Decentralized Quasi-Newton Methods

Mark Eisen, Aryan Mokhtari, and Alejandro Ribeiro

***Abstract*—We introduce the decentralized Broyden–Fletcher–Goldfarb–Shanno (D-BFGS) method as a variation of the BFGS quasi-Newton method for solving decentralized optimization problems. Decentralized quasi-Newton methods are of interest in problems that are not well conditioned, making first-order decentralized methods ineffective, and in which second-order information is not readily available, making second-order decentralized methods impossible. D-BFGS is a fully distributed algorithm in which nodes approximate curvature information of themselves and their neighbors through the satisfaction of a secant condition. We additionally provide a formulation of the algorithm in asynchronous settings. Convergence of D-BFGS is established formally in both the synchronous and asynchronous settings and strong performance advantages relative to existing methods are shown numerically.**

***Index Terms*—Multi-agent network, consensus optimization, quasi-Newton methods, asynchronous optimization.**

I. Introduction

DECENTRALIZED optimization involves a group of interconnected agents seeking to jointly minimize a common objective function using information that is local and partial. The agents collaborate by successively sharing information with other agents located in their communication neighborhood with the goal of eventually converging to the network-wide optimal argument. Decentralized optimization has proven effective in contexts where information is gathered by different nodes of a network, such as decentralized control [3]–[5], wireless systems [6]–[8], sensor networks [9]–[12], and large scale machine learning [13]–[15].

Although there are different formulations of decentralized optimization problems, all have in common a reliance on the distributed computability of the gradient. This property refers to the ability of each agent to compute gradients with respect to its local variable using its own variable and the variables of neighboring nodes. If this property holds, it is possible for nodes to exchange variables with neighbors, compute gradients with respect to their local variables, implement the corresponding block of a gradient descent algorithm, and proceed to a new variable exchange to repeat the process. Distributed gradient computability is sometimes inherent to the objective function [16], but more often the result of some reformulation. The latter is the case in consensus optimization problems which do not have distributedly computable gradients but can be transformed into problems where the gradients are (see Section VI). The most popular techniques for doing so are the use of penalties to enforce the consensus constraint [17]–[20] and the use of gradient ascent in the dual domain for a problem in which consensus is imposed as a constraint [9], [11], [21], [22]. Convergence rates in the dual domain have also been increased through the use of augmented Lagrangian [23]–[27].

The problem with methods that rely on distributed gradient computations is that gradient descent methods exhibit slow convergence. This limits applicability to cases where the function to be optimized is well conditioned, which in practice implies arguments with low dimension. Newton's method, which uses second order Hessian information, cannot be implemented directly in a distributed manner because it requires taking a matrix inverse that is not sparse. However, the Hessian can still be used to determine a better descent direction if it so happens to also be distributedly computable – which it is if the Hessian matrix has the same sparsity pattern of the network – by approximating the inverse with a series expansion. This has been done for consensus optimization problems reformulated as penalty methods [20] and for the dual problem of optimal linear flow control [28]. These approximate Newton methods exhibit faster convergence relative to their corresponding first order methods.

An alternative to the approximation of Newton steps is the use of quasi-Newton methods that rely on gradients to produce a curvature estimation to use in lieu of the Hessian inverse [29], [30]. Distributed versions of quasi-Newton methods have previously been proposed [31], some without any convergence guarantees [32], [33]. The goal of this paper is to adapt the curvature estimation technique of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton optimization method for use in distributed settings. In our distributed adaptation, unlike in [31], we focus on maintaining the global secant condition—the central property of centralized quasi-Newton methods. This adaptation leads to the development of the distributed (D-)BFGS method that we show can be implemented with nodes that operate either synchronously or asynchronously. We further prove convergence for both methods in the case of convex function and establish a linear convergence rate for the case of strongly convex functions in both, the synchronous and the asynchronous formulations. The advantages of D-BFGS relative to

Manuscript received April 21, 2016; revised November 22, 2016 and January 23, 2017; accepted January 28, 2017. Date of publication February 9, 2017; date of current version March 14, 2017. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Kostas D. Berberidis. This work was supported in part by the National Science Foundation under CAREER CCF-0952867 and in part by the U.S. Office of Naval Research under Grant N00014-12-1-0997. This paper was presented in part at the IEEE 55th Conference on Decision and Control, Las Vegas, NV, USA, 2016, and in part at the IEEE GlobalConference on Signal and Information Processing (GlobalSIP), Washington, DC, USA, December 2016.

The authors are with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA19104 USA (e-mail: maeisen@seas.upenn.edu; aryanm@seas.upenn.edu; aribeiro@seas.upenn.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2017.2666776

approximate Newton methods are that they do not require computation of Hessians, which can itself be expensive, and that they apply in any scenario in which gradients are distributedly computable irrespectively of the structure of the Hessian.

The paper starts with the introduction of notation and a formal statement of the gradient distributed computability property (Section II). The curvature approximation used in BFGS is then introduced (Section II-A). The fundamental observation here is that this curvature approximation is chosen to satisfy a secant condition because this is a property that the Hessian has. We then point out that the secant condition can be written distributedly as long as gradients are distributedly computable. Building on this observation we define D-BFGS as a method where the Hessian inverse is approximated by a matrix that satisfies the secant condition but whose sparsity pattern is chosen a fortiori to match the sparsity pattern of the graph (Section III). This matching of sparsity patterns guarantees that the method can be implemented in a distributed manner (Algorithm 1). The D-BFGS method requires three separate variable exchanges in each iteration. Since the time cost of this synchronization can be significant, we introduce an asynchronous version where nodes operate on their local memories which are synchronized by a communication protocol that runs on a separate clock (Section IV). In the asynchronous algorithm nodes operate with possibly outdated variables to avoid the time cost of running in synch (Algorithm 2).

Convergence properties are then established (Section V). In the case of synchronous D-BFGS we prove convergence for smooth convex functions and further establish a linear rate when the functions are strongly convex (Section V-A). For the case of asynchronous implementations we impose an upper bound in the number of iterations that it takes for the information of a node to get updated in the local memory of its neighbors. Under this hypothesis we also establish convergence for smooth convex functions and a linear rate for strongly convex functions (Section V-B). The convergence rate decreases with increasing levels of asynchronicity. The application of D-BFGS in consensus optimization problems is then explicitly discussed (Section VI). We explain how D-BFGS can be used in combination with penalty methods to obtain a quasi-Newton version of distributed gradient descent (Section VI-A) and how it can be used in the dual domain to obtain a quasi-Newton version of distributed dual ascent (Section VI-B). We close the paper with numerical results comparing the performance of D-BFGS to first order methods on various consensus problems in both the synchronous and asynchronous settings (Section VII).

II. Problem Formulation

Consider a decentralized system of n nodes, each of which has access to a local variable $\mathbf{x}_i \in \mathbb{R}^p$. Nodes are connected by a communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V} = \{1, \ldots, n\}$ and m edges $\mathcal{E} = \{(i,j) \mid i \text{ and } j \text{ are connected}\}$. We assume the graph $\mathcal{G}$ is undirected which implies $(i,j) \in \mathcal{E}$ if and only if $(j,i) \in \mathcal{E}$. Define the set n_i as the neighborhood of node i including i, i.e., $n_i = \{j \mid (j = i) \vee ((i,j) \in \mathcal{E})\}$, and the neighborhood size $m_i := |n_i|$. Further define the global variable $\mathbf{x} = [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{np}$ as the concatenation of the local variables $\mathbf{x}_i$ and, for each i, define the neighborhood variable $\hat{\mathbf{x}}_{n_i} \in \mathbb{R}^{np}$ to be a vector whose jth component equals $\mathbf{x}_j$ for $j \in n_i$ and $\mathbf{0}$ otherwise. The system's goal is to find the optimal argument $\mathbf{x}^* \in \mathbb{R}^{np}$ that minimizes the smooth convex cost function $f : \mathbb{R}^{np} \to \mathbb{R}$ when the gradient components $\nabla_i f(\mathbf{x}) := \partial f(\mathbf{x})/\partial \mathbf{x}_i$ have a local structure,

$$\mathbf{x}^* := \underset{\mathbf{x} \in \mathbb{R}^{np}}{\arg\min}\ f(\mathbf{x}), \qquad \text{with } \nabla_i f(\mathbf{x}) = \nabla_i f(\hat{\mathbf{x}}_{n_i}). \tag{1}$$

Since they are functions of variables that are available in their respective neighborhoods, gradient components $\nabla_i f(\mathbf{x})$ can be evaluated at node i using only single hop communications. We study examples of network optimization problems with gradients that have this property in Section VI.

A. Gradient Descent and BFGS

The gradient property in (1) means that it is possible to implement gradient descent on $f(\mathbf{x})$ in a distributed manner whereby the ith component of $\mathbf{x}$ is updated iteratively at node i until it converges to the ith component of the optimal solution $\mathbf{x}^*$. Introduce then the time index t and the variable $\mathbf{x}(t)$ to be its value at time t and define the update

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \epsilon(t)\mathbf{d}(t), \tag{2}$$

where $\epsilon(t)$ is a scalar stepsize. For convex functions, convergence of $\mathbf{x}(t)$ to $\mathbf{x}^*$ is guaranteed if $\mathbf{d}(t)$ is a proper descent direction for which $\mathbf{d}(t)^T \nabla f(\mathbf{x}(t)) \leq 0$ and step size $\epsilon(t)$ is properly chosen. Since the negative gradient has this property [34], a natural choice is to make

$$\mathbf{d}(t) = -\nabla f(\mathbf{x}(t)) := -\mathbf{g}(t). \tag{3}$$

Using the descent direction in (3) for the update in (2) yields the gradient descent method [34]. The corresponding iterations can be written componentwise as $\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \epsilon(t)\nabla_i f(\mathbf{x}(t))$ and can be implemented in a distributed manner if the gradient $\nabla_i f(\mathbf{x}(t))$ satisfies the property in (1).

As in the centralized case, decentralized gradient descent methods are often slow to converge. In centralized systems, speed of convergence can be increased by premultiplying $\mathbf{g}(t)$ by a positive definite matrix to obtain a better descent direction. Newton's method premultiplies the gradient with the Hessian inverse $\nabla^2 f(\mathbf{x})^{-1}$ and is recommended whenever possible [20]. However, the use of Hessian inverses in a distributed implementation requires further assumptions on (1) specific to the problem of interest and leads to problem specific challenges and limitations; see, e.g., [20].

Alternatively, quasi-Newton methods approximate the objective function Hessian inverse using subsequent gradient evaluations. To be more precise, define the descent direction

$$\mathbf{d}(t) = -\mathbf{B}(t)^{-1}\mathbf{g}(t), \tag{4}$$

where $\mathbf{B}(t)$ is a symmetric positive definite matrix that serves as an approximation of the Hessian $\nabla^2 f(\mathbf{x}(t))$. Various quasi-Newton methods differ in how they define $\mathbf{B}(t)$, with the most common being the method of Broyden-Fletcher-Goldfarb-Shanno (BFGS). To formulate BFGS begin by defining the variable variation $\mathbf{v}(t)$ and the gradient variation $\mathbf{r}(t)$ vectors,

$$\mathbf{v}(t) = \mathbf{x}(t+1) - \mathbf{x}(t), \qquad \mathbf{r}(t) = \mathbf{g}(t+1) - \mathbf{g}(t). \tag{5}$$

Observe that $\mathbf{v}(t)$ and $\mathbf{r}(t)$ capture differences of two consecutive variables and gradients, respectively, evaluated at steps $t+1$ and t. At each iteration, we select a new Hessian approximation $\mathbf{B}(t+1)$ that satisfies the secant condition $\mathbf{B}(t+1)\mathbf{v}(t) = \mathbf{r}(t)$. This condition is fundamental, as the secant condition is satisfied by the actual Hessian for small $\mathbf{v}(t)$. As this is an underdetermined system, we select $\mathbf{B}(t+1)$ such that it is closest to the previous approximation in terms of Gaussian differential entropy,

$$\mathbf{B}(t+1) = \underset{\mathbf{Z}}{\arg\min}\ \text{tr}[\mathbf{B}(t)^{-1}\mathbf{Z}] - \text{logdet}[\mathbf{B}(t)^{-1}\mathbf{Z}] - n,$$
$$\text{s.t.} \quad \mathbf{Z}\mathbf{v}(t) = \mathbf{r}(t), \quad \mathbf{Z} \succeq \mathbf{0}. \tag{6}$$

Note that we also require the next approximation to be positive semidefinite to ensure a proper descent. In order for the problem to be feasible with a positive definite solution, it is necessary to have $\mathbf{v}(t)^T\mathbf{r}(t) > 0$. This is always true when the objective function is strongly convex [35]. The closed-form solution to (6) provides the BFGS update formula

$$\mathbf{B}(t+1) = \mathbf{B}(t) + \frac{\mathbf{r}(t)\mathbf{r}(t)^T}{\mathbf{r}(t)^T\mathbf{v}(t)} - \frac{\mathbf{B}(t)\mathbf{v}(t)\mathbf{v}(t)^T\mathbf{B}(t)}{\mathbf{v}(t)^T\mathbf{B}(t)\mathbf{v}(t)}, \tag{7}$$

which shows that $\mathbf{B}(t+1)$ can be computed using the previous approximation matrix $\mathbf{B}(t)$ as well as the variable $\mathbf{v}(t)$ and gradient $\mathbf{r}(t)$ variations at step t.

The matrices $\mathbf{B}(t)$ that solve (6) – which are explicitly given in (7) – depend only on gradients that we are assuming can be computed in a distributed manner. This does *not* mean that a distributed implementation of BFGS is possible because neither $\mathbf{B}(t)$ nor $\mathbf{B}(t)^{-1}$ have a sparsity pattern to permit local evaluation of descent directions. Additionally, the computation of $\mathbf{B}(t+1)$ in (7) includes the inner product $\mathbf{r}(t)^T\mathbf{v}(t)$, which itself requires global information. It is important to note, however, is that the choice of objective function in (6) is of secondary importance to satisfying the secant condition and the secant condition *does* have a structure that allows for distributed evaluation. In the following section we resolve the issue of decentralization by introducing a variation of BFGS which modifies the Hessian approximation such that $\mathbf{d}(t)$ is computable distributedly.

Notation remark: The ith block of a vector $\mathbf{z} \in \mathbb{R}^{np}$ is denoted as $\mathbf{z}_i \in \mathbb{R}^p$, while $\mathbf{z}_{n_i} \in \mathbb{R}^{m_i p}$ denotes the components in n_i. To have global representations, we define $\hat{\mathbf{z}}_{n_i} \in \mathbb{R}^{np}$ to be the vector $\mathbf{z}_{n_i}$ padded with zeros in locations corresponding to nodes not in n_i. Likewise, for any matrix $\mathbf{A} \in \mathbb{R}^{np\times np}$, we define $\mathbf{A}_{n_i} \in \mathbb{R}^{m_i p \times m_i p}$ to be the $m_i p$ rows and columns of $\mathbf{A}$ corresponding to nodes in n_i and $\hat{\mathbf{A}}_{n_i} \in \mathbb{R}^{np\times np}$ to be the matrix $\mathbf{A}_{n_i}$ padded with zeros in other locations.

