared variable $\lambda^v_{i,j}$ is chosen at least once.

**Lemma 2** (Essentially Cyclic)**.** *For every execution of the distributed algorithm, the simulation under $\mathcal{F}$ yields an execution of the centralized coordinate descent algorithm that follows an essentially cyclic policy.*

The proof, deferred to Appendix D.2, is performed by bounding the maximal ratio between steps of shared variables that are not at the optimum value.

**Convergence** We now prove Theorem 1, which states that the distributed algorithm converges to the optimal solution.

**Theorem 1.** *After GST, the values of the variables in $\Theta$, as maintained by the nodes, converge to the optimal estimate, i.e., the solution of Problem (2-3).*

*Proof.* Lemma 1 shows that the distributed algorithm simulates the centralized coordinate ascent algorithm, and Lemma 2 shows that the order of coordinate updates of the simulated algorithm follows the essentially cyclic rule. As shown in [21], a centralized coordinate ascent algorithm using an essentially cyclic rule converges to a stationary point. In our problem formulation, the stationary point is the unique, optimal solution of Problem (13). Since the steps of the distributed algorithm correspond to the messages updating the values of the optimization variables, and since they occur at most every $\text{maxDegree} \times \text{maxChain} \times 2 \times \Delta$ (Lemma 4), we conclude that the distributed algorithm converges to the solution of Problem (13), as required. □

As explained in Section 4.2, convergence of $\Lambda$ to the optimum implies convergence of the variables $\Theta$ to the solution of Problem (2-3), as required in Theorem 1.

# 7 Conclusion

We have presented a distributed optimization algorithm for estimating a continuous function over a geographic region based on samples taken by sensors inside the region. The region is divided into sectors, with a node in charge of each sector. Sample data for a particular sector is not transmitted to other sectors, and the algorithm generates a continuous, piecewise estimate that minimizes a *global* error function with respect to all samples. Our algorithm accommodates dynamic inputs (samples) and node arrivals and departures. It converges to the *global* optimum with only *local* communication. To achieve this, we first formulate the dual of the optimization problem in order to eliminate the need for vertex-wise coordination. We then devise a novel, distributed implementation of coordinate ascent optimization to solve this dual problem.

This work demonstrates the benefits and power of distributed *selective* learning, where agents cooperate to calculate a global optimum, while each of them learns only a part of the solution. These results call for future work, studying the possibility of relaxing the communication patterns even further and extending the algorithm to other optimization problems with different objective functions and constraints. We note that our algorithm is generic in essence. It can be directly applied to many equality constrained convex optimization problem whose constraints coincide with the communication topology.

## References


[1] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 2nd edition, 1999.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[3] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *Proceedings of the 28th International Conference on Machine Learning*, pages 321–328, 2011.

[4] C.Y. Chong and S.P. Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, 2003.

[5] S. Dolev and N. Tzachar. Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. *Theoretical Computer Science*, 410(6-7):514–532, 2009.

[6] M. Elad, B. Matalon, and M. Zibulevsky. Coordinate and subspace optimization methods for linear least squares with non-quadratic regularization. *Applied and Computational Harmonic Analysis*, 23:346–367, November 2007.

[7] Y. Fernandess and D. Malkhi. K-clustering in wireless ad hoc networks. In *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing*, pages 31–37. ACM, 2002.

[8] W.J. Fleming. New automotive sensors - a review. *IEEE Sensors Journal*, 8(11):1900–1921, 2008.

[9] S. Gilbert, N. Lynch, S. Mitra, and T. Nolte. Self-stabilizing mobile robot formations with virtual nodes. *Stabilization, Safety, and Security of Distributed Systems*, pages 188–202, 2008.

[10] D. Jiang, A. Delgrossi, L. Festag, A. Hessler, R. Baldessari, L. Le, W. Zhang, and D. Westhoff. Vehicle-to-vehicle and road-side sensor communication for enhanced road safety. In *Proceedings of the ITS World Congress and Exhibition, New York*, 2008.

[11] D. Jiang and L. Delgrossi. IEEE 802.11 p: Towards an international standard for wireless access in vehicular environments. In *IEEE Vehicular Technology Conference*, pages 2036–2040. IEEE, 2008.

[12] B. Johansson, M. Rabi, and M. Johansson. A randomized incremental subgradient method for distributed optimization in networked systems. *SIAM Journal on Optimization*, 20(3):1157–1170, 2009.

[13] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil. Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions. *IEEE Communications Surveys & Tutorials*, (99):1–33, 2011.

[14] Dong-Young Lee and Simon S. Lam. Efficient and accurate protocols for distributed Delaunay triangulation under churn. In *IEEE International Conference on Network Protocols*, pages 124–136, 2008.

[15] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, Inc., 2nd edition, 1984.