III. Decentralized BFGS

Our goal here is to develop an algorithm of the form

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \epsilon(t)\mathbf{d}_i(t), \tag{8}$$

Algorithm 1: D-BFGS method at node i.

```
Require: $\mathbf{B}^i(0), \mathbf{x}_i(0), \mathbf{g}_i(0), \mathbf{x}_{n_i}(0), \mathbf{g}_{n_i}(0)$
1: for $t = 0, 1, 2, \ldots$ do
2:     Descent from (13): $\mathbf{e}^i_{n_i}(t) = -(\mathbf{B}^i(t)^{-1} + \Gamma\mathbf{D}_{n_i})\mathbf{g}_{n_i}(t)$
3:     Exchange descent $\mathbf{e}^i_j(t)$ with neighbors $j \in n_i$
4:     Local descent from (14): $\mathbf{d}_i(t) := \sum_{j\in n_i} \mathbf{e}^j_i(t)$.
5:     Local update from (8): $\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \epsilon(t)\mathbf{d}_i(t)$
6:     Exchange $\mathbf{x}_i(t+1)$ with neighbors. Build $\mathbf{x}_{n_i}(t+1)$
7:     Compute $\mathbf{g}_i(t+1) = \nabla_i f(\mathbf{x}_{n_i})$ using property (1)
8:     Exchange $\mathbf{g}_i(t+1)$ with neighbors. Build $\mathbf{g}_{n_i}(t+1)$
9:     Compute $\tilde{\mathbf{v}}_{n_i}(t), \tilde{\mathbf{r}}_{n_i}(t), \mathbf{B}^i(t+1)$ using (9), (10), and (12):
           $\tilde{\mathbf{v}}_{n_i}(t) := \mathbf{D}_{n_i}\left[\mathbf{x}_{n_i}(t+1) - \mathbf{x}_{n_i}(t)\right]$
           $\tilde{\mathbf{r}}_{n_i}(t) := \mathbf{g}_{n_i}(t+1) - \mathbf{g}_{n_i}(t) - \gamma\tilde{\mathbf{v}}_{n_i}(t)$
       $\mathbf{B}^i(t+1) := \mathbf{B}^i(t) + \frac{\tilde{\mathbf{r}}_{n_i}(t)\tilde{\mathbf{r}}^T_{n_i}(t)}{\tilde{\mathbf{r}}^T_{n_i}(t)\tilde{\mathbf{v}}_{n_i}(t)} - \frac{\mathbf{B}^i(t)\tilde{\mathbf{v}}_{n_i}(t)\tilde{\mathbf{v}}^T_{n_i}(t)\mathbf{B}^i(t)}{\tilde{\mathbf{v}}^T_{n_i}(t)\mathbf{B}^i(t)\tilde{\mathbf{v}}_{n_i}(t)} + \gamma\mathbf{I}$
10: end for
```

where $\mathbf{x}_i$ is a variable kept at node i and $\mathbf{d}_i(t)$ is a local descent direction for node i that depends on iterates $\mathbf{x}_{n_i}(t)$. The idea to determine $\mathbf{d}_i(t)$ in the decentralized (D)-BFGS method is to let nodes locally approximate the curvature of $f(\mathbf{x})$ and those of their neighbors with a local Hessian inverse approximation. We use an update similar to (6) that maintains the secant condition while allowing for decentralized computation.

To construct such update, define the diagonal normalization matrix $\mathbf{D} \in \mathbb{R}^{np}$ whose ith block is $m_i^{-1}\mathbf{I}$ and a (small) scalar regularization parameter $\gamma > 0$. Recalling the neighborhood subscript notation, define the modified neighborhood variable and gradient variations, $\tilde{\mathbf{v}}_{n_i}(t) \in \mathbb{R}^{m_i p}$ and $\tilde{\mathbf{r}}_{n_i}(t) \in \mathbb{R}^{m_i p}$, as

$$\tilde{\mathbf{v}}_{n_i}(t) := \mathbf{D}_{n_i}\left[\mathbf{x}_{n_i}(t+1) - \mathbf{x}_{n_i}(t)\right] \tag{9}$$

$$\tilde{\mathbf{r}}_{n_i}(t) := \mathbf{g}_{n_i}(t+1) - \mathbf{g}_{n_i}(t) - \gamma\tilde{\mathbf{v}}_{n_i}(t). \tag{10}$$

The neighborhood variations in (9) and (10) are not simply local components of (5). The variable variation in (9) differs from the one in (5) by the presence of the normalizing matrix $\mathbf{D}_{n_i}$ and the gradient variation in (10) differs by the presence of the term $\gamma\tilde{\mathbf{v}}_{n_i}(t)$. Since $\tilde{\mathbf{v}}_{n_i}(t)$ and $\tilde{\mathbf{r}}_{n_i}(t)$ use only information node i can locally access through neighbors, we can compute and maintain a local Hessian approximation $\mathbf{B}^i(t) \in \mathbb{R}^{m_i p\times m_i p}$, which is updated as the solution of a local regularized version of (6),

$$\mathbf{B}^i(t+1) := \underset{\mathbf{Z}}{\arg\min}\ \text{tr}[(\mathbf{B}^i(t))^{-1}(\mathbf{Z} - \gamma\mathbf{I})]$$
$$- \ \text{logdet}[(\mathbf{B}^i(t))^{-1}(\mathbf{Z} - \gamma\mathbf{I})] - n$$
$$\text{s.t.} \quad \mathbf{Z}\tilde{\mathbf{v}}_{n_i}(t) = \mathbf{r}_{n_i}(t), \quad \mathbf{Z} \succeq \mathbf{0}. \tag{11}$$

Two properties differentiate (11) from (6): (i) The log-determinant forces $\mathbf{B}^i(t+1)$ to have eigenvalues greater than γ. (ii) The secant condition is expressed with respect to the *unmodified* neighborhood gradient variation $\mathbf{r}_{n_i}(t)$ and the *modified* neighborhood variable variation $\tilde{\mathbf{v}}_{n_i}(t)$. Property (i) is a regularization of (6) first proposed in the context of stochastic quasi-Newton methods [35]. Property (ii), while not obvious, ensures the secant condition is satisfied as shown in Proposition 1.

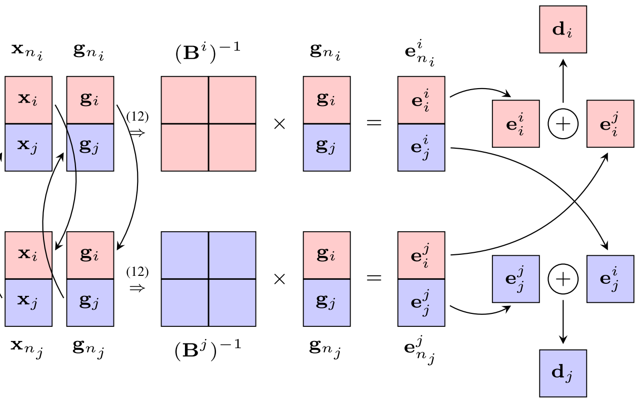

Fig. 1. D-BFGS variable flow. Nodes exchange variable and gradients – $\mathbf{x}_i$ and $\mathbf{g}_i$ sent to j and $\mathbf{x}_j$ and $\mathbf{g}_j$ sent to i – to build variable and gradient variations $\tilde{\mathbf{v}}$ and $\tilde{\mathbf{r}}$ that they use to determine local curvature matrices – $\mathbf{B}^i$ and $\mathbf{B}^j$. They then use gradients $\mathbf{g}_{n_i}$ and $\mathbf{g}_{n_j}$ to compute descent directions $\mathbf{e}^i_{n_i}$ and $\mathbf{e}^j_{n_j}$. These contain a piece to add locally – $\mathbf{e}^i_i$ stays at node i and $\mathbf{e}^j_j$ stays at node – and a piece to add at neighbors – $\mathbf{e}^i_j$ is sent to node j and $\mathbf{e}^j_i$ is sent to node i.

For the problem in (11) to have a solution, we must have $\tilde{\mathbf{v}}_{n_i}(t)^T\tilde{\mathbf{r}}_{n_i}(t) > 0$ – see Remark 1. When this condition is satisfied, the problem is not only solvable but admits the closed form solution [35, Proposition 1]

$$\mathbf{B}^i(t+1) = \mathbf{B}^i(t) + \frac{\tilde{\mathbf{r}}_{n_i}(t)\tilde{\mathbf{r}}^T_{n_i}(t)}{\tilde{\mathbf{r}}^T_{n_i}(t)\tilde{\mathbf{v}}_{n_i}(t)} - \frac{\mathbf{B}^i(t)\tilde{\mathbf{v}}_{n_i}(t)\tilde{\mathbf{v}}^T_{n_i}(t)\mathbf{B}^i(t)}{\tilde{\mathbf{v}}^T_{n_i}(t)\mathbf{B}^i(t)\tilde{\mathbf{v}}_{n_i}(t)} + \gamma\mathbf{I}. \tag{12}$$

The differences between (11) and (6) lead to corresponding differences between (12) and (7). These differences are the addition of the $\gamma\mathbf{I}$ term and the use of the variations in (9) and (10) which are not simple local decompositions of the variations in (5).

The matrices $\mathbf{B}^i$ along with an additional (and also small) regularization parameter $\Gamma > 0$ are used by node i to compute the neighborhood descent direction $\mathbf{e}^i_{n_i}(t) \in \mathbb{R}^{m_i p}$ as

$$\mathbf{e}^i_{n_i}(t) = -\left(\mathbf{B}^i(t)^{-1} + \Gamma\mathbf{D}_{n_i}\right)\mathbf{g}_{n_i}(t). \tag{13}$$

The neighborhood descent direction $\mathbf{e}^i_{n_i}(t) \in \mathbb{R}^{m_i p}$ contains components for variables of node i itself and all neighbors $j \in n_i$ – see Fig. 1. Likewise, neighboring nodes $j \in n_i$ contain a descent component of the form $\mathbf{e}^j_i(t)$. The local descent $\mathbf{d}_i(t)$ is then given by the sum of the components $\mathbf{e}^j_i(t)$ for all neighbors $j \in n_i$,

$$\mathbf{d}_i(t) = \sum_{j\in n_i} \mathbf{e}^j_i(t). \tag{14}$$

The descent direction in (14) substituted in (8) yields the D-BFGS algorithm outlined in Algorithm 1. Each node begins with an initial variable $\mathbf{x}_i(0)$, Hessian approximation $\mathbf{B}^i(0)$, and gradient $\mathbf{g}_i(0)$. Nodes exchange initial variables and gradients to construct initial neighborhood variables $\mathbf{x}_{n_i}(0)$ and gradients $\mathbf{g}_{n_i}(0)$. For each step t, nodes compute their neighborhood descent direction $\mathbf{e}_{n_i}(t)$ in Step 2 and exchange the descent elements $\mathbf{e}^i_j(t)$ with their neighbors in Step 3 to compute the local descent direction $\mathbf{d}_i(t)$ in Step 4. They use the local descent direction $\mathbf{d}_i(t)$ to update the variable $\mathbf{x}_i(t+1)$ and exchange it with their neighbors to form $\mathbf{x}_{n_i}(t+1)$ in Steps 5 and 6, respectively. They use these neighbor variables $\mathbf{x}_j(t+1)$ to compute an updated local gradient $\mathbf{g}_i(t+1)$ as in Step 7 and exchange their values in Step 8. In Step 9, nodes compute their neighborhood variable and gradient variations $\tilde{\mathbf{v}}_{n_i}(t)$ and $\tilde{\mathbf{r}}_{n_i}(t)$ that are required for computing the updated neighborhood Hessian approximation matrix $\mathbf{B}^i(t+1)$.

An alternative representation of Algorithm 1 is given in Fig. 1 where we emphasize the flow of variables among neighbors. Variable and gradient variations are exchanged – $\tilde{\mathbf{v}}_i(t)$ and $\tilde{\mathbf{r}}_i(t)$ are sent to node j and $\tilde{\mathbf{v}}_j(t)$ and $\tilde{\mathbf{r}}_j(t)$ are sent to node i – and (12) is used to compute the curvature estimation matrices – $\mathbf{B}^i(t)$ at node i and $\mathbf{B}^j(t)$ at node j. The inverses of these matrices are used to premultiply the neighborhood gradients $\mathbf{g}_{n_i(t)}$ and $\mathbf{g}_{n_j(t)}$, which necessitates an exchange of local gradients – $\mathbf{g}_i(t)$ is sent to node j and $\mathbf{g}_j(t)$ to node i. This operation results in the computation of the neighborhood descent directions – $\mathbf{e}^i_{n_i}(t)$ and $\mathbf{e}^j_{n_j}(t)$. These descent directions contain a piece to be added locally – $\mathbf{e}^i_i(t)$ stays at node i and $\mathbf{e}^j_j(t)$ stays at node – and a piece to be added at the neighboring node – $\mathbf{e}^i_j(t)$ is sent to node j and $\mathbf{e}^j_i(t)$ is sent to node i. The local descent direction $\mathbf{d}_i(t)$ is the addition of the locally computed $\mathbf{e}^i_i(t)$ and the remotely computed $\mathbf{e}^j_i(t)$ as stated in (14).

A. Secant Condition in D-BFGS

To explain the rationale of selecting $\mathbf{B}^i(t)$ as in (11) we show in the following proposition that definitions have been made so that D-BFGS satisfies the secant condition from centralized BFGS.

Proposition 1: Consider the D-BFGS method defined by (8), (13), and (14) with matrices $\mathbf{B}^i(t)$ as given in (12). Recall the notational conventions $\mathbf{x}(t) = [\mathbf{x}_1(t); \ldots; \mathbf{x}_n(t)]$ and $\mathbf{g}(t) = [\mathbf{g}_1(t); \ldots; \mathbf{g}_n(t)]$ as well as the definitions of the variable and gradient variations in (5). We can rewrite (8), (13), and (14) as

$$\mathbf{x}(t+1) = \mathbf{x}(t) - \epsilon(t)\big[\mathbf{H}(t) + \Gamma\mathbf{I}\big]\mathbf{g}(t), \tag{15}$$

with a matrix $\mathbf{H}(t)$ that satisfies the global secant condition $\mathbf{v}(t-1) = \mathbf{H}(t)\mathbf{r}(t-1)$.

Proof: Define the matrix $\mathbf{H}^i(t) \in \mathbb{R}^{np\times np}$ to be a block sparse matrix with the sparsity pattern of n_i, with a dense submatrix $\mathbf{B}^i(t)^{-1}$, i.e. $\mathbf{H}^i_{n_i}(t) = \mathbf{B}^i(t)^{-1}$. Recall that $\hat{\mathbf{x}}_{n_i}(t) \in \mathbb{R}^{np}$ stands in for the neighborhood vector $\mathbf{x}_{n_i}(t)$ padded with zeros in locations corresponding to non-neighbors. Further recall the matrix $\hat{\mathbf{D}}_{n_i} \in \mathbb{R}^{np\times np}$ amounts to the matrix $\mathbf{D}_{n_i}$ padded with zeros in locations corresponding to non-neighbors. It is apparent then that the global formulation of the neighborhood descent computed by node i from (13) is $\hat{\mathbf{e}}^i_{n_i}(t) = -[\mathbf{H}^i(t) + \Gamma\hat{\mathbf{D}}_{n_i}]\mathbf{g}(t)$. Then, summing over all nodes we have full concatenated descent

$$\mathbf{d}(t) = -\sum_{i=1}^{n}[\mathbf{H}^i(t) + \Gamma\hat{\mathbf{D}}_{n_i}]\mathbf{g}(t) = -[\mathbf{H}(t) + \Gamma\mathbf{I}]\mathbf{g}(t), \tag{16}$$

where we define $\mathbf{H}(t) := \sum_{i=1}^n \mathbf{H}^i(t)$. To see that $\mathbf{H}(t)$ satisfies the secant condition, Consider then that, by virtue of containing the inverse of a solution to (11), $\mathbf{H}^i$ satisfies the local secant relation relation $\hat{\mathbf{D}}_{n_i}\mathbf{v}(t-1) = \mathbf{H}^i(t)\mathbf{r}(t-1)$. Again summing over all nodes, we have

$$\sum_{i=1}^{n} \hat{\mathbf{D}}_{n_i}\mathbf{v}(t-1) = \sum_{i=1}^{n} \mathbf{H}^i(t)\mathbf{r}(t-1), \tag{17}$$

which implies $\mathbf{v}(t-1) = \mathbf{H}(t)\mathbf{r}(t-1)$. ■

The result in Proposition 1 explains the choices in the formulation of the problem that determines the selection of the $\mathbf{B}^i(t+1)$ matrices in (11). These matrices are chosen so that the overall update in (15) satisfies the same secant condition satisfied by (centralized) BFGS. The use of $\mathbf{v}_{n_i}(t)$ $\mathbf{r}_{n_i}(t)$ rather than the modified $\tilde{\mathbf{v}}_{n_i}(t)$ $\tilde{\mathbf{r}}_{n_i}(t)$ would result in an update that only satisifed local secant condition but not the global secant condition. The numerical benefits of this modification are explored briefly in Section VII-C.

To clarify the role and need of the regularization parameters γ and Γ we point out that if $\mathbf{B}^i(t)$ is positive semidefinite, the constants γ and Γ impose the following property on the descent matrix,

$$\frac{\Gamma}{\check{m}_i}\mathbf{I} \preceq \mathbf{B}^i(t+1)^{-1} + \Gamma\mathbf{D}_{n_i} \preceq \left(\frac{1}{\gamma} + \frac{\Gamma}{\check{m}_i}\right)\mathbf{I}, \tag{18}$$

where $\check{m}_i = \min_{j\in n_i} m_j$ and $\bar{m}_i = \max_{j\in n_i} m_j$. In particular, (18) implies that $\mathbf{B}^i(t+1)^{-1}$ is positive semidefinite. Thus, if $\mathbf{B}^i(0)^{-1}$ is positive semidefinite, the property in (18) holds for all times t. Inspection of (18) shows that the role of Γ is to prevent the algorithm from stalling if the eigenvalues of $\mathbf{B}^i(t)^{-1}$ become too small. The role of γ is to prevent the eigenvalues of $\mathbf{B}^i(t)^{-1}$ to become too large. Observe that since it is $\mathbf{B}^i(t)^{-1}$ that premultiplies $\mathbf{g}_{n_i}(t)$, simply adding a regularization factor to (7) – which one could mistakenly assume is what we do in (12)–may result in a matrix that is very far from satisfying the secant condition. The update in (12) utilizes the modified gradient and variable variations to pre-compensate for the addition of the $\gamma\mathbf{I}$ term so that the secant condition is satisfied *after* adding this term. The bounds in (18) are required for the convergence analyses in Section V. Note that the effect of various choices of both γ and Γ are reflected in the convergence constants and step size selection, not the convergence rate itself, and proper choice of these constants in practice may depend on other properties of the problem, e.g. n, p, strong convexity, etc.

Remark 1: For the problem in (11) to have a solution and the update in (12) to be valid the inner product between the neighborhood variations must be $\tilde{\mathbf{v}}_{n_i}(t)^T\tilde{\mathbf{r}}_{n_i}(t) > 0$. This condition imposes a restriction in functions that can be handled by D-BFGS. In practical implementations, however, we can check the value of this inner product and proceed to update $\mathbf{B}^i(t)$ only when it satisfies $\tilde{\mathbf{v}}_{n_i}(t)^T\tilde{\mathbf{r}}_{n_i}(t) > 0$.

IV. Asynchronous D-BFGS

Given the amount of coordination between nodes required to implement D-BFGS in Algorithm 1, we consider now the D-BFGS algorithm in the asynchronous setting, in which nodes' communications are uncoordinated with those of their neighbors. Our model for asynchronicity follows that used in [36].

Algorithm 2: Asynchronous D-BFGS method at node i.

```
Require: $\mathbf{B}^i(0), \mathbf{x}_i(0), \mathbf{g}_i(0), \mathbf{e}^i_{n_i}(0)$ [cf. (13)]
1: for $t \in T^i$ do
2:     Send $\mathbf{x}_i(t), \mathbf{g}_i(t), \mathbf{e}^i_j(t)$ to neighbors $j \in n_i$
3:     Read $\mathbf{e}^j_i(t), \mathbf{x}^i_j(t), \mathbf{g}^i_j(t)$ for $j \in n_i$ from local memory
4:     Update $\mathbf{x}_i(t+1), \mathbf{g}_i(t+1)$ [cf. (22), (23)]
5:     Compute $\tilde{\mathbf{v}}^i_{n_i}(t), \tilde{\mathbf{r}}^i_{n_i}(t), \mathbf{B}^i(t+1)$ [cf. (24), (25), (12)]:
         $\tilde{\mathbf{v}}^i_{n_i}(t) := \mathbf{D}_{n_i}\left[\mathbf{x}^i_{n_i}(t+1) - \mathbf{x}^i_{n_i}(t)\right]$
         $\tilde{\mathbf{r}}^i_{n_i}(t) := \mathbf{g}^i_{n_i}(t+1) - \mathbf{g}^i_{n_i}(t) - \gamma\tilde{\mathbf{v}}^i_{n_i}(t)$
     $\mathbf{B}^i(t+1) := \mathbf{B}^i(t) + \frac{\tilde{\mathbf{r}}^i_{n_i}(t)\tilde{\mathbf{r}}^{iT}_{n_i}(t)}{\tilde{\mathbf{r}}^{iT}_{n_i}(t)\tilde{\mathbf{v}}^i_{n_i}(t)} - \frac{\mathbf{B}^i(t)\tilde{\mathbf{v}}^i_{n_i}(t)\tilde{\mathbf{v}}^{iT}_{n_i}(t)\mathbf{B}^i(t)}{\tilde{\mathbf{v}}^{iT}_{n_i}(t)\mathbf{B}^i(t)\tilde{\mathbf{v}}^i_{n_i}(t)} + \gamma\mathbf{I}$
6:     Compute $\mathbf{e}^i_{n_i}(t+1) = -(\mathbf{B}^i(t)^{-1} + \Gamma\mathbf{D}_{n_i})\mathbf{g}^i_{n_i}(t)$ [cf. (13)]
7:     end for
```

Consider that the time indices are partitioned finely enough so that node i's primary computation, namely the $\mathcal{O}(p^3)$ computation of descent direction $\mathbf{e}^i_{n_i}(t)$, requires multiple consecutive time iterates to complete. For each node i, we then define a set $T^i \subseteq \mathbb{Z}^+$ of all time indices in which node i is available to send and receive information, i.e. not busy performing a computation.

We define for each node i a function $\pi^i(t)$ that returns, for any t, the most recent time node i was available, i.e.

$$\pi^i(t) := \max\{\hat{t} | \hat{t} < t, \hat{t} \in T^i\}. \tag{19}$$

Moreover, we define a function $\pi^i_j(t)$ that, given a time index t, returns the most recent time node j sent information received by node i by time t, or explicitly,

$$\pi^i_j(t) := \pi^j(\pi^i(t)). \tag{20}$$

We may consider $\pi^i_j(t)$ to be a delay between nodes i and j at time t. Now, in the asynchronous setting, we may use the superscript notation used to denote locally stored information to additionally signify a node's dated knowledge of a variable,

$$\mathbf{x}^i_j(t) := \mathbf{x}_j(\pi^i_j(t)), \qquad \mathbf{x}^i_{n_i}(t) = [\mathbf{x}^i_j(t)]_{j\in n_i}. \tag{21}$$

It is clear then that $\mathbf{x}^i_j(t) \neq \mathbf{x}^k_j(t)$ for any two nodes i and k at any time t. We consider as the current global variable state $\mathbf{x}(t)$ the concatenation of each node's current knowledge of its own variable, i.e. $\mathbf{x}(t) := [\mathbf{x}^i_i(t); \ldots; \mathbf{x}^n_n(t)]$. We use the same notation for gradients $\mathbf{g}^i_j(t)$ and descent directions $\mathbf{e}^i_j(t)$.

We assume at any time $t \in T^i$ that node i has finished computing a local descent direction it does three things: (i) It reads the variable, gradient, and descent directions from neighboring nodes $j \in n_i$ sent while it was busy. (ii) Node i can send its locally computed descent direction and updated variable and gradient info. (iii) It updates its local variables and gradient using the descent direction it has just finished computing as well as the descent directions received from its neighbors. To state in more explicit terms, node i performs the following update to its own block coordinate at all t:

$$\mathbf{x}^i_i(t+1) = \mathbf{x}^i_i(t) + \epsilon(t)\mathbf{d}_i(t), \tag{22}$$

where $\mathbf{d}_i(t)$ is the decent for the ith block $\mathbf{x}_i(t)$ at time t,

$$\mathbf{d}_i(t) = \begin{cases} \sum_{j \in n_i} \mathbf{e}_i^j(t) & \text{if } t \in T^i \\ \mathbf{0}. & \text{otherwise.} \end{cases} \tag{23}$$

If $t \in T^i$, node i applies all descent directions available, otherwise it does nothing. Observe that the descent direction in (23) contains descents calculated with information from time $\pi^i(t)$ and times $\pi^j(t)$ that neighbor j most recently updated its local variable.

To specify the asynchronous version of the decentralized regularized BFGS algorithm, we first reformulate the variable and gradient differences, $\tilde{\mathbf{v}}_{n_i}^i(t)$ and $\tilde{\mathbf{r}}_{n_i}^i(t)$ for the asynchronous case:

$$\tilde{\mathbf{v}}_{n_i}^i(t) = \mathbf{D}_{n_i}\left[\mathbf{x}_{n_i}^i(t+1) - \mathbf{x}_{n_i}^i(t)\right], \tag{24}$$

$$\tilde{\mathbf{r}}_{n_i}^i(t) = \mathbf{g}_{n_i}^i(t+1) - \mathbf{g}_{n_i}^i(t) - \gamma \mathbf{v}_{n_i}^i(t). \tag{25}$$

We stress that—recalling the superscript notation defined in (21)—$\mathbf{x}_{n_i}^i(t)$ is the variable state known to i at time $\pi^i(t)$, or the last time node i was available. With this redefined notation, the computation of the local asynchronous BFGS update matrix $\mathbf{B}^i(t)$ and the corresponding descent direction $\mathbf{e}_{n_i}^i(t)$ follows respectively (12) and (13) exactly as in the synchronous setting.

The complete asynchronous algorithm is outlined in Algorithm 2. Each node begins with an initial variable $\mathbf{x}_i(0)$, Hessian approximation $\mathbf{B}^i(0)$, gradient $\mathbf{g}_i(0)$, and descent component $\mathbf{e}_i^i(0)$. At each time index t, they begin by sending its variables to neighbors in Step 2 and reading the variables of neighbors $\mathbf{e}_i^j(t), \mathbf{x}_j^i(t), \mathbf{g}_i^j(t)$ in Step 3 to construct neighborhood variables. The aggregated descent direction $\mathbf{d}_i(t)$ is used to update variables $\mathbf{x}_i(t+1)$ and $\mathbf{g}_i(t+1)$ in Step 4. Then, with the updated local variable $\mathbf{x}_i(t+1)$ and gradient $\mathbf{g}_i(t+1)$, node i computes the D-BFGS variables $\tilde{\mathbf{v}}_{n_i}^i(t)$, $\tilde{\mathbf{r}}_{n_i}^i(t)$, and $\mathbf{B}^i(t+1)$ in Step 5. In Step 6, it performs the computation of the next descent direction $\mathbf{d}_{n_i}^i(t+1)$. We alternatively present a variable flow diagram between any two neighboring nodes i and j in Fig. 2. The flow differs from that presented in Fig. 1 in the fact that variables are not exchanged directly with neighbors, but instead read and written to local memories, where local exchanges then take place. The information exchanged between memories occur on asynchronous local clocks. Note that while Fig. 2 illustrates the variable flow between only two nodes, this process can be generalized for the rest of the network.

While Algorithm 2 is similar in its basic structure to the synchronous Algorithm 1, we highlight a particular difference. In the synchronous algorithm, three rounds of communication were required at each iteration of Algorithm 1 to properly communicate the dual variable, primal variable, and dual gradient information. In the asynchronous setting, naturally only a single round of communication is possible at each time iteration. As such, all coordination is removed from the algorithm and the order of computation is rearranged slightly in Algorithm 2.

As in the synchronous case, we provide a global formulation of the local descents for the aid in subsequent analysis. While (23) is an accurate physical description of how the descent is performed by node i, the asynchronous setup of (23) makes it difficult to formulate an equivalent descent direction for the global variable $\mathbf{x}(t)$. We alternatively define a virtual formulation for the global descent direction $\mathbf{d}(t) \in \mathbb{R}^{np}$ that is algorithmically equivalent to the one in (23), i.e. leads to the same result. Consider the following virtual global update at time t,

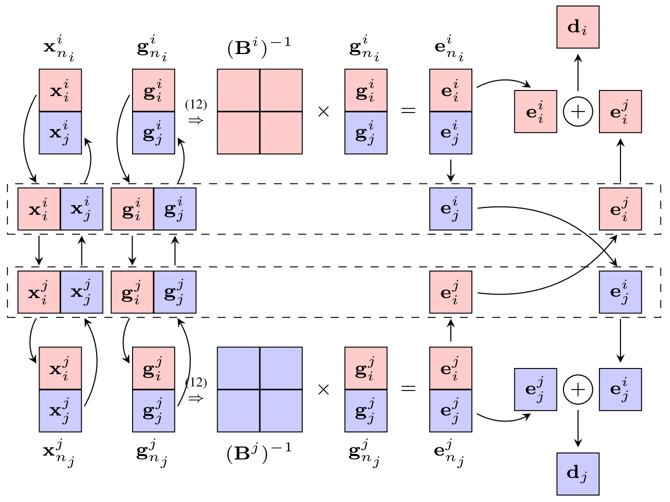

Fig. 2. Asynchronous D-BFGS variable flow. Nodes write and read variable and gradients to memory (in dashed block) – e.g. node i writes $\mathbf{x}_i^i$ and $\mathbf{g}_i^i$ and reads $\mathbf{x}_j^i$ and $\mathbf{g}_j^i$ – while variables are subsequently exchanged in memory – $\mathbf{x}_i^i$ and $\mathbf{g}_i^i$ sent to j and $\mathbf{x}_j^j$ and $\mathbf{g}_j^j$ sent to i. Nodes then build variable and gradient variations $\tilde{\mathbf{v}}$ and $\tilde{\mathbf{r}}$ to determine local curvature matrices – $\mathbf{B}^i$ and $\mathbf{B}^j$ and subsequently use gradients $\mathbf{g}_{n_i}^i$ and $\mathbf{g}_{n_j}^j$ to compute descent directions $\mathbf{e}_{n_i}^i$ and $\mathbf{e}_{n_j}^j$. Components pertaining to neighboring nodes are written to memory to be exchanged – $\mathbf{e}_j^i$ is sent to node j and $\mathbf{e}_i^j$ is sent to node i. Note that all variable exchanges between nodes occur on asynchronous local clocks.