[16] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[17] A. Nedic, A. Ozdaglar, and P.A. Parrilo. Constrained consensus and optimization in multi-agent networks. *IEEE Transactions on Automatic Control*, 55(4):922–938, 2010.

[18] S.S. Ram, A. Nedic, and VV Veeravalli. Asynchronous gossip algorithms for stochastic optimization. In *Proceedings of the 48th IEEE Conference on Decision and Control*, pages 3581–3586, 2009.

[19] S.S. Ram, A. Nedic, and VV Veeravalli. A new class of distributed optimization algorithms: Application to regression of distributed data. *Optimization Methods and Software*, 27(1):71–88, 2012.

[20] K. Srivastava and A. Nedic. Distributed asynchronous constrained stochastic optimization. *IEEE Journal of Selected Topics in Signal Processing*, 5(4):772–790, 2011.

[21] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109(3):475–494, June 2001.

[22] J. Tsitsiklis, D. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, 1986.

[23] F.Y. Wang, D. Zeng, and L. Yang. Smart cars on smart roads: an IEEE intelligent transportation systems society update. *IEEE Pervasive Computing*, 5(4):68–69, 2006.

# A Mathematical Derivations for Piecewise Linear Estimation

## A.1 Derivation of $q_i$

In our examples, each $q_i$ can be written as

$$q_i(\Lambda_i; \mathcal{S}_i) = \inf_{\Theta_i} \Theta_i^\mathsf{T}(X_i^\mathsf{T} X_i)\Theta_i \; - \; 2z_i^\mathsf{T} X_i^\mathsf{T}\Theta_i + z_i^\mathsf{T} z_i + \sum_{v\in\mathcal{V}_i}\left(\sum_{j\in M(v), j\neq i} \mathrm{sgn}(j-i)\lambda_{i,j}^v\right)\theta_{v,i}$$

With some abuse of notation, we use $\Theta_i$ to also represent the vector form of the set. The function $\mathrm{sgn}(j-i)$ returns 1 if $i<j$, i.e. $i$ is the leader, and returns 0 otherwise. The matrix $X_i$ and the vector $z_i$ are completely determined by the vertices and samples of sector $i$. The precise definitions of $X_i$ and $z_i$ are given below

**Definition of $X_i$ for a road segment** For a road segment with id $i$, $X_i$ is a $k\times 2$ matrix, where $k=|\mathcal{S}_i|$. Each row $j$ corresponds to a sample $(x_j, y_j, z_j)\in\mathcal{S}_i$, and is defined

$$X_i^{[j,:]} = \left[\; 1-(loc_i(x_j,y_j)/d_i) \quad loc_i(x_j,y_j)/d_i \;\right]$$

Recall that $d_i$ is the length of road segment $i$ and $loc_i(\cdot,\cdot)$ is a function that maps a sample location to its distance along the road segment from the vertex with the smallest ID.

**Definition of $X_i$ for a triangular sector** Let the vertices of sector $i$ be $u$, $v$, and $w$, and let the location of these vertices be $(u_x,u_y)$, $(v_x,v_y)$ and $(w_x,w_y)$. The matrix $X_i$ has $k$ rows, with $k=|\mathcal{S}_i|$ and three columns. Each row $j$ corresponds to a sample $(x_j,y_j,z_j)$ and is given by

$$\begin{aligned}
X_i^{[j,1]} &= \frac{1}{a}\left((x_j-v_x)(w_y-u_y)+(y_j-v_y)(v_x-w_x)\right)\\
X_i^{[j,2]} &= \frac{1}{a}\left((x_j-w_x)(u_y-w_y)+(y_j-w_y)(w_x-u_x)\right)\\
X_i^{[j,3]} &= \frac{1}{a}\left((x_j-u_x)(v_y-u_y)+(v_y-u_y)(u_x-v_x)\right)
\end{aligned}$$

where $a$ is as defined in Equation (1).

**Definition of $z_i$** For both the road network and two-dimensional examples, $z_i$ is vector with $k=|\mathcal{S}_i|$ components, one per sample. Each component corresponds to a sample $(x_j,y_j,z_j)$ and its value is the value of the sample, $z_j$.

We define a vector $\Gamma_i$ with one component for each vertex of sector $i$. The component for vertex $v$ is

$$\Gamma_i^{[v]} = \sum_{\substack{j\in M(v)\\ j\neq i}} \mathrm{sgn}(j-i)\lambda_{i,j}^v.$$

Note the components of $\Gamma_i$ are ordered according to the vertices of sector $i$. If the vertices of sector $i$ are, in order, $u$ and $v$ (for the road network example), the first component of $\Gamma_i$ corresponds to vertex $u$ and the second component corresponds to vertex $v$. This order is mimicked in the components of $\Theta_i$ and $\Lambda_i$ and in the rows of $X_i$.