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \epsilon(t)\mathbf{d}(t), \tag{26}$$

where the descent direction $\mathbf{d}(t) = [\mathbf{d}_1(t); \ldots; \mathbf{d}_n(t)]$ is

$$\mathbf{d}(t) = \sum_{k: t \in T^k} \hat{\mathbf{e}}_{n_k}^k(t). \tag{27}$$

In (27), we perform a descent using all directions $\hat{\mathbf{e}}_{n_k}^k(t)$ that finished being computed at time t. This is effectively equivalent to assuming that when node k finishes computing a descent direction $\mathbf{e}_{n_k}^k(t)$ at time $t \in T^k$, it is instantaneously applied to all neighboring nodes, regardless of whether or not they are busy at time t. We assert that, although node i does not physically descend with $\mathbf{e}_i^k$ at any time $t \notin T^i$, the virtual update produces the same result as the physical update at all times $t \in T^i$. This is stated in the following proposition:

Proposition 2: The virtual update described by (26) and (27) leads to the same result as the local physical update described by (22) and (23) performed by all nodes.

Proof: To show the virtual update is equivalent to the physical update for each node, we first present the coordinate-wise formulation of the virtual update (27) at time t for node i:

$$\mathbf{d}_i(t) = \sum_{k: t \in T^k} \mathbf{e}_i^k(t). \tag{28}$$

Consider two nodes $\mathbf{x}_i$ and $\bar{\mathbf{x}}_i$ who at time $t \in T^i$ are equivalent, i.e. $\mathbf{x}_i(t) = \bar{\mathbf{x}}_i(t)$, and will descend asynchronously from their neighbors. Because descent directions calculated by node i

are only calculated using information available at times $t \in T^i$, it suffices to show that $\mathbf{x}_i(t) = \bar{\mathbf{x}}_i(t)$ at all future $t_+ \in T^i$.

At time t both nodes compute $\mathbf{e}^i_{n_i}(t)$. Node $\mathbf{x}_i$ uses (23) to descend while node $\bar{\mathbf{x}}_i$ uses (28). Consider the update performed by the first node at the next available time t_+:

$$\mathbf{x}_i(t_+) = \mathbf{x}_i(t) + \epsilon\left(\sum_{j\in n_i} \mathbf{e}^j_i(t)\right). \tag{29}$$

Meanwhile, the second node adds descent components as it receives them for all times between t and t_+. At time t_+, the cumulative update performed by the second node is

$$\begin{aligned}\bar{\mathbf{x}}_i(t_+) &= \bar{\mathbf{x}}_i(t) + \epsilon \sum_{k:t+1\in T^k} \mathbf{e}^k_i(t) + \ldots + \epsilon \sum_{k:t_+\in T^k} \mathbf{e}^k_i(t)\\ &= \mathbf{x}_i(t) + \epsilon\left(\sum_{j\in n_i} \mathbf{e}^j_i(t)\right) = \mathbf{x}_i(t_+). \end{aligned}\tag{30}$$

As this is true for any node i, we can obtain that the full variable state $\mathbf{x}(t_+) = \bar{\mathbf{x}}(t_+)$. Hence, if it is true that $\mathbf{x}(t_+) = \bar{\mathbf{x}}(t_+)$, then this remains true for all future $t \in T^i$. ■

With Proposition 2 we show that the global virtual update is equivalent to the physical local update. We continue by establishing the convergence properties of D-BFGS in in both the synchronous and asynchronous settings.

V. Convergence Analysis

We analyze the convergence of D-BFGS method performed on the distributed optimization problem in (1) with objective function $f(\mathbf{x})$ with gradient components of the form $\mathbf{g}_j(\mathbf{x}) = \mathbf{g}_j(\mathbf{x}_{n_j})$. To begin, we make the following assumption on the eigenvalues of the objective function Hessian,

Assumption 1: The objective function $f(\mathbf{x})$ is twice differentiable and the eigenvalues of the Hessian are nonnegative and bounded from above by a positive constant $0 < L < \infty$,

$$\mathbf{0} \preceq \nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}. \tag{31}$$

Assumption 1 ensures the objective function f is convex. The upper bound L on the eigenvalues of the Hessian implies that the associated gradient $\mathbf{g}(\mathbf{x})$ is Lipschitz continuous with parameter L, i.e. $\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}')\| \leq L\|\mathbf{x} - \mathbf{x}'\|$. In some instances, we can be sure that the $f(\mathbf{x})$ is not just convex, but strongly convex. In these cases, we can show stronger convergence properties of D-BFGS. We therefore introduce the following second assumption.

Assumption 2: The objective function $f(\mathbf{x})$ is twice differentiable and the eigenvalues of the objective function Hessian are nonnegative and bounded from above and below by positive constants $0 < \mu < L < \infty$, i.e.

$$\mu\mathbf{I} \preceq \nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}. \tag{32}$$

In addition to Lipschitz continuity, objective functions that satisfy Assumption 2 are strongly convex with constant μ. As we will show in Section VI it is possible to derive distributed objective functions for a common class of problems that are both convex and strongly convex.

We finally make an assumption regarding the inner product of neighborhood variable and gradient variations.

Assumption 3: For all i and t, the inner product between the neighborhood modified variable and gradient vector variations is strictly positive, i.e. $\tilde{\mathbf{v}}^T_{n_i}\tilde{\mathbf{r}}_{n_i} > 0$.

This assumption is necessary to ensure all local Hessian approximations are well defined in (12). While this assumption does not always hold in practice, we use it regardless to simplify analysis. We stress that, in the case the assumption is violated, setting $\mathbf{B}^i(t+1) = \mathbf{B}^i(t)$ (See Remark 1) does not have any bearing on the proceeding analysis. We proceed to establish the convergence properties of D-BFGS in convex and strongly convex cases.

A. Synchronous Convergence

To discuss the convergence properties of the D-BFGS method in the synchronous setting, we recall that, as established in Proposition 1, the global descent of D-BFGS can be formulated as $\mathbf{x}(t+1) = \mathbf{x}(t) - \epsilon(t)[\mathbf{H}(t) + \Gamma\mathbf{I}]\mathbf{g}(t)$, where $\mathbf{H}(t)$ is a matrix built from the local Hessian inverse approximation of each node. The following lemma establishes the positive definiteness of $\mathbf{H}^i(t)$ for all i and t with specific bounds on its eigenvalues.

Lemma 1: Consider the D-BFGS method introduced in (12)–(14). Further, recall both the positive constants γ and Γ as the regularization parameters of D-BFGS and the definition of the global Hessian inverse approximation $\mathbf{H}(t) = \sum_{i=1}^n \mathbf{H}^i(t)$. The eigenvalues of the global regularized Hessian inverse approximation $\mathbf{H}(t) + \Gamma\mathbf{I}$ are uniformly bounded as

$$\Gamma\mathbf{I} \preceq \mathbf{H}(t) + \Gamma\mathbf{I} \preceq \Delta\mathbf{I}, \tag{33}$$

where $\Delta := (\Gamma + n/\gamma)$ and n is the size of network.

Proof: The lower bound on $\mathbf{H}(t) + \Gamma\mathbf{I}$ follows immediately from the fact that $\mathbf{H}(t)$ is a sum of positive semidefinite matrices and is therefore a positive semidefinite matrix with eigenvalues greater than or equal to 0. The upper bound subsequently follows from the fact that each $\mathbf{H}_i(t)$ have eigenvalues upper bounded by $1/\gamma$, as the dense submatrix $\mathbf{B}^i(t)^{-1} \preceq 1/\gamma\mathbf{I}$. Then, the sum of n such matrices recovers the upper bound in (33). ■

In Lemma 1 we show that there exists lower and upper bounds on the eigenvalues of the Hessian inverse approximation $\mathbf{H}(t) + \Gamma\mathbf{I}$, which in particular depend on the choice of regularization parameters γ and Γ. From here, it is natural to establish the convergence of D-BFGS in the case of both convex and strongly convex functions $f(\mathbf{x})$. In the former case, we show sub-linear convergence of the order of $o(1/t)$.

Theorem 1: Consider the D-BFGS method introduced in (8)–(14). If Assumptions 1 and 3 hold and the stepsize $\epsilon(t)$ satisfies $\epsilon(t) < 2\Gamma/(L\Delta^2)$, then the objective function error $f(\mathbf{x}(t)) - f(\mathbf{x}^*)$ converges to zero at least in the order of $o(1/t)$, i.e.,

$$f(\mathbf{x}(t)) - f(\mathbf{x}^*) \leq o\left(\frac{1}{t}\right). \tag{34}$$

Proof: See Appendix A. ■

With Theorem 1 we establish the sub-linear convergence of D-BFGS when the objective function is convex but not strongly convex. Observe that the step size $\epsilon(t)$ must satisfy a condition dependent upon the the choice of regularization parameters Γ and γ. Specifically, the step size is upper bounded by a term inversely proportional to Δ, which in turn grows with increasing Γ and decreasing γ. By adding a lower bound on the eigenvalues of the Hessian, thus implying strong convexity, we can establish linear convergence as we show in the following theorem.

Theorem 2: Consider the D-BFGS proposed in defined in (8)–(14). If Assumptions 2 and 3 hold and stepsize is chosen as $\epsilon(t) < 2\Gamma/(L\Delta^2)$, then the sequence of objective function values $f(\mathbf{x}(t))$ converges to the optimal value $f(\mathbf{x}^*)$ at least linearly with some constant $0 < c < 1$, i.e.

$$f(\mathbf{x}(t)) - f(\mathbf{x}^*) \leq c^t \left(f(\mathbf{x}(0)) - f(\mathbf{x}^*)\right). \tag{35}$$

Proof: See Appendix B. ■

With Theorem 2 we establish the linear convergence of D-BFGS in the synchronous setting for a strongly convex objective function. Due to strong convexity, linear convergence of the sequence $f(\mathbf{x}(t)) - f(\mathbf{x}^*) \to 0$ implies linear convergence of the variable $\|\mathbf{x}(t) - \mathbf{x}^*\| \to 0$. Again the step size is determined in part by the choice of Γ and γ. We proceed by establishing the convergence properties of asynchronous D-BFGS.

B. Asynchronous Convergence

To establish the convergence of decentralized BFGS in the asynchronous setting, it is first necessary to assume a limit to the partial asynchronicity between the nodes.

Assumption 4: There exists an asynchronicity limit $0 < B < \infty$ such that, for all i, j, and t,

$$\max\{0, t - B + 1\} \leq \pi_j^i(t) \leq t. \tag{36}$$

Assumption 4 implies a number of things. First, a node available at time t will be available again at least by the time $t + B$. Additionally, any nodes is at most B time iterations out of sync, i.e. node i's knowledge of $\mathbf{x}_j$ is at most B descent steps away from the true state of $\mathbf{x}_j$. We further assume that a node's communication delay with any other node is bounded by B. There are also important implications regarding the convergence of the physical variable update in (22) and (23) with respect to the convergence of the virtual update in (26) and (27). Specifically, if the the virtual update has converged by time t^*, any and all node's local variables will be locally convergent by time $t^* + B$. It is thus sufficient for us to show convergence properties for the virtual update in (26). We proceed to show that the asynchronous D-BFGS algorithm converges with the following theorem.

Theorem 3: Consider the asynchronous D-BFGS method proposed in (22)–(25) and (12)–(13) where $\mathbf{x}(0) = \mathbf{x}_0$. If Assumptions 1, 3, and 4 hold, then there exists a stepsize $\epsilon(t) > 0$ such that $\lim_{t\to\infty} \mathbf{g}(t) = 0$.

Proof: See Appendix C. ■

With the preceding theorem we demonstrate that in the asynchronous setting the the D-BFGS method will indeed converge to the optimal point as time goes to infinity.

We now establish a linear rate of convergence of asynchronous D-BFGS, the rate for synchronous D-BFGS, on a strongly convex function. For the remaining asynchronous analysis we adjust our definition of the asynchronous algorithm slightly to ease the analysis. Given that the discrete time indeces we assign is of our own construction to model real-world time, we can say without loss of generality that only at each time t, exactly one node k executes its descent direction $\mathbf{e}_{n_k}^k(t)$, i.e. a single term in (27) rather than a sum. This is equivalent to the time being discretized finely enough so that no two nodes complete the computation of the descent direction at the same time.

To begin, we use an idea used in analysis of incremental gradient algorithms [37] and first establish a bound on the error between the the asynchronous gradient used by active node k, $\mathbf{g}_{n_k}^k(t)$, and the kth neighborhood component of the true gradient $\mathbf{g}_{n_k}(t)$. This is stated in the following lemma.

Lemma 2: Consider the asynchronous D-BFGS algorithm proposed in (22)–(25) and (12)–(13). If Assumptions 2, 3, and 4 hold, then the norm of the gradient error $\boldsymbol{\delta}_{n_k}(t) := \mathbf{g}_{n_k}^k(t) - \mathbf{g}_{n_k}(t)$ is upper bounded as

$$\|\boldsymbol{\delta}_{n_k}(t)\| \leq 3\epsilon m_k^2 L^2 \Delta B \max_{t-2B\leq l\leq t-1} \|\mathbf{x}(l) - \mathbf{x}^*\|. \tag{37}$$

Proof: See Appendix D. ■

With this lemma, we establish that the difference between the synchronous and asynchronous gradient at time t has an upper bound that is proportional to the maximum distance between the optimal variable and the previous $t - 2B$ variables. This is important in establishing a linear convergence rate for asynchronous D-BFGS as we show in the proceeding theorem.

Theorem 4: Consider the asynchronous D-BFGS algorithm proposed in (22)–(25) and (12)–(13). If Assumptions 2, 3, and 4 hold, then with proper choice of stepsize $\epsilon(t) > 0$ such that there exits an $0 < c < 1$ such that the following holds

$$f(\mathbf{x}(t)) - f(\mathbf{x}^*) \leq c^t (f(\mathbf{x}(0)) - \mathbf{x}^*)). \tag{38}$$

Proof: See Appendix E. ■

In Theorem 4 we establish a linear convergence rate for asynchronous D-BFGS, thus demonstrating that introducing asynchronicity between neighboring nodes does not introduce deterioration in convergence rate. We proceed to show benefits of D-BFGS numerically by first introducing a common distributed optimization problem called consensus optimization.

VI. Consensus Optimization

A problem that is often solved distributedly is the minimization of the cost function $\sum_{i=1}^n f_i(\tilde{\mathbf{x}})$ with common variable $\tilde{\mathbf{x}} \in \mathbb{R}^p$ and local functions $f_i : \mathbb{R}^p \to \mathbb{R}$. This problem can be reformulated into problems that have the structure in (1). To do so, introduce local $\mathbf{x}_i \in \mathbb{R}^p$ and the aggregate $\mathbf{x} = [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{np}$. The minimization of the sum $\sum_{i=1}^n f_i(\tilde{\mathbf{x}})$ can be replaced by

$$\mathbf{x}^* := \operatorname*{arg\,min}_{\mathbf{x}\in\mathbb{R}^{np}} \sum_{i=1}^n f_i(\mathbf{x}_i), \quad \text{s.t.}\ (\mathbf{I} - \mathbf{Z})\mathbf{x} = \mathbf{0}, \tag{39}$$

where the matrix $\mathbf{Z} \in \mathbb{R}^{np\times np}$ is chosen so that the feasible variables in (39) satisfy $\mathbf{x}_i = \mathbf{x}_j$ for all i, j. A customary choice of a matrix $\mathbf{Z}$ with this property is to make it the Kronecker product $\mathbf{Z} := \mathbf{W} \otimes \mathbf{I}_p$ of a weight matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$ and the identity matrix $\mathbf{I}_p \in \mathbb{R}^{p\times p}$. The elements of the weight matrix are $w_{ij} > 0$ if $(i, j) \in \mathcal{E}$ and $w_{ij} = 0$ otherwise and the weight matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ is further assumed to satisfy

$$\mathbf{W} = \mathbf{W}^T, \quad \mathbf{W}\mathbf{1} = \mathbf{1}, \quad \text{null}\{\mathbf{I} - \mathbf{W}\} = \text{span}\{\mathbf{1}\}. \tag{40}$$

Since $\text{null}(\mathbf{I} - \mathbf{W}) = \text{span}\{\mathbf{1}\}$, it follows that for any vector $\mathbf{x} = [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{np}$ the relation $(\mathbf{I} - \mathbf{Z})\mathbf{x} = \mathbf{0}$ holds if and only if $\mathbf{x}_1 = \ldots = \mathbf{x}_n$. This means that the feasible variables in (39) indeed satisfy $\mathbf{x}_i = \mathbf{x}_j$ for all i, j and that, consequently, the problem in (39) is equivalent to the minimization of $\sum_{i=1}^n f_i(\tilde{\mathbf{x}})$.

The problem in (39) does *not* have the structure in (1), but it can be transformed into problems with that structure by the use of penalties in the primal domain, or, alternatively, ascending in the dual domain as we study in the following two sections.

A. Primal Domain Penalty Methods

To transform (39) into a formulation with the structure in (1) we incorporate the constraint as a penalty term to define the problem

$$\tilde{\mathbf{x}}^* = \underset{\mathbf{x} \in \mathbb{R}^{np}}{\arg\min} \sum_{i=1}^n f_i(\mathbf{x}_i) + \frac{1}{2\alpha}\mathbf{x}^T(\mathbf{I} - \mathbf{Z})\mathbf{x} := \underset{\mathbf{x} \in \mathbb{R}^{np}}{\arg\min}\, \phi(\mathbf{x}), \tag{41}$$

where α is a given penalty coefficient. The term $(1/2)\mathbf{x}^T(\mathbf{I} - \mathbf{Z})\mathbf{x}$ is a quadratic penalty that pushes $\mathbf{x}^*$ to the null space of $(\mathbf{I} - \mathbf{Z})^{1/2}$, meaning that $\tilde{\mathbf{x}}^*$ is pushed towards the feasible space of (39) [cf. (40)]. The difference between the solutions $\tilde{\mathbf{x}}^*$ of (41) and $\mathbf{x}^*$ of (39) is of order α; see [18].

To compute the gradient of the function $\phi(\mathbf{x})$, begin by observing that the ith component of the gradient of the penalty term is given by $\nabla_i[\mathbf{x}^T(\mathbf{I} - \mathbf{Z})\mathbf{x}/2] = \sum_{j \in n_i} w_{ij}(\mathbf{x}_i - \mathbf{x}_j)$. As the i component of the gradient of the first summand is simply $\nabla_i\phi(\mathbf{x}) = \nabla f_i(\mathbf{x})$, the ith component of the full gradient is

$$\nabla\phi(\mathbf{x})_i = \nabla f_i(\mathbf{x}_i) + \frac{1}{\alpha}\sum_{j \in n_i} w_{ij}(\mathbf{x}_i - \mathbf{x}_j). \tag{42}$$

The gradients in (42) are locally computable if neighbors exchange variables. The corresponding distributed implementation of gradient descent yields DGD [17]. In our case, (42) is a statement of the distributed computability of the gradient required in (1). We use the explicit form in (42) to compute $\mathbf{g}_i(t+1)$ in the gradient computation step of the D-BFGS algorithm (i.e. Step 7 of Algorithm 1 and Step 3 of Algorithm 2 for the synchronous and asynchronous implementations, respectively). This yields synchronous and asynchronous implementations of a quasi-Newton version of DGD. The local estimation of curvature of this method results in faster convergence – see Section VII.

B. Dual Ascent Methods

To derive the dual formulation of (39), introduce the dual variable $\boldsymbol{\nu} = [\boldsymbol{\nu}_1; \ldots; \boldsymbol{\nu}_n] \in \mathbb{R}^{np}$ composed of multipliers $\boldsymbol{\nu}_i \in \mathbb{R}^p$ associated with node i and define the Lagrangian as

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\nu}) = \sum_{i=1}^n f_i(\mathbf{x}_i) + \boldsymbol{\nu}^T(\mathbf{I} - \mathbf{Z})\mathbf{x}. \tag{43}$$

Of importance to our discussion are the primal Lagrangian minimizers that we define as $\mathbf{x}(\boldsymbol{\nu}) := \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\nu})$. Note we can write the second term in (43) as $\boldsymbol{\nu}^T(\mathbf{I} - \mathbf{Z})\mathbf{x} = \sum_{i,j \in n_i} w_{ij}\mathbf{x}_i^T(\boldsymbol{\nu}_i - \boldsymbol{\nu}_j)$. Using this fact we conclude that the components $\mathbf{x}_i(\boldsymbol{\nu})$ of the Lagrangian minimizer $\mathbf{x}(\boldsymbol{\nu})$ are

$$\mathbf{x}_i(\nu) = \text{argmin}\, f_i(\mathbf{x}_i) + \sum_{j \in n_i} w_{ij}\mathbf{x}_j^T(\boldsymbol{\nu}_i - \boldsymbol{\nu}_j) \tag{44}$$

The Lagrangian minimizers in (44) can be used to define the dual function $\psi(\boldsymbol{\nu}) := \mathcal{L}(\mathbf{x}(\boldsymbol{\nu}), \boldsymbol{\nu})$ and the corresponding dual problem as finding the argument that maximizes the dual function,

$$\boldsymbol{\nu}^* := \underset{\boldsymbol{\nu}}{\text{argmax}}\, \psi(\boldsymbol{\nu}) = \underset{\boldsymbol{\nu}}{\text{argmax}}\, \mathcal{L}(\mathbf{x}(\boldsymbol{\nu}), \boldsymbol{\nu}). \tag{45}$$

The importance of the optimal dual argument in (45) is that the optimal primal argument $\mathbf{x}^*$ of (39) can be recovered from the Lagrangian minimizer $\mathbf{x}(\boldsymbol{\nu}^*) := \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\nu}^*)$ if the primal functions f_i are strongly convex. Another important observation is that gradients of the dual function can be computed by evaluating the constraint slack associated with the Lagrangian minimizers. Specifically, note that $\nabla\psi(\boldsymbol{\nu}) = (\mathbf{I} - \mathbf{Z})\mathbf{x}(\boldsymbol{\nu})$. Given the block sparsity pattern of $\mathbf{Z}$, these gradients can be locally computed as

$$\nabla_i\psi(\boldsymbol{\nu}) \;=\; \mathbf{x}_i - \sum_{j \in n_i} w_{ij}\mathbf{x}_j \;=\; \sum_{j \in n_i} w_{ij}(\mathbf{x}_i - \mathbf{x}_j). \tag{46}$$

Since the gradients in (46) are functions of neighboring variables only, the distributed computability required in (39) holds for the maximization of the dual function in (45). We therefore use (46) in the computation of $\mathbf{g}_i(t+1)$ in either Step 7 of Algorithm 1 or Step 3 of Algorithm 2. This yields synchronous and asynchronous implementations of a quasi-Newton version of distributed dual ascent – see Section VII.

VII. Numerical Results

We provide numerical results of the performance of D-BFGS on the consensus problem for various objective functions and condition numbers. Simulations are initially performed with the following convex quadratic objective function of variable $\mathbf{x} \in \mathbb{R}^p$.

$$f(\mathbf{x}) := \sum_{i=1}^n \frac{1}{2}\mathbf{x}^T\mathbf{A}_i\mathbf{x} + \mathbf{b}_i^T\mathbf{x} \tag{47}$$

where $\mathbf{A}_i \in \mathbb{R}^{p \times p}$ and $\mathbf{b}_i \in \mathbb{R}^p$ define the local objective functions available to node i. We control the problems condition number by defining the matrices $\mathbf{A}_i = \text{diag}\{\mathbf{a}_i\}$. For a chosen condition number 10^η, $\mathbf{a}_i$ is a vector with its $p/2$ elements chosen randomly from the interval $[1, 10^1, \ldots, 10^{\eta/2}]$ and its last $p/2$ elements chosen randomly from the interval $[1, 10^{-1}, \ldots, 10^{-\eta/2}]$, resulting in the sum $\sum_{i=1}^n \mathbf{A}_i$ having eigenvalues in the range $[n10^{-\eta/2}, n10^{\eta/2}]$. For the vectors $\mathbf{b}_i$, the elements are chosen uniformly and randomly from the box $[0, 1]^p$. In our simulations we fix the variable dimension $p = 4$ and use a d-regular cycle for the graph, in which d is an even number and nodes are connected to their $d/2$ nearest neighbors in either direction. The others parameters such as condition number 10^η and number of nodes n are varied by simulation. The regularization parameters for BFGS are chosen to be $\gamma = 10^{-2}$

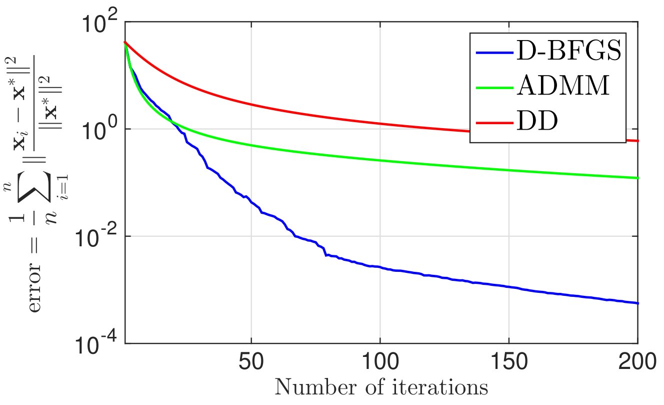

Fig. 3. Convergence of D-BFGS, ADMM, and DD in the dual domain for a quadratic objective function on the cycle graph.

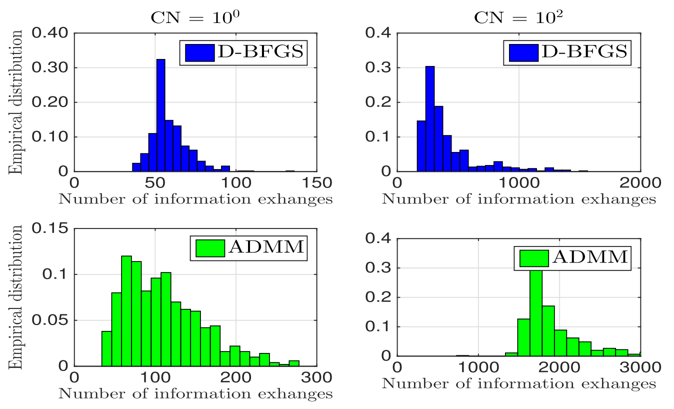

Fig. 4. Empirical distribution of number of information exchanges needed to reach error of 10^{-2} for D-BFGS and ADMM for quadratic cost function with condition numbers 1 and 100. For large condition number the gap between the methods is larger.

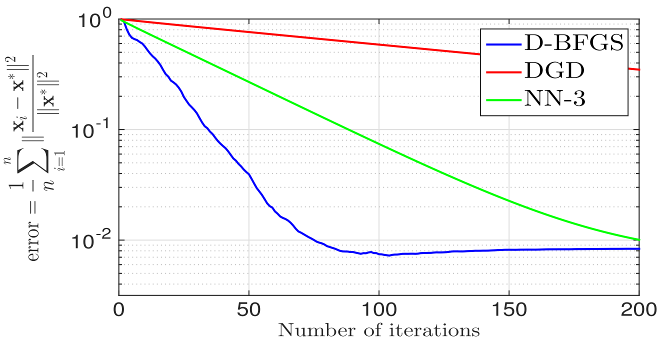

Fig. 5. Convergence of D-BFGS, DGD, and NN-3 in the primal domain for quadratic objective function. D-BFGS converges faster than DGD and NN-3 by almost factors of 10 and 3, respectivley.

and $\Gamma = 10^{-3}$. In all experiments, we choose a constant stepsize and attempt to pick the largest stepsize for which the algorithms are observed to converge.

We demonstrate results with solving (47) using both the dual and primal formulations in (45) and (41), respectively. The optimal point $\mathbf{x}^*$ is calculated exactly for the quadratic problem in (47) and we evaluate the average error as

$$\text{error}(t) := \frac{1}{n}\sum_{i=1}^{n}\frac{\|\mathbf{x}_i(t)-\mathbf{x}^*\|^2}{\|\mathbf{x}^*\|^2}. \tag{48}$$

In the dual domain, we find $\mathbf{x}(t)$ as the Lagrangian maximizer with respect to the corresponding dual variable $\boldsymbol{\nu}(t)$.

A. Synchronous Algorithms

We start in by simulating performance in the traditional synchronous setting. We simulate the performance of D-BFGS on the dual problem in (45) along with the corresponding first order dual methods, ADMM and DD [23], using respective stepsizes of 0.01, 0.002, and 0.002 on a network of size $n = 50$, connectivity $d = 4$, and condition number parameter $\eta = 2$. Fig. 3 shows the convergence rates of both algorithms in a representative simulation, specifically showing the iteration number vs the average error to the optimal primal variable. Observe that D-BFGS converges substantially faster than both first order methods, achieving an average error of 5×10^{-4} by iteration 200, while ADMM and DD just reaches average errors of 0.12 and 0.6 respectively by iteration 200.

We present a more comprehensive view of the difference in convergence times by creating an empirical distribution over a large number of trials on the cycle graph. Because D-BFGS requires twice as many communications per iteration as ADMM, Fig. 4 shows histograms of convergence times of each algorithm in terms of number of local information exchanges. Not only does D-BFGS outperform ADMM in both cases, but the difference in convergence times increases with larger condition number. In particular, there is a factor of 2 between the convergence times of D-BFGS and ADMM for a condition number of 1 and a factor of 10 for a condition number of 10^2.