Each $q_i$ is a quadratic minimization problem and thus can be solved analytically at each node $i$. The expression for $q_i$ simplifies to

$$q_i(\Lambda_i;\mathcal{S}_i) = z_i^\mathsf{T} X_i\left(X_i^\mathsf{T}X_i\right)^{-1}X_i^\mathsf{T}z_i + z_i^\mathsf{T}X_i\Gamma_i - \frac{1}{4}\Gamma_i^\mathsf{T}\left(X_i^\mathsf{T}X_i\right)^{-1}\Gamma_i.$$

## A.2 Algorithm Coefficients

To simplify the notation, we define the following

$$A_i \triangleq \left(X_i^\mathsf{T} X_i\right)^{-1} \qquad b_i \triangleq X_i^\mathsf{T} z_i.$$

Let the components of $A_i$ and $b_i$ be denoted $a_i^{[j,k]}$ and $b_i^{[j]}$ respectively.

We can then rewrite $q_i$ as

$$q_i(\Lambda_i; \mathcal{S}_i) = b_i^\mathsf{T} A_i b_i + b_i^\mathsf{T} \Gamma_i - \frac{1}{4} \Gamma_i^\mathsf{T} A_i \Gamma_i.$$

We wish to find the partial derivative of $q_i$ with respect to a single component $\lambda_{i,j}^v \in \Lambda_i$ when all other components are held fixed. The partial derivative is of the form

$$\frac{\partial}{\partial \lambda_{i,j}^v} = \alpha_i^v \lambda_{i,j}^v + \beta_i^v.$$

We specify the values of the coefficients $\alpha_i^v$ and $\beta_i^v$ for our examples below.

**Road Network** Let the vertices of segment $i$ be $u$ and $v$, where $u$ corresponds to the first component in the vector $\Theta_i$ and $v$ corresponds to the second component. For an update of coordinate $\lambda_{i,j}^u$,

$$\alpha_i^u = -\frac{1}{4} a_i^{[1,1]} \qquad \beta_i^u = b_i^{[1]} - \frac{1}{2} a_i^{[1,2]} \Gamma_i^{[2]} - \frac{1}{2} a_i^{[1,1]} \sum_{\substack{k \in M(u) \\ k \neq i, k \neq j}} \operatorname{sgn}(k-i) \lambda_{i,k}^u.$$

For an update of coordinate $\lambda_{i,j}^v$,

$$\alpha_i^v = -\frac{1}{2} a_i^{[2,2]} \qquad \beta_i^v = b_i^{[2]} - \frac{1}{2} a_i^{[1,2]} \Gamma_i^{[1]} - \frac{1}{2} a_i^{[2,2]} \sum_{\substack{k \in M(v) \\ k \neq i, k \neq j}} \operatorname{sgn}(k-i) \lambda_{i,k}^u.$$

**2-Dimensional Region** Let the vertices of sector $i$ be $u$, $v$, and $w$, where $u$ corresponds to the first component in the vector $\Theta_i$, $v$ corresponds to the second component, and $w$ corresponds to the third component. For an update of coordinate $\lambda_{i,j}^u$

$$\alpha_i^u = -\frac{1}{2} a_i^{[1,1]} \qquad \beta_i^u = b_i^{[1]} - \frac{1}{2} a_i^{[1,2]} \Gamma_i^{[2]} - \frac{1}{2} a_i^{[1,3]} \Gamma_i^{[3]} - \frac{1}{2} a_i^{[1,1]} \sum_{\substack{k \in M(u) \\ k \neq i, k \neq j}} \operatorname{sgn}(k-i) \lambda_{i,k}^u.$$

For an update of coordinate $\lambda_{i,j}^v$,

$$\alpha_i^v = -\frac{1}{2} a_i^{[2,2]} \qquad \beta_i^v = b_i^{[2]} - \frac{1}{2} a_i^{[2,1]} \Gamma_i^{[1]} - \frac{1}{2} a_i^{[2,3]} \Gamma_i^{[3]} - \frac{1}{2} a_i^{[2,2]} \sum_{\substack{k \in M(v) \\ k \neq i, k \neq j}} \operatorname{sgn}(k-i) \lambda_{i,k}^v.$$

For an update of coordinate $\lambda_{i,j}^w$

$$\alpha_i^w = -\frac{1}{2} a_i^{[3,3]} \qquad \beta_i^w = b_i^{[3]} - \frac{1}{2} a_i^{[3,1]} \Gamma_i^{[1]} - \frac{1}{2} a_i^{[3,2]} \Gamma_i^{[2]} - \frac{1}{2} a_i^{[3,3]} \sum_{\substack{k \in M(w) \\ k \neq i, k \neq j}} \operatorname{sgn}(k-i) \lambda_{i,k}^w.$$

## B Mathematical Derivations for Estimation with Cublic Splines

For a sector $i$ with vertices $u$ and $v$, the vector of variables that parameterize the estimate $g_i$ is

$$\eta_i = \left[ \begin{array}{cccc} \theta_{u,i} & \theta_{v,i} & \phi_{u,i} & \phi_{v,i} \end{array} \right]^\mathsf{T}$$