We numerically evaluate the performance of D-BFGS, DGD, and NN-3 [20] on the primal problem in (41), with parameters set as $n = 100$, $d = 4$, $p = 4$, and $\eta = 2$. Note that we use NN-3 as a comparison because it requires 3 communications per iteration, same as D-BFGS. Further note that D-BFGS and NN-3 have the same computational cost per iteration, so it is indeed sufficient to compare the number of iterations until convergence. We additionally set the objective function penalty parameter $\alpha = 10^{-3}$. We choose the row stochastic weight matrix $\mathbf{W}$ to be a matrix with diagonal entries $w_{ii} = 1/2 + 1/2(d+1)$ and off diagonal entries $w_{ij} = 1/2(d+1)$ if $j \in n_j$ and 0 otherwise. The results of a sample simulation, with stepsizes of 0.3, 1, and 1 for D-BFGS, DGD, and NN-3 respectively, are shown in Fig. 5. As in the dual domain, D-BFGS converges substantially faster than its gradient descent counterpart, reaching an average error of 7.5×10^{-3} by iteration 93, while DGD reaches an average error of 0.33 by iteration 200. D-BFGS also outperforms the distributed second order method NN-3, which achieves an average error of 0.01 by iteration 200, despite incorporating actual second order Hessian information.

As an informative point of comparison, we present in Fig. 6 a comparison of D-BFGS on the primal problem with $n = 100$ nodes against a full BFGS (i.e. complete graph) on the same problem. We compare the convergence path of D-BFGS on the cycle graph with network connectivities of both $d = 4$ and $d = 10$. Observe that, while in the less connected graph

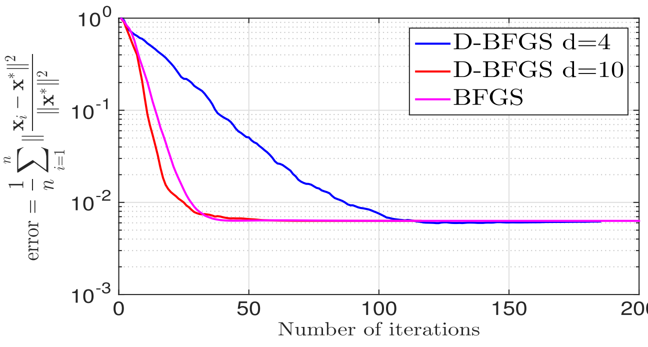

Fig. 6. Convergence of D-BFGS for network connectivity $d = 4$ and $d = 10$ compared to centralized BFGS in the primal domain for a quadratic objective function with $n = 100$ nodes. With $d = 10$, D-BFGS performs as well as centralized BFGS.

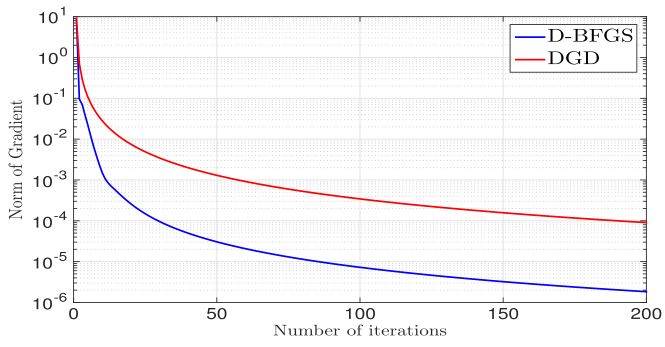

Fig. 7. Convergence of D-BFGS and DGD in the primal domain for a logistic regression problem.

D-BFGS naturally converges more slowly than the centralized method (requiring about twice as many iterations), with a network connectivity of $d = 10$, i.e. one tenth of total nodes, D-BFGS performs at the same speed as the BFGS. This is an interesting numerical result because it demonstrates that, numerically speaking, D-BFGS may be able to perform as well as BFGS if there is enough connectivity in the graph even though it doesn't have the same analytical convergence properties. The amount of communications necessary to implement D-BFGS, $d = 10$, is much lower than that needed to perform full BFGS method across 100 nodes.

B. Logistic Regression

To evaluate the performance of D-BFGS on problem of more practical interest, we additionally look at the logistic regression problem. In logistic regression, we seek to learn a linear classifier $\mathbf{x}$ that can predict the label of a data point $v_j \in \{-1, 1\}$ given a feature vector $\mathbf{u}_j \in \mathbb{R}^p$. To do so, we evaluate for a set of training samples the likelihood of a label given a feature vector as $P(v = 1|\mathbf{u}) = 1/(1 + \exp(-\mathbf{u}^T\mathbf{x}))$ and find $\mathbf{x}$ that maximizes the log likelihood over all samples. In the distributed setting, it is often assumed that the training set is large and distributed amongst n servers, with server i receiving q_i samples. It is then the case that each server i has access to a different objective function given the training samples $\{\mathbf{u}_{il}\}_{l=1}^{q_i}$ and $\{v_{il}\}_{l=1}^{q_i}$. The aggregate objective function can be defined as

$$f(\mathbf{x}) := \frac{\lambda}{2}\|\mathbf{x}\|^2 + \sum_{i=1}^{n}\sum_{l=1}^{q_i}\log[1 + \exp(-v_{il}\mathbf{u}_{il}^T\mathbf{x})], \qquad (49)$$

where the first term is a regularization term used to reduce overfitting and is parametrized by $\lambda \geq 0$.

For our simulations we generate a dataset of feature vectors $\mathbf{u}_{il}$ with label $v_{il} = 1$ from a normal distribution with mean μ and standard deviation σ_+, and with label $v_{il} = -1$ from a normal distribution with mean $-\mu$ and standard deviation σ_-. Each node i receives $q_i = 100$ samples and the regularization parameters is fixed to be $\lambda = 10^{-4}$. The feature vector parameters are set as $\mu = 3$ and $\sigma_+ = \sigma_- = 1$ to make the data linearly separable.

Se set the other parameters as in earlier simulations, i.e., $n = 100$ nodes in a $d = 4$-regular cycle with $p = 4$. The D-BFGS regularization parameters are $\Gamma = \gamma = 10^{-1}$ with stepsizes of 0.3 and 1 for D-BFGS and DGD, respectively. The resulting convergence paths are shown in Fig. 7, in this case shown with respect to the norm of the gradient. D-BFGS reaches a gradient magnitude of 1.3×10^{-6} before iteration 200 with DGD reaching a gradient magnitude of 9.1×10^{-5}.

C. Effect of Normalization

In Section III-A, we showed that the normalization factor $\mathbf{D}_{n_i}$ is included in the definition of our modified variable variation $\tilde{\mathbf{v}}_{n_i}(t)$ in (9) so that we may satisfy the global secant condition $\mathbf{v}(t-1) = \mathbf{H}(t)\mathbf{r}(t-1)$ across all n local updates. While this is a desirable property, it does not effect any of the convergence analysis in Section V. Here we illustrate the numerical benefits of including the $\mathbf{D}_{n_i}$ in the modified variable variation in (13) and to satisfy the global secant condition. Consider again the quadratic in (47) with dimension $p = 4$ and $n = 50$ nodes. For the network, we simulate star structure with all nodes connected to only one of 5 centrally connected nodes. Note that this graph structure is highly irregular relative to the cycle graph considered in previous simulations. Additionally, consider in the quadratic objective in (47) that $n/2$ local objective functions have condition number 1 and $n/2$ local functions have condition number 100.

Considering the irregularity of both the graph structure and function condition numbers, we simulate the performance of D-BFGS on the quadratic problem in the primal domain with regularization (i.e. Algorithm 1 as written) and without regularization (i.e. Algorithm 1 with $\mathbf{D}_{n_i} = \mathbf{I}$ in (9)), both with stepsize 0.01. The resulting convergence paths are shown in Fig. 8. Observe that, with the normalization factor $\mathbf{D}_{n_i}$, the method converges in roughly half the number of iterations than without the normalization factor. This illustrates that satisfying the global secant condition also provides robustness against irregularities.

D. Asynchronous Algorithms

We compare the performance of D-BFGS and DD in the asynchronous setting on dual formulation of the quadratic problem in (47). The model we use for the asynchronicity is modeled after a

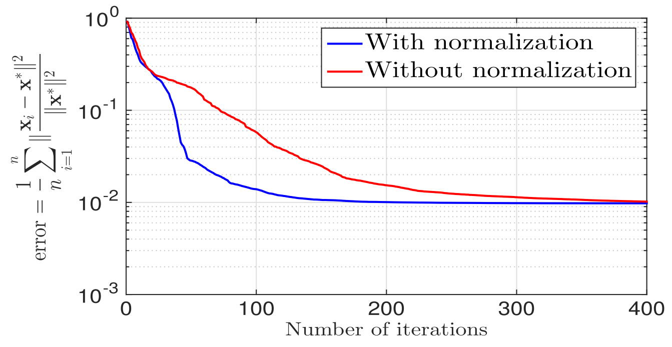

Fig. 8. Convergence of D-BFGS on star graph with and without normalization. Normalization helps convergence for irregular structure.

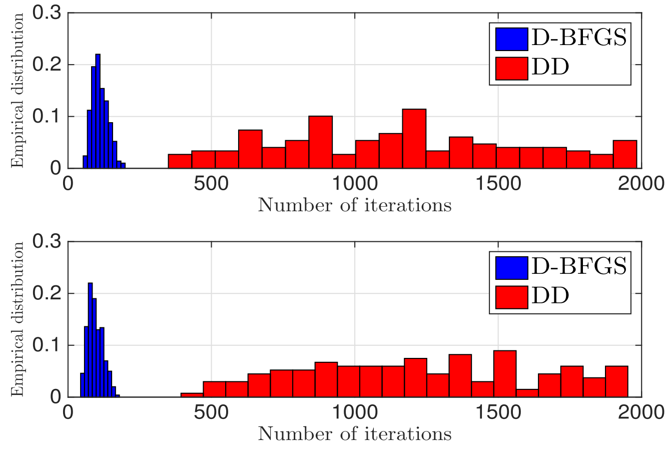

Fig. 9. Empirical distribution of convergence times for asynchronous D-BFGS and DD in the dual domain for a quadratic objective function. The level of asynchronicity varies between $\sigma = 0.1$ (top) and $\sigma = 0.3$ (bottom).

random delay phenomenon in physical communication systems that creates asynchronous local clocks between servers. Each server i's local clock begins at $t_0^i = t_0$ and selects subsequent times as $t_k^i = t_{k-1}^i + N_k^i$, where N_k^i is drawn from a normal distribution with mean μ and standard deviation σ. The standard deviation effectively controls the level of drift or asynchronicity between the nodes, i.e. a larger standard deviation will lead to nodes deviating further from the local clocks of their neighbors.

In our initial experiment, we set $n = 50$ nodes with dimension $p = 4$, condition number parameter $\eta = 1$, and network connectivity $d = 4$. The D-BFGS regularization parameters are set to be $\gamma = \Gamma = 10^{-1}$ and use the same stepsizes used in the synchronous setting (0.01 and 0.002 respectively for D-BFGS and DD). For our asynchronicity parameters we set $\mu = 1$ and select two standard deviations $\sigma = 0.1$ and $\sigma = 0.3$. The resulting empirical distributions of convergence times to error 10^{-2} are shown in Fig. 9. Observe that for $\sigma = 0.1$ D-BFGS outperforms DD, converging and order of magnitude less than DD. Further observe that, for the case of $\sigma = 0.3$, the larger drift does not result in a substantially different convergence time for either method, suggesting that the performance of the asynchronous algorithms is not very sensitive to changes in the level of asynchronicity between nodes.

Alternatively, Fig. 10 shows a histogram of convergence times for the asynchronous algorithm in the primal domain. Note that

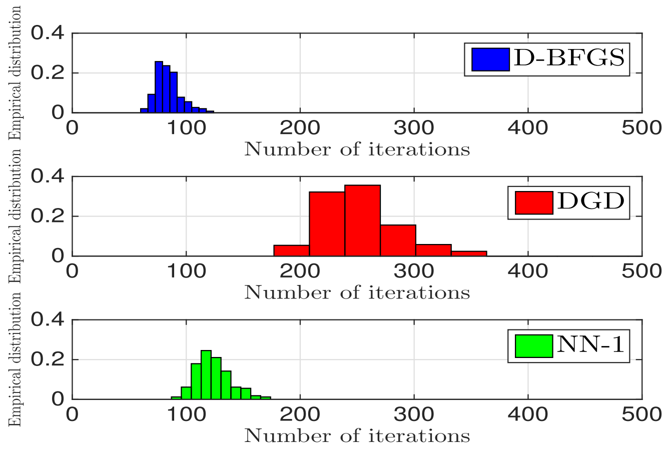

Fig. 10. Empirical distribution of convergence times for asynchronous D-BFGS, DGD, and NN-1 for quadratic objective function.

for the asynchronous setting, we compare against NN-1 rather NN-3 because only a single information exchange per iteration is possible for each algorithm. In this case, we set the regularization constants to be $\gamma = 10^{-1}$ and $\Gamma = 10^{-2}$ and use a constant step size of 0.1 for all methods. D-BFGS tends converges by iteration 100 while DGD and NN-1 converge around 250 and 150 iterations, respectively. For this setting, we observe that asynchronous D-BFGS method narrowly outperforms NN-1 without using second order information.

VIII. Conclusion

We considered the problem of general decentralized optimization, in which nodes sought to minimize a cost function while only being aware of a local strictly convex component. The problem was solved through the introduction of D-BFGS as a decentralized quasi-Newton method. In D-BFGS, each node approximates the curvature of its local cost function and its neighboring nodes to correct its descent direction. Analytical results were established in both synchronous and asynchronous versions of the algorithm. We also showed numerical results on two types of consensus optimization problems in both the dual and primal domains, in which significant improvement was observed over alternatives.

Appendix A: Proof of Theorem 1

The steps of the proof follows closely those of Proposition 1.3.3 in [38] for gradient descent for (not strongly) convex functions. Given the Lipschitz continuity of f in (31), we have for $f(\mathbf{x}(t+1))$ and constant stepsize $\epsilon(t) = \epsilon$,

$$f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t)) - \epsilon \mathbf{g}(t)^T(\mathbf{H}(t) + \Gamma\mathbf{I})\mathbf{g}(t) + \frac{L\epsilon^2}{2}\|(\mathbf{H}(t) + \Gamma\mathbf{I})\mathbf{g}(t)\|^2. \quad (50)$$

Using the lower and upper bounds on the eigenvalues of $\mathbf{H}(t) + \Gamma\mathbf{I}$ for the second and third term, respectively, then

$$f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t)) - \epsilon\|\mathbf{g}(t)\|^2\left[\Gamma - \frac{\epsilon L\Delta^2}{2}\right]. \quad (51)$$

As stated in Theorem, we choose ϵ to satisfy $\Gamma - \epsilon L\Delta^2/2 > 0$. Denote by $\mathcal{N}^*$ the set of global minima and distance between $\mathbf{x}(t)$ and the set of minima

$$d(\mathbf{x}(t), \mathcal{N}^*) := \min_{\mathbf{x}^* \in \mathcal{N}^*} \|\mathbf{x}(t) - \mathbf{x}^*\| \tag{52}$$

The convexity of $h(\mathbf{x})$ implies that for any $\mathbf{x}^* \in \mathcal{N}^*$ we have $f(\mathbf{x}(t)) \leq f(\mathbf{x}^*) + \mathbf{g}(t)^T(\mathbf{x}(t) - \mathbf{x}^*)$. This inequality in conjunction with the Cauchy-Swartz inequality implies that

$$f(\mathbf{x}(t)) \leq f(\mathbf{x}^*) + \|\mathbf{g}(t)\|\|(\mathbf{x}(t) - \mathbf{x}^*\|. \tag{53}$$

As this holds for all $\mathbf{x}^* \in \mathcal{N}^*$, we can obtain

$$f(\mathbf{x}(t)) - f(\mathbf{x}^*) \leq \|\mathbf{g}(t)\| d(\mathbf{x}(t), \mathcal{N}^*). \tag{54}$$

For notational convenience, define $e(t) := f(\mathbf{x}(t)) - f(\mathbf{x}^*)$ and assume W.L.O.G. that $d(\mathbf{x}(t), \mathcal{N}^*) \neq 0$. Combine the results in (51) and (54) and rearrange terms to obtain

$$e(t+1) \leq e(t)\left[1 - \epsilon\left(\Gamma - \frac{\epsilon L \Delta^2}{2}\right)\frac{e(k)}{d(\mathbf{x}(t), \mathcal{N}^*)^2}\right]. \tag{55}$$

The inequality in (55) indeed implies that $e(t) \leq o(1/t)$. The details of this derivation are provided in the aforementioned proposition in [38], which we remove for space considerations.