The function $q_i$ in Section 5.1 is by

$$q_i(\Lambda_i; \mathcal{S}_i) = \inf_{\eta_i} \eta_i^\mathsf{T}(X_i^\mathsf{T} X_i)\eta_i \ - \ 2z_i^\mathsf{T} X_i \eta_i + z_i^\mathsf{T} z_i \ + \tag{17}$$

$$\sum_{\substack{v\in\mathcal{V}_i \\ u\in\mathcal{V}_i, u\neq v}} \left[ \sum_{\substack{j\in M(v) \\ j\neq i}} \mathrm{sgn}(j-i)\lambda_{i,j}^{v,1} + \sum_{\substack{j\in M(v) \\ j\neq i}} \mathrm{sgn}(j-i)c_{v,i}^{\theta_v}\lambda_{i,j}^{v,2} + \sum_{\substack{k\in M(u) \\ k\neq i}} \mathrm{sgn}(k-i)c_{u,i}^{\theta_v}\lambda_{i,k}^{u,2} \right] \theta_{v,i} \tag{18}$$

$$+ \sum_{\substack{v\in\mathcal{V}_i \\ u\in\mathcal{V}_i, u\neq v}} \left[ \sum_{\substack{j\in M(v) \\ j\neq i}} \mathrm{sgn}(j-i)\lambda_{i,j}^{v,3} + \sum_{\substack{j\in M(v) \\ j\neq i}} \mathrm{sgn}(j-i)c_{v,i}^{\phi_v}\lambda_{i,j}^{v,2} + \sum_{\substack{k\in M(u) \\ k\neq i}} \mathrm{sgn}(k-i)c_{u,i}^{\phi_v}\lambda_{i,k}^{u,2} \right] \phi_{v,i} \tag{19}$$

The matrix $X_i$ is a $k \times 4$ matrix, where $k$ is the number of samples in $\mathcal{S}_i$. Each row $j$ of $X_i$ corresponds to a single sample $(x_j, y_j, z_j)$ and is given by,

$$\begin{aligned} X_i^{[j,1]} &= 1 - \frac{loc_i(x_j, y_j)}{d_i} \\ X_i^{[j,2]} &= \frac{loc_i(x_j, y_j)}{d_i} \\ X_i^{[j,3]} &= -\frac{1}{6d_i}(loc_i(x_j, y_j))^3 + \frac{1}{2}(loc_i(x_j, y_j))^2 \ - \ \frac{1}{3}d_i loc_i(x_j, y_j) \\ X_i^{[j,4]} &= \frac{1}{6d_i}(loc_i(x_j, y_j))^3 - \frac{1}{6}d_i loc_i(x_j, y_j). \end{aligned}$$

The vector $z_i$ is a $k$-vector, where the $j^{th}$ component is the value of the corresponding sample, $z_j$.

We define the vector $\Gamma_i$ with two components for each vertex $v \in \mathcal{V}_i$, one corresponding to the expression involving $\theta_{v,i}$ in (18) and one for the expression involving $\phi_{v,i}$ in (19),

$$\begin{aligned} \Gamma_i^{\theta_v} &= \sum_{\substack{j\in M(v) \\ j\neq i}} \mathrm{sgn}(j-i)\lambda_{i,j}^{v,1} + \sum_{\substack{j\in M(v) \\ j\neq i}} \mathrm{sgn}(j-i)c_{v,i}^{\theta_v}\lambda_{i,j}^{v,2} + \sum_{\substack{k\in M(u) \\ k\neq i}} \mathrm{sgn}(k-i)c_{u,i}^{\theta_v}\lambda_{i,k}^{u,2} \\ \Gamma_i^{\phi_v} &= \sum_{\substack{j\in M(v) \\ j\neq i}} \mathrm{sgn}(j-i)\lambda_{i,j}^{v,3} + \sum_{\substack{j\in M(v) \\ j\neq i}} \mathrm{sgn}(j-i)c_{v,i}^{\phi_v}\lambda_{i,j}^{v,2} + \sum_{\substack{k\in M(u) \\ k\neq i}} \mathrm{sgn}(k-i)c_{u,i}^{\phi_v}\lambda_{i,k}^{u,2}. \end{aligned}$$

Here $u$ corresponds to the other vertex of sector$i$, $u \neq v$.

As in the first-order estimation examples, each $q_i$ is a quadratic minimization problem and thus can be solved analytically at each node $i$. The expression for $q_i$ simplifies to

$$q_i(\Lambda_i; \mathcal{S}_i) = z_i^\mathsf{T} X_i \left(X_i^\mathsf{T} X_i\right)^{-1} X_i^\mathsf{T} z_i + z_i^\mathsf{T} X_i \Gamma_i - \frac{1}{4}\Gamma_i^\mathsf{T} \left(X_i^\mathsf{T} X_i\right)^{-1} \Gamma_i.$$

## B.1 Derivation of Algorithm Coefficients

To find the partial derivative of $q_i$ with respect to $\lambda_{i,j}^{v,1}$, $\lambda_{i,j}^{v,2}$, and $\lambda_{i,j}^{v,3}$, we first rewrite $q_i$ in terms of these variables. It is straightforward to show that $q_i$ can be expressed as,

$$q_i(\Lambda_i, \Theta_i; \mathcal{S}_i) \;=\; \left(\alpha_i^{v,1}\lambda_{i,j}^{v,1} \;+\; \alpha_i^{v,2}\lambda_{i,j}^{v,2} \;+\; \alpha_i^{v,3}\lambda_{i,j}^{v,3}\right)^2 \;- 2\alpha_i^{v,1}\alpha_i^{v,3}\lambda_{i,j}^{v,1}\lambda_{i,j}^{v,3} \;+\; \alpha_i^{v,4}.$$

The coefficients $\alpha$ are determined by values of the other variables in $\Lambda_i$ and the locations and values of samples.

# C Distributed Algorithm Pseudocode

Algorithm 1 details the distributed algorithm described in Section 4.3.

**Algorithm 1:** Distributed Optimization Algorithm with Pairwise Communication

**state**
  neighbors, initially neighbors $= \{i\}$
  *myLambdas*, initially $\perp$
  busy, initially $\perp$
  samples, initially $\emptyset$
  requestQueue, initially empty
  notificationQueue, initially empty
  notified, initially $\emptyset$

**function** `init`()
  `notifyLeaders`(neighbors)

**function** `notifyLeaders`(N)
  **for** $j \in N \setminus$ *notified* **do**
    **if** $j < i$ **then**
      notified $\leftarrow$ notified $\cup \{j\}$
      `send`($j$, NOTIFY)

**function** `processQueue`()
  **while** *requestQueue not empty* **do** (follower)
    $\langle j, coeffs \rangle \leftarrow$ pop(requestQueue)
    update *myLambdas* according to Eq. (14), *myLambdas* and *coeffs*
    **if** *myLambdas have changed* **then**
      *coeffs* $\leftarrow$ coefficients according to Appendix A.2
      `send`($j$, *coeffs*)
      `notifyLeaders`(neighbors $\setminus \{j\}$)
  **if** *notificationQueue not empty* **then**
    $j =$ pop(notificationQueue)
    busy $\leftarrow j$
    *coeffs* $\leftarrow$ coefficients according to Appendix A.2
    `send`($j$, *coeffs*)

**on** $\texttt{addSample}_i(x, y, z)$
  samples $\leftarrow$ samples $\cup \{(x, y, z)\}$
  `notifyLeaders`(neighbors)

**on** $\texttt{delSample}_i(x, y, z)$
  samples $\leftarrow$ samples $\setminus \{(x, y, val)\}$
  `notifyLeaders`(neighbors)

**on** $\texttt{addNeighbor}_i(j)$
  neighbors $\leftarrow$ neighbors $\cup \{j\}$
  **for** *vertex v belonging to i and j* **do**
    $myLambdas(j, v) \leftarrow 0$
  `notifyLeaders`(neighbors)

**on** $\texttt{delNeighbor}_i(j)$
  neighbors $\leftarrow$ neighbors $\setminus \{j\}$
  **for** *vertex v belonging to i and j* **do**
    $myLambdas(j, v) \leftarrow \perp$
  `notifyLeaders`(neighbors)
  **if** *busy* $= \perp$ **then** `processQueue`()

**on** $\texttt{recv}_i(j, coeffs_j)$
  **if** $i < j$ **then** (I'm the leader)
    update *myLambdas* according to Eq. (14)
    *busy* $\leftarrow \perp$
    `processQueue`()
  **else** ($j$ is the leader)
    notified $\leftarrow$ notified $\setminus \{j\}$
    push(requestQueue, $\langle j, coeffs_j \rangle$)
    **if** *busy* $= \perp$ **then** `processQueue`()

**on** $\texttt{recv}_i(j, \langle \text{NOTIFY} \rangle)$
  push(notificationQueue, $j$)
  **if** *busy* $= \perp$ **then** `processQueue`()

# D Convergence Proof Lemmas

## D.1 Simulation Lemma

**Lemma 1 (restated)** *After GST, the distributed algorithm simulates the centralized coordinate ascent algorithm, under the mapping $\mathcal{F}$.*

In order to prove Lemma 1, we first prove Claim 1 and Lemma 3.