Appendix B: Proof of Theorem 2

Consider the D-BFGS update with constant stepsize $\mathbf{x}(t+1) = \mathbf{x}(t) - \epsilon(\mathbf{H}(t) + \Gamma\mathbf{I})\mathbf{g}(t)$ along with Taylor's expansion of the function f in (50). We use the upper and lower bounds on the eigenvalues of $\mathbf{H}(t) + \Gamma\mathbf{I}$ from Lemma 1 and subtract $f^* := f(\mathbf{x}^*)$ from both sides to upper bound (50) as

$$f(\mathbf{x}(t+1)) - f^* \leq f(\mathbf{x}(t)) - f^* - \epsilon\left[\Gamma - \frac{L\Delta^2\epsilon}{2}\right]\|\mathbf{g}(t)\|^2. \tag{56}$$

As a result from strong convexity, we have $\|\mathbf{g}(t)\|^2 \geq 2\mu(f(\mathbf{x}(t)) - f^*)$. If we choose $\epsilon < 2\Gamma/(L\Delta^2)$, we subsequently have after rearranging terms

$$f(\mathbf{x}(t+1)) - f^* \leq [f(\mathbf{x}(t)) - f^*]\left[1 - 2\mu\epsilon\left(\Gamma - \frac{L\Delta^2\epsilon}{2}\right)\right]. \tag{57}$$

We obtain linear convergence if $0 < 1 - \left(2\mu\Gamma\epsilon - \mu L\Delta^2\epsilon^2\right) < 1$, which holds for our choice of ϵ. Expanding (57) t times we achieve the result in (35) with $c = 1 - \left(2\mu\Gamma\epsilon - \mu L\Delta^2\epsilon^2\right)$.

Appendix C: Proof of Theorem 3

The steps of our analysis follow closely that of asynchronous gradient descent in [36, Proposition 5.1]. Consider the global virtual descent formulation in (26) and (27). As established in Proposition 2, this is equivalent to the asynchronous formulation. The Hessian eigenvalue bounds in (32) allow us to write

$$f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t)) + \epsilon\mathbf{d}(t)^T\mathbf{g}(t) + \frac{L}{2}\epsilon^2\|\mathbf{d}(t)\|^2. \tag{58}$$

We look first at bounding the second term in the summand. Recall the hat notated $\hat{\mathbf{g}}^k(t) \in \mathbb{R}^{np}$, which signifies the local vector $\mathbf{g}^k_{n_k}(t)$ padded with zeros. Further define $\mathbf{H}^k_{n_k}(t) := \mathbf{B}^k(t)^{-1} + \Gamma\mathbf{I}$. We can then substitute for $\mathbf{d}(t)$ to obtain

$$\mathbf{d}(t)^T\mathbf{g}(t) = -\mathbf{g}(t)^T \sum_{k: t \in T^k} \hat{\mathbf{H}}^k_{n_k}(t)\hat{\mathbf{g}}^k_{n_k}(t) \tag{59}$$

$$= -\sum_{k: t \in T^k} \mathbf{g}^T_{n_k}(t)\mathbf{H}^k_{n_k}(t)\mathbf{g}^k_{n_k}(t). \tag{60}$$

We proceed by adding and subtracting the asynchronous gradient $\mathbf{g}^k_{n_k}(t)$ and rearranging terms to obtain

$$\mathbf{d}(t)^T\mathbf{g}(t) = \sum_{k: t \in T^k}\Big[-\mathbf{g}^{kT}_{n_k}(t)\mathbf{H}^k_{n_k}(t)\mathbf{g}^k_{n_k}(t) + (\mathbf{g}^k_{n_k}(t) - \mathbf{g}_{n_k}(t))^T\mathbf{H}^k_{n_k}(t)\mathbf{g}^k_{n_k}(t)\Big]. \tag{61}$$

We bound the terms in summand with the eigenvalue bounds in (18) and Cauchy-Schwartz inequality, respectively.

$$\mathbf{d}(t)^T\mathbf{g}(t) \leq \sum_{k: t \in T^k}\Big[-\Gamma\|\mathbf{g}^k_{n_k}(t)\|^2 + \Delta\|\mathbf{g}^k_{n_k}(t) - \mathbf{g}_{n_k}(t)\|\|\mathbf{g}^k_{n_k}(t)\|\Big]. \tag{62}$$

Next, we bound $\|\mathbf{g}^k_{n_k}(t) - \mathbf{g}_{n_k}(t)\|$, which represents the difference between the actual and asynchronous gradients seen by node k. Split the norm into its components as

$$\|\mathbf{g}^k_{n_k}(t) - \mathbf{g}_{n_k}(t)\| \leq \sum_{j \in n_k}\|\mathbf{g}^k_j(t) - \mathbf{g}_j(t)\|. \tag{63}$$

We bound $\|\mathbf{g}^k_j(t) - \mathbf{g}_j(t)\|$ by noting that they represent components of the gradient of the global variable at times t and $\pi^k_j(t)$,

$$\|\mathbf{g}^k_j(t) - \mathbf{g}_j(t)\| = \|\nabla f(\mathbf{x}(\pi^k_j(t)))_j - \nabla f(\mathbf{x}(t))_j\|. \tag{64}$$

This can be bounded using gradients' Lipschitz continuity as

$$\|\mathbf{g}^k_j(t) - \mathbf{g}_j(t)\| \leq L\|\mathbf{x}(\pi^k_j(t))) - \mathbf{x}(t)\|. \tag{65}$$

Recall the asynchronicity bound B that limits that amount of time between $\pi^k_j(t)$ and t. We proceed to bound difference between the global and asynchronous variables as

$$\|\mathbf{x}(\pi^k_j(t))) - \mathbf{x}(t)\| \leq \epsilon\sum_{\tau = t - \pi^k_j(t)}^{t-1}\|\mathbf{d}(\tau)\| \leq \epsilon\sum_{\tau = t-B}^{t-1}\|\mathbf{d}(\tau)\|, \tag{66}$$

where the triangle inequality was used in the first inequality. Substituting (63), (65), (66) back into (62) and rearranging,

$$\mathbf{d}(t)^T\mathbf{g}(t) \leq \sum_{k: t \in T^k}\left[-\Gamma\|\mathbf{g}^k_{n_k}(t)\|^2 + \epsilon m_k L\Delta\|\mathbf{g}^k_{n_k}(t)\|\sum_{\tau = t-B}^{t-1}\|\mathbf{d}(\tau)\|\right]. \tag{67}$$

Finally, substitute the result in (67) back into (58) to obtain

$$f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t)) + \frac{\epsilon^2 L}{2}\|\mathbf{d}(t)\|^2 - \epsilon\Gamma\Big[\sum_{k:t\in T^k}\|\mathbf{g}_{n_k}^k(t)\|\Big]^2 + \epsilon^2 \bar{m} L\Delta \sum_{k:t\in T^k}\|\mathbf{g}_{n_k}^k(t)\| \sum_{\tau=t-B}^{t-1}\|\mathbf{d}(\tau)\|, \quad (68)$$

where we introduce the term $\bar{m} := \max_k\{m_k\}$ for notational convenience. The third term on the right hand side was further bounded using the triangle inequality. We simplify notation by introducing the variable $K(t) := \sum_{k:t\in T^k}\|\mathbf{g}_{n_k}^k(t)\|$. We proceed by bounding $\|\mathbf{d}(t)\|$ using the triangle inequality as

$$\|\mathbf{d}(t)\| = \|\sum_{k:t\in T^k}\hat{\mathbf{H}}_{n_k}^k(t)\hat{\mathbf{g}}_{n_k}^k(t)\| \leq \sum_{k:t\in T^k}\|\mathbf{H}_{n_k}^k(t)\mathbf{g}_{n_k}^k(t)\|.$$

The upper bound on the eigenvalues of $\mathbf{H}_{n_k}^k$ yields

$$\|\mathbf{d}(t)\| \leq \Delta \sum_{k:t\in T^k}\|\mathbf{g}_{n_k}^k(t)\| = \Delta K(t). \quad (69)$$

Replace $\|\mathbf{d}(t)\|$ in (68) by the upper bound in (69) to obtain

$$f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t)) - \left(\Gamma\epsilon - L\Delta^2\epsilon^2/2\right)K(t)^2 + L\Delta^2\epsilon^2\bar{m}K(t)\sum_{\tau=t-B}^{t-1}K(\tau). \quad (70)$$

The last term in (70) can be bounded further using the inequality $|a||b| \leq a^2 + b^2$ and rearranging terms to obtain

$$f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t)) - \left[\Gamma\epsilon - \frac{L\Delta^2\epsilon^2}{2} - BL\Delta^2\epsilon^2\bar{m}\right]K(t)^2 + L\Delta^2\epsilon^2\bar{m}\sum_{\tau=t-B}^{t-1}K(\tau)^2. \quad (71)$$

By adding the inequality in (71) or all τ between 0 and t, notice that the last term includes a summation of $K(\tau)^2$ from 0 to t, with each term repeated B times. We then obtain

$$f(\mathbf{x}(t+1)) \leq f(\mathbf{x}_0) - \left[\Gamma\epsilon - \frac{L\Delta^2\epsilon^2}{2} - 2BL\Delta^2\epsilon^2\bar{m}\right]\sum_{\tau=0}^{t}K(\tau)^2. \quad (72)$$

Choosing $0 < \epsilon < \Gamma/(L\Delta^2/2 + 2BM\bar{m}\Delta^2)$, the second term in (72) is positive. We subtract optimal f^* from both sides and, noting that $f(\mathbf{x}(t+1)) \geq f^*$, rearrange terms to obtain

$$\sum_{\tau=0}^{t}K(\tau)^2 \leq \frac{f(\mathbf{x}_0) - f^*}{\Gamma\epsilon - \frac{L\Delta^2\epsilon^2}{2} - 2BL\Delta^2\epsilon^2\bar{m}}. \quad (73)$$

Following the assumption that $f(\mathbf{x}_0) - f^*$ is bounded and positive, we conclude that the limit of $K(\tau)$ must go to zero,

$$\lim_{\tau\to\infty}K(\tau) = \lim_{\tau\to\infty}\sum_{k:\tau\in T^k}\|\mathbf{g}^k(\tau))\| = 0. \quad (74)$$

We substitute (74) into (69) to obtain $\lim_{\tau\to\infty}\|\mathbf{d}(\tau)\| = 0$ and by extension with (66) that $\lim_{\tau\to\infty}\|\hat{\mathbf{x}}^k(\tau) - \mathbf{x}(\tau)\| = 0$ for all k. The Lipschitz continuity that follows from (31) yields

$$\lim_{\tau\to\infty}\|\hat{\mathbf{g}}^k(\tau) - \mathbf{g}(\tau))\| = 0. \quad (75)$$

Finally, we conclude from (74) and the Assumption 4 that $\lim_{\tau\to\infty}\|\hat{\mathbf{g}}^k(\tau)\| = 0$ for all k and, with (75), we have $\lim_{\tau\to\infty}\|\mathbf{g}(\tau)\| = 0$. Thus, the global virtual variable $\mathbf{x}(t)$ and, by partial asynchronicity, all local $\mathbf{x}_i(t)$ are convergent.

Appendix D: Proof of Lemma 2

The steps of this proof are adapted from Section 3.2 in [37]. We begin to find an upper bound on the norm of $\boldsymbol{\delta}_{n_i}(t)$ by considering the definition along with bounds from (63)–(66).

$$\|\boldsymbol{\delta}_{n_k}(t)\| = \|\mathbf{g}_{n_k}^k(t) - \mathbf{g}_{n_k}(t)\| \leq \epsilon m_k L\sum_{\tau=t-B}^{t-1}\|\mathbf{d}(\tau)\|$$

We can subsequently bound $\|\boldsymbol{\delta}_{n_k}(t)\|$ using the bound in (69),

$$\|\boldsymbol{\delta}_{n_k}(t)\| \leq \epsilon m_k L\Delta\sum_{\tau=t-B}^{t-1}\|\mathbf{g}_{n_k}^k(\tau)\|. \quad (76)$$

Using the definition of $\boldsymbol{\delta}(t)$ and the triangle inequality on the final factor and substitute the bound from (65), we then have

$$\|\boldsymbol{\delta}_{n_k}(t)\| \leq \epsilon m_k L\Delta\sum_{\tau=t-B}^{t-1}\left[\|\mathbf{g}_{n_k}(\tau)\| + \|\mathbf{e}_{n_k}(\tau)\|\right].$$

We bound the second summand with an alternative bound of $\|\boldsymbol{\delta}_{n_k}(t)\|$. To do so, we use the bound from (63)–(65) to obtain

$$\|\boldsymbol{\delta}_{n_k}(t)\| \leq \epsilon m_k L\Delta\sum_{\tau=t-B}^{t-1}[\|\mathbf{g}_{n_k}(\tau)\| + L\sum_{j\in n_k}\|\mathbf{x}(\pi_j^k(\tau)) - \mathbf{x}(\tau)\|]. \quad (77)$$

We proceed in bounding the final term by adding and subtracting $\mathbf{x}^*$ and then using the triangle inequality to obtain

$$\|\boldsymbol{\delta}_{n_k}(t)\| \leq \epsilon m_k L\Delta\sum_{\tau=t-B}^{t-1}[\|\mathbf{g}_{n_k}(\tau)\| + L\sum_{j\in n_k}\left(\|\mathbf{x}(\pi_j^k(\tau)) - \mathbf{x}^*\| + \|\mathbf{x}^* - \mathbf{x}(\tau)\|\right)]. \quad (78)$$

Take the maximum over all time iterations between $\tau - B$ and τ to bound both terms in the final sum to obtain

$$\|\boldsymbol{\delta}_{n_k}(t)\| \leq \epsilon m_k L\Delta\sum_{\tau=t-B}^{t-1}[\|\mathbf{g}_{n_k}(\tau)\| + 2m_k L\max_{\tau-B\leq l\leq\tau}\|\mathbf{x}(l) - \mathbf{x}^*\|]. \quad (79)$$

To combine terms in the sum in (79), consider that we can bound the first summand on the right hand side using Lipschitz continuity and then similarly take the maximum over $\tau - B \leq l \leq \tau$ to obtain

$$\|\boldsymbol{\delta}_{n_k}(t)\| \leq \epsilon m_k L\Delta\sum_{\tau=t-B}^{t-1}3m_k L\max_{\tau-B\leq l\leq\tau}\|\mathbf{x}(l) - \mathbf{x}^*\|. \quad (80)$$