**Claim 1** (Simulation Initialization)**.** *The function $\mathcal{F}$ maps initial states of the distributed algorithm to valid initial states of the centralized one.*

*Proof.* The claim follows from the definition of $\mathcal{F}$, since the centralized algorithm can be initialized to an arbitrary state. □

**Lemma 3** (Simulation Step)**.** *After GST, let $state_1$ be a reachable state of the distributed algorithm where $\mathcal{F}(state_1)$ is a reachable state of the centralized algorithm, and let $state_2$ be the state of the distributed algorithm after one step. Then, the step from $\mathcal{F}(state_1)$ to $\mathcal{F}(state_2)$ is a valid step of the centralized coordinate descent algorithm.*

In order to prove this lemma, we use the following claim, which states that a node does not participate in the optimization of more than one shared variable concurrently. Each shared variable is shared by two nodes $i$ and $j$ (assume $i < j$). We call these the *owners* of the variable; the one with the lower ID ($i$) is the *leader*, and the one with the higher ID ($j$) is the *follower*.

**Claim 2.** *For any two steps $x_1 > x_2 > GST$ where node $i$ sends coefficients to node $j_1$ in step $x_1$, and node $i$ sends coefficients to node $j_2$ in $x_2$ (with $j_1, j_2 < i$), there exists a step $x_1 < y < x_2$ in which $i$ receives coefficients from $j_1$.*

*Proof.* Node $i$ sends coefficients to its follower $j_1$ (line 28), only after setting its busy flag (line 26), and will not send coefficients again until it receives one from $j_1$ (line 49). □

We now prove Lemma 3.

*Proof.* A step of the distributed algorithm affects the mapped state only if it includes an update of a shared variable by its follower. We map all other steps to empty steps of the simulated centralized algorithm (i.e., the centralized algorithm does not perform a step).

For a step $x$ in which a node $i$ updates one of its shared variables $\lambda$, for which it is the follower, the update function is the same update equation as that of the centralized algorithm. It is therefore left to prove that the distributed algorithm uses the same parameters as the centralized one. The update of a shared variable is a function of the shared variables of the nodes that own it, for example consider the update steps in Appendix A for the road network example and for the two dimensional region. We therefore have to prove that the shared variables owned by the nodes $i$ and $j$, as used in the step, are the same as they were at the end of step $x - 1$.

When a leader $i$ sends the coefficients to its follower $j$, it moves to a busy state by setting its busy flag, and deferring all message processing by pushing incoming messages to queues. Specifically, (1) it does not perform follower steps, and therefore the state of the shared variables for which it is a follower does not change, and (2) it does not send new coefficients as a leader, nor does it have pending exchanges (Claim 2), therefore no other node changes the shared variables for which it is a leader. Therefore, when the follower $j$ processes the coefficients received from $i$, the $i$-side shared variables on which the coefficients are based on are the same as those of the centralized algorithm.

When the follower $j$ processes the coefficients received from $i$, the shared variables for which it is a leader are the same as those sent by its followers (otherwise it would still be busy), therefore they are the same as those of the centralized algorithm. The shared variables for which it's a follower are also the same as those of the centralized algorithm, since it updates them before sending (line 19).

Therefore, a follower node updates its shared variable using coefficients generated from shared variables of step $x - 1$. The transition of the mapped states is therefore a transition that the centralized coordinate descent algorithm would make if it chose to optimize that $\lambda$. □

Lemma 1 follows directly from Claim 1 and Lemma 3.

### D.2 Essentially Cyclic Rule Lemma

**Lemma 2 (restated)** *For every execution of the distributed algorithm, the simulation under $\mathcal{F}$ yields an execution of the centralized coordinate descent algorithm that follows an essentially cyclic policy.*

**Definition 2** (Stable shared variable)**.** *A shared variable $\lambda$ is* stable *at time $t$ if its value, as sent by the latest message from its follower, is the optimal one considering the samples taken by its owners, and the other shared variables they have at $t$ (as sent by the latest messages from followers before $t$).*

**Lemma 4.** *A shared variable $\lambda$ that is not stable at $t$, will take a pairwise step after at most $maxDegree \times maxChain \times 2 \times \Delta$ time units.*

*Proof.* Initially, before the network is connected, there are no shared variables. Then, if a shared variable is not stable, one of the nodes sharing it had a state change — its neighbor set, the values of its other shared variables, or its sample set. Once the state has changed, the station notified the leader of $\lambda$ with the `notifyLeaders` function. The leader receives the message at most $\Delta$ later and adds the appropriate neighbor ID to its notificationQueue (line 56) The leader will process the notification once it cleared previous notifications. To clear each notification, it sends coefficients to a follower, and waits for a reply. A follower replies to a coefficient message once it becomes not busy, i.e., after sending coefficients to a follower and getting an answer at most once. For each follower this takes at most $2\Delta$ plus the time it takes for the follower's follower to reply. There may form a chain of nodes waiting for one another, each as the follower of the next. Note that there may not be a cycle since the leader/follow relations are ID based. Therefore in each step, the ID of the follower is larger than the previous one. In summary, the leader may have to wait to each of its other followers, and for each one the length of a long chain. In total: $\text{maxDegree} \times \text{maxChain} \times 2 \times \Delta$. □

**Lemma 5.** *A pairwise step for a specific shared variable takes place every $2\delta$ time units or more.*

*Proof.* Since after GST the message latency is bounded from below by $\delta$, the minimal time for a pairwise step is $2\delta$ (leader detects a change and notifies itself (0), sends coefficients to its follower ($\delta$), and receives the follower's coefficients ($\delta$). □

We now prove Lemma 2.

*Proof.* We conclude from Lemmas 4 and 5 that the rate between pairwise steps of different vertices is at most $\frac{\text{maxDegree} \times \text{maxChain} \times 2 \times \Delta}{2\delta}$. While one shared variable takes $\text{maxDegree} \times \text{maxChain} \times 2 \times \Delta$ time to take a step, each of the other vertices in the system take at most $\lceil \frac{\text{maxDegree} \times \text{maxChain} \times 2 \times \Delta}{2\delta} \rceil$ steps. Therefore, in every $T = \lceil \frac{\text{maxDegree} \times \text{maxChain} \times 2 \times \Delta}{2\delta} \rceil$ pairwise steps, each shared variable is updated at least once, so this $T$ satisfies the definition of an essentially cyclic execution.

In the distributed algorithm, some of the shared variables (or all of them) may reach quiescence, in which case these shared variables do not take pairwise steps. This situation maps to the centralized coordinate descent algorithm as steps where the coordinates are at their optimum, and the state is not changed. □

# E Global Estimation with Vertex-Wise Communication

We present our distributed algorithm optimal piecewise linear estimation using vertex-wise communication. While we restrict the prevention to the example of estimation over a road network, it is straightforward to extend the algorithm to estimation over a two-dimensional region. Our algorithm is based on the method of coordinate ascent[2], but unlike the pairwise algorithms presented in Sections 4 and 5 that solve the dual problem, we solve the primal problem directly. We first show how to the rewrite problem (2) - (3) as an unconstrained convex optimization problem (Section E.1). We then present the details of the vertex-wise optimization algorithm that solves this unconstrained problem (Section E.2).

We note that the vertex-wise distributed estimation algorithm operates under a slightly different model than presented in Section 3. Namely, we assume that all nodes know the bound $\Delta$.

## E.1 Problem Formulation

In the distributed algorithm in Section 4, every node $i \in M(v)$ has its own copy of $\theta_{v,i}$, and the nodes converge to agreement at the optimal value of $\theta$ at vertex $v$. In the vertex-wise algorithm, we assign a single variable $\theta_v$ to each vertex $v \in vertexSet$. We denote the vector of all $\theta$ variables by $\Theta = [\theta_1, \theta_2, \ldots, \theta_m]$, where $m$ is the number of vertices. Each vertex $v \in \mathcal{V}i$ is assigned a unique local ID, either 0 or 1, and we define a mapping $\pi(i, localID)$ from each sector $i$ and local ID *localID* to the relevant global ID in $\Theta$.

We write an unconstrained convex optimization problem that is equivalent to (2) - (3) by making the continuity constraints implicit,

$$\underset{\Theta}{\text{minimize}} \quad C(\Theta; \mathcal{S}) = \sum_{i=1}^{m} C_i(\theta_{\pi(i,0)}, \theta_{\pi(i,1)}; \mathcal{S}_i). \tag{20}$$

With this reframing of the optimization problem, we are able to apply the method of coordinate descent. We describe our distributed implementation below.

## E.2 Distributed Coordinate Descent Algorithm

We first present our distributed coordinate descent algorithm in the context of a static network with a fixed set of samples. We then show how to extend the algorithm to accommodate dynamics, namely the addition and removal of segments and samples.

The objective is to solve the unconstrained optimization problem in Equation (20). This problem can be solved using the method of coordinate descent, where the coordinates are $\theta_v$, $v = 1 \ldots m$. Each $\theta_v$ is a shared variable. Every node $i \in members(v)$ stores a copy of the shared variable, and the nodes collaborate to perform the coordinate update step on it. We now explain the details of this update step.

**Coordinate Update Step** The node with the lowest ID among those in $M(v)$ acts as the leader for vertex $v$. It is responsible for performing the descent step for coordinate $\theta_v$ and communicating the results to all stations in $M(v)$. To perform the descent iteration, the leader must find $\overline{\theta}_v$ such that

$$\overline{\theta}_v = \arg\min_{\xi} C(\theta_1, \ldots \theta_{i-1}, \xi, \theta_{i+1}, \ldots, \theta_n; S).$$

[2]Coordinate descent is identical to coordinate ascent, except in each update step, we find the argument that *minimizes* the objective function with respect to the chosen coordinate.