We obtain our result in (37) by increasing the range of the maximum to $t-2B \leq l \leq t-1$ and summing B times

$$\|\boldsymbol{\delta}_{n_k}(t)\| \leq 3\epsilon m_k^2 L^2 \Delta B \max_{t-2B\leq l\leq t-1} \|\mathbf{x}(l) - \mathbf{x}^*\|. \qquad (81)$$

Appendix E: Proof of Theorem 4

Consider the following resulting from Lipschitz continuity,

$$f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t) - \epsilon \mathbf{g}(t)^T \mathbf{d}(t) + \frac{L\epsilon^2}{2}\|\mathbf{d}(t)\|^2.$$

We substitute the asynchronous $\mathbf{d}(t) = -\hat{\mathbf{H}}_{n_k}^k(t)\hat{\mathbf{g}}_{n_k}^k(t)$–where we again notate by $\mathbf{H}_{n_k}^k(t) := \mathbf{B}^k(t)^{-1} + \Gamma\mathbf{I}$ and k is the active node at time t–and add and subtract the true gradient $\mathbf{g}(t)$ from the second two terms. After applying the upper eigenvalue bound of $\mathbf{H}_{n_k}^k(t)$ on the final term and rearranging terms, we obtain

$$\begin{aligned} f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t)) &- \epsilon\Gamma\|\mathbf{g}(t)\|^2 + \epsilon^2 L\Delta^2\|\mathbf{g}(t)\|^2 \\ &+ \epsilon\mathbf{g}(t)^T\hat{\mathbf{H}}_{n_k}^k(t)(\mathbf{g}(t) - t\hat{\mathbf{g}}_{n_k}^k(t)) \\ &+ \epsilon^2 L\Delta^2\|\mathbf{g}_{n_k}(t) - \mathbf{g}_{n_k}^k(t)\|^2. \qquad (82) \end{aligned}$$

We can then apply the Cauchy-Schwartz inequality and the upper eigenvalue bound of $\mathbf{H}^k(t)$ to the third term in the previous expression. After rearranging terms we have

$$\begin{aligned} f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t)) &- \epsilon\left(\Gamma - \epsilon L\Delta^2\right)\|\mathbf{g}(t)\|^2 \\ &+ \epsilon\Delta\|\mathbf{g}(t)\|\|\boldsymbol{\delta}(t)\| + \epsilon^2 L\Delta^2\|\boldsymbol{\delta}(t)\|^2. \qquad (83) \end{aligned}$$

We substitute the gradient error bound from (37) into (83),

$$\begin{aligned} f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t)) &- \epsilon\left(\Gamma - \epsilon L\Delta^2\right)\|\mathbf{g}(t)\|^2 \qquad (84) \\ &+ C\epsilon^2\Delta\|\mathbf{g}(t)\| \max_{t-2B\leq l\leq t-1}\|\mathbf{x}(l) - \mathbf{x}^*\| \\ &+ C^2\epsilon^4 L\Delta^2\left(\max_{t-2B\leq l\leq t-1}\|\mathbf{x}(l) - \mathbf{x}^*\|\right)^2, \end{aligned}$$

where we define the constant $C := 3m_k^2 L^2\Delta B$ for notational convenience. We use the bound given by strong convexity to bound $\|\mathbf{x}(l) - \mathbf{x}^*\| \leq \|\mathbf{g}(l)\|/\mu$ and combine terms to obtain

$$\begin{aligned} f(\mathbf{x}(t+1)) \leq f(\mathbf{x}(t)) &- \epsilon\left(\Gamma - \epsilon L\Delta^2\right)\|\mathbf{g}(t)\|^2 \qquad (85) \\ &+ \frac{C\epsilon^2\Delta}{\mu}\left(1 + C\epsilon^2 L\Delta/\mu\right)\max_{t-2B\leq l\leq t-1}\|\mathbf{g}(l)\|^2. \end{aligned}$$

Subtract $f^* := f(\mathbf{x}^*)$ from both sides of the inequality. In addition, we can bound the second and third terms respectively by the common lower and upper bounds on the gradient norm, i.e. $\|\mathbf{g}(t)\|^2 \geq 2\mu(f(\mathbf{x}(t)) - f^*)$ and $\|\mathbf{g}(t)\|^2 \leq 2L(f(\mathbf{x}(t)) - f^*)$. After substitution of these bounds, we obtain

$$\begin{aligned} &f(\mathbf{x}(t+1)) - f^* \qquad (86) \\ &\leq f(\mathbf{x}(t)) - f^* - 2\epsilon\mu\left(\Gamma - \epsilon L\Delta^2\right)(f(\mathbf{x}(t)) - f^*) \\ &\quad + \frac{2LC\epsilon^2\Delta}{\mu}\left(1 + C\epsilon^2 L\Delta/\mu\right)\max_{t-2B\leq l\leq t-1}(f(\mathbf{x}(l)) - f^*). \end{aligned}$$

To establish linear convergence, we repeat [39, Lemma 3].

Lemma 3: Consider a nonnegative sequence V_t and constants $p, q > 0$ satisfying

$$V(t+1) \leq pV(t) + q\max_{t-d(t)\leq l\leq t} V(l). \qquad (87)$$

If $p + q < 1$ and $0 \leq d(t) \leq d_{\max}$ for some constant $d_{\max} > 0$, then the sequence converges at a linear rate, i.e.

$$V(t) \leq (p+q)^{\frac{t}{1+d_{\max}}} V(0). \qquad (88)$$

We conclude the proof by restating (86) as follows:

$$\begin{aligned} f(\mathbf{x}(t+1)) - f^* &\leq p(f(\mathbf{x}(t)) - f^*) \\ &\quad + q\max_{t-2B\leq l\leq t-1}(f(\mathbf{x}(l)) - f^*), \qquad (89) \end{aligned}$$

where $p = 1 - 2\epsilon\mu(\Gamma - \epsilon L\Delta^2)$ and $q = 2LC\epsilon^2\Delta(1 + C\epsilon^2 L\Delta/\mu)/\mu$. Choosing ϵ small enough such that $p + q < 1$ holds, we have linear convergence as a result of Lemma 3.

References

[1] M. Eisen, A. Mokhtari, and A. Ribeiro, "A decentralized quasi-Newton method for dual formulations of consensus optimization," *Proc. IEEE 55th Conf. Decision and Control (CDC)*, pp. 1951–1958, 2016.

[2] M. Eisen, A. Mokhtari, and A. Ribeiro, "An asynchronous quasi-Newton method for consensus optimization," in *Proc. IEEE Global Conf. Signal Inf. Process.*, 2016.

[3] F. Bullo, J. Cortés, and S. Martinez, *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. Princeton, NJ, USA: Princeton Univ. Press, 2009.

[4] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Trans. Ind. Inform.*, vol. 9, no. 1, pp. 427–438, Feb. 2013.

[5] C. G. Lopes and A. H. Sayed, "Diffusion least-mean squares over adaptive networks: Formulation and performance analysis," *IEEE Trans. Signal Process.*, vol. 56, no. 7, pp. 3122–3136, Jul. 2008.

[6] A. Ribeiro, "Ergodic stochastic optimization algorithms for wireless communication and networking," *IEEE Trans. Signal Process.*, vol. 58, no. 12, pp. 6369–6386, Dec. 2010.

[7] G. Scutari, F. Facchinei, P. Song, D. P. Palomar, and J.-S. Pang, "Decomposition by partial linearization: Parallel optimization of multi-agent systems," *IEEE Trans. Signal Process.*, vol. 62, no. 3, pp. 641–656, Feb. 2014.

[8] A. Ribeiro, "Optimal resource allocation in wireless communication and networking," *EURASIP J. Wireless Commun. Netw.*, vol. 2012, no. 1, pp. 1–19, 2012.

[9] I. D. Schizas, A. Ribeiro, and G. B. Giannakis, "Consensus in ad hoc WSNs with noisy links—Part I: Distributed estimation of deterministic signals," *IEEE Trans. Signal Process.*, vol. 56, no. 1, pp. 350–364, Jan. 2008.

[10] U. A. Khan, S. Kar, and J. M. Moura, "DILAND: An algorithm for distributed sensor localization with noisy distance measurements," *IEEE Trans. Signal Process.*, vol. 58, no. 3, pp. 1940–1947, Mar. 2010.

[11] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," in *Proc. 3rd Int. Symp. Inf. Process. Sensor Netw.*, 2004, pp. 20–27.

[12] S. Barbarossa and G. Scutari, "Decentralized maximum-likelihood estimation for sensor networks composed of nonlinearly coupled dynamical systems," *IEEE Trans. Signal Process.*, vol. 55, no. 7, pp. 3456–3470, Jul. 2007.

[13] R. Bekkerman, M. Bilenko, and J. Langford, *Scaling Up Machine Learning: Parallel and Distributed Approaches*. Cambridge, U.K.: Cambridge Univ. Press, 2011.

[14] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning," in *Proc. 50th Annu. Allerton Conf. Commun. Control, Comput.*, 2012, pp. 1543–1550.

[15] V. Cevher, S. Becker, and M. Schmidt, "Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 32–43, Sep. 2014.

[16] C. Eksin and A. Ribeiro, "Distributed network optimization with heuristic rational agents," *IEEE Trans. Signal Process.*, vol. 60, no. 10, pp. 5396–5411, Oct. 2012.

[17] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, Jan. 2009.
[18] K. Yuan, Q. Ling, and W. Yin, "On the convergence of decentralized gradient descent," *SIAM J. Optim.*, vol. 26, no. 3, pp. 1835–1854, 2016.
[19] D. Jakovetic, J. Xavier, and J. M. Moura, "Fast distributed gradient methods," *IEEE Trans. Autom. Control*, vol. 59, no. 5, pp. 1131–1146, May 2014.
[20] A. Mokhtari, Q. Ling, and A. Ribeiro, "Network Newton distributed optimization methods," *IEEE Trans. Signal Process.*, vol. 65, no. 1, pp. 146–161, Jan. 2017.
[21] A. Makhdoumi and A. Ozdaglar, "Convergence rate of distributed ADMM over networks," 2016, arXiv:1601.00194.
[22] P. Bianchi, W. Hachem, and F. Iutzeler, "A stochastic coordinate descent primal-dual algorithm and applications to large-scale composite optimization," in *Proc. IEEE Int. Workshop Mach. Learn. Signal Process. (MLSP)*, 2014, pp. 1–6.
[23] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.
[24] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the linear convergence of the ADMM in decentralized consensus optimization," *IEEE Trans. Signal Process.*, vol. 62, no. 7, pp. 1750–1761, Apr. 2014.
[25] Q. Ling, W. Shi, G. Wu, and A. Ribeiro, "DLM: Decentralized linearized alternating direction method of multipliers," *IEEE Trans. Signal Process.*, vol. 63, no. 15, pp. 4051–4064, Aug. 2015.
[26] N. Chatzipanagiotis, D. Dentcheva, and M. M. Zavlanos, "An augmented Lagrangian method for distributed optimization," *Math. Programm.*, vol. 152, nos. 1/2, pp. 405–434, 2015.
[27] A. Mokhtari, W. Shi, Q. Ling, and A. Ribeiro, "A decentralized second-order method with exact linear convergence rate for consensus optimization," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 2, no. 4, pp. 507–522, Dec. 2016.
[28] M. Zargham, A. Ribeiro, A. Ozdaglar, and A. Jadbabaie, "Accelerated dual descent for network flow optimization," *IEEE Trans. Autom. Control*, vol. 59, no. 4, pp. 905–920, Apr. 2014.
[29] J. E. Dennis and J. J. Moré, "A characterization of superlinear convergence and its application to quasi-Newton methods," *Math. Comput.*, vol. 28, no. 126, pp. 549–560, 1974.
[30] M. J. Powell, "Some global convergence properties of a variable metric algorithm for minimization without exact line searches," *Nonlinear Programm.*, vol. 9, no. 1, pp. 53–72, 1976.
[31] D. Bajovic, D. Jakovetic, N. Krejic, and N. K. Jerinkic, "Newton-like method with diagonal correction for distributed optimization," 2015, arXiv:1509.01703.
[32] J. E. Dennis Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, vol. 16. Philadelphia, PA, USA: SIAM, 1996.
[33] S. Bolognani and S. Zampieri, "Distributed quasi-Newton method and its application to the optimal reactive power flow problem," *IFAC Proc.*, vol. 43, no. 19, pp. 305–310, 2010.
[34] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[35] A. Mokhtari and A. Ribeiro, "RES: Regularized stochastic BFGS algorithm," *IEEE Trans. Signal Process.*, vol. 62, no. 23, pp. 6089–6104, Dec. 2014.
[36] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1989.
[37] M. Gurbuzbalaban, A. Ozdaglar, and P. Parrilo, "On the convergence rate of incremental aggregated gradient algorithms," 2015, arXiv:1506.02081.
[38] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 1999.
[39] H. R. Feyzmahdavian, A. Aytekin, and M. Johansson, "A delayed proximal gradient method with linear convergence rate," in *Proc. IEEE Int. Workshop Mach. Learn. Signal Process.*, 2014, pp. 1–6.

Mark Eisen received the B.Sc. degree in electrical engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 2014. He is currently working toward the Ph.D. in the Department of Electrical and Systems Engineering, University of Pennsylvania. His research interests include distributed optimization and machine learning. In the summer of 2013, he was a research intern in the Institute for Mathematics and its Applications, University of Minnesota, Minneapolis, MN, USA. He received the Outstanding Student Presentation at the 2014 Joint Mathematics Meeting, as well as the 2016 Penn Outstanding Undergraduate Research Mentor Award.

Aryan Mokhtari received the B.Sc. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2011, and the M.Sc. degree in electrical engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 2014. Since 2012, he has been working toward the Ph.D. degree in the Department of Electrical and Systems Engineering, University of Pennsylvania. From June to August 2010, he was an intern at the Advanced Digital Sciences Center, Singapore. From June to August 2016, he was a research intern with the Big-Data Machine Learning group at Yahoo!, Sunnyvale, CA, USA. His research interests include the areas of optimization, machine learning, control, and signal processing. His current research focuses on developing methods for large-scale optimization problems.

Alejandro Ribeiro received the B.Sc. degree in electrical engineering from the Universidad de la Republica Oriental del Uruguay, Montevideo, Uruguay, in 1998, and the M.Sc. and Ph.D. degrees in electrical engineering from the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA, in 2005 and 2007. From 1998 to 2003, he was a member of the technical staff at Bellsouth Montevideo. After his M.Sc. and Ph.D. studies, in 2008 he joined the University of Pennsylvania (Penn), Philadelphia, PA, USA, where he is currently the Rosenbluth Associate Professor in the Department of Electrical and Systems Engineering. His research interests include the applications of statistical signal processing to the study of networks and networked phenomena. His focus is on structured representations of networked data structures, graph signal processing, network optimization, robot teams, and networked control. He received the 2014 O. Hugo Schuck Best Paper Award, the 2012 S. Reid Warren, Jr. Award presented by Penn's undergraduate student body for outstanding teaching, the NSF CAREER Award in 2010, and paper awards at the 2016 SSP Workshop, 2016 SAM Workshop, 2015 Asilomar SSC Conference, ACC 2013, ICASSP 2006, and ICASSP 2005. He is a Fulbright scholar and a Penn Fellow.