This can be achieved by taking the partial derivative of $C$ with respect to $\theta_v$ and then finding the root of that partial derivative. The partial derivative of $C$ with respect to $\theta_v$ is

$$\frac{\partial C}{\partial \theta_v} = \sum_{i=1}^{m} \frac{\partial C_i}{\partial \theta_v} = \sum_{i \in M(v)} \frac{\partial C_i}{\partial \theta_v}. \tag{21}$$

The last equality follows from the fact that the partial derivative of $C_i$ is 0 if node $i$ is not adjacent to $v$. Therefore, $\overline{\theta}_v$ depends only on information from nodes adjacent to $v$.

Each node $i \in M(v)$ finds the partial derivative of $C_i$ with respect to $\theta_v$ and sends this partial derivative information to the leader. This information can be encapsulated in two coefficients $\alpha_i^v$ and $\beta_i^{v,localID}$ as follows. Let $X_i$ be the matrix formed from the sample locations where each row $j$ corresponds to a sample $(x_j, y_j, z_j)$ and is given by

$$X_i^{[j,:]} = \begin{bmatrix} 1 - \frac{x_j}{d_i} & \frac{loc_i(x_j,y_j)}{d_i} \end{bmatrix}.$$

We denote the entries in the matrix $X_i^\mathsf{T} X_i$ and the row vector $z_i^\mathsf{T} X_i$ as follows,

$$\begin{bmatrix} a_i^1 & a_i^2 \\ a_i^2 & a_i^1 \end{bmatrix} \triangleq X_i^\mathsf{T} X_i \qquad \begin{bmatrix} b_i^1 & b_i^2 \end{bmatrix}^\mathsf{T} \triangleq y_i^\mathsf{T} X_i.$$

The two coefficients are

$$\alpha_i^v \triangleq 2a_i^1 \qquad \beta_i^{v,localID} \triangleq \begin{cases} 2a_i^2\theta_{\pi(i,1)} - 2b_i^1 & \text{if } localId = 0 \\ 2a_i^2\theta_{\pi(i,0)} - 2b_i^2 & \text{otherwise,} \end{cases} \tag{22}$$

and the partial derivative of $C_i$ with respect to $\theta_v$ can be expressed as

$$\frac{\partial C_i}{\partial \theta_v} = \alpha_i \theta_v + \beta_i^{side}.$$

Once the leader has received coefficients from all nodes in $Mv$, it constructs the partial derivative of $C$ To find value $\overline{\theta}_v$ by summing as in (21). It then finds to the root of this partial derivative,

$$\overline{\theta}_v = \frac{\sum_{i \in M(v)} \beta_i^{v,localID}}{\sum_{i \in M(v)} \alpha_i^v}, \tag{23}$$

and sends the new value for $\theta_v$ to all nodes in $M(v)$.

**Atomic Vertex-Wise Communication** To initiate the update step, the leader sends a REQUEST message to all followers. Each follower $i$ computes its corresponding partial derivative locally and sends the coefficients to the leader in a REPLY message. When the leader receives all replies, it computes the value for the update and sends it to all followers in an UPDATE message.

When the leader sends a REQUEST message, it does not block while waiting for replies and can send and receive messages relating to both of its vertices, but when a follower responds to a REQUEST message, it blocks while it waits for the UPDATE message. This means that, while the stations in $M(v)$ are waiting for an UPDATE for vertex $v$, they cannot participate in updating their other vertices by sending REPLY messages. Note that vertices that are not adjacent to any station in $M(v)$ may continue to be updated while the stations in $M(v)$ are blocked.

**Dealing with dynamics** We now overview the algorithm flow. Changes to neighbor sets (entailing leader changes) while messages outstanding are handled with timeouts to avoid blocking on an obsolete leader. After GST, every vertex has unique leader known to all nodes adjacent to that vertex, and since all messages arrive within a known bound, no timeouts expire.

Each station $i$ with vertices $u$ and $v$ can independently detect whether its coefficients for a vertex $v$ have changed since the previous update. This can occur due to a change of $i$'s sample set, or an update of the value $\theta_u$ at the node's other vertex. When this happens, the station should notify $v$'s leader to perform an optimization step on $\theta_u$ by sending it a NOTIFY message. However, if the station has sent it a REPLY message for which it has not yet received an UPDATE, it is blocked, and therefore, it records the occurrence by setting a newUpdate flag. Then, once the UPDATE arrives, it sends the NOTIFY message. If the leader receives a NOTIFY message from any follower, it eventually sends a new REQUEST.

A REQUEST messages is only triggered by NOTIFY messages, and a NOTIFY message is only triggered by changes to the partial derivative for a vertex at a station. Therefore, if after GST, the system reaches the optimal solution to problem (20), then after a finite period of time, the system achieves quiescence and no additional messages are sent.

### Convergence

**Theorem 2.** *After GST, the values of the $\theta_u$ variables, as maintained by the stations, converge to the optimal estimate, i.e., the solution of Problem 20.*