Decentralized Consensus Optimization with Asynchrony and Delays

Tianyu Wu, Kun Yuan, Qing Ling, Wotao Yin, and Ali H. Sayed

Abstract—We propose an asynchronous, decentralized algorithm for consensus optimization. The algorithm runs over a network in which the agents communicate with their neighbors and perform local computation.

In the proposed algorithm, each agent can compute and communicate independently at different times, for different durations, with the information it has even if the latest information from its neighbors is not yet available. Such an asynchronous algorithm reduces the time that agents would otherwise waste idle because of communication delays or because their neighbors are slower. It also eliminates the need for a global clock for synchronization.

Mathematically, the algorithm involves both primal and dual variables, uses fixed step-size parameters, and provably converges to a certain constraint on the optimization variable. Decentralized algorithms rely on delayed neighboring variables; this situation is illustrated in Fig. 1. For example, when iteration $k$ is still relying on delayed neighboring variables $\{x_{1,0}, x_{2,0}\}$, rather than the updated variables $\{x_{1,2}, x_{2,1}\}$, to update $x_{3,3}$. Therefore, both computation and communication cause delays.

A. Relationship to certain synchronous algorithms

The proposed algorithm, if running synchronously, can be algebraically reduced to PG-EXTRA [2], they solve problem (1) with a fixed step-size parameter and are typically faster.
than algorithms using diminishing step sizes. Also, both algorithms generalize EXTRA [3], which only deals with differentiable functions. However, divergence (or convergence to wrong solutions) can be observed when one runs EXTRA and PG-EXTRA in the asynchronous setting, where the proposed algorithm works correctly. The proposed algorithm in this paper must use additional dual variables that are associated with edges, thus leading to a moderate cost of updating and communicating the dual variables.

The proposed algorithm is also very different from decentralized ADMM [3]–[5] except that both algorithms can use fixed parameters. Distributed consensus methods [6], [7] that rely on fixed step-sizes can also converge fast, albeit only to approximate solutions [6], [7].

Several useful diffusion strategies [8]–[13] have also been developed for solving stochastic decentralized optimization problems where realizations of random cost functions are observed at every iteration. To keep continuous learning alive, these strategies also employ fixed step-sizes, and they converge fast to a small neighborhood of the true solution. The diffusion strategies operate on the primal domain, but they can outperform some primal-dual strategies in the stochastic setting due to the influence of gradient noise [14]. These studies focused on synchronous implementations. Here, our emphasis is on asynchronous networks, where delays are present and become critical, and also on deterministic optimization where convergence to the true solution of Problem (1) is desired.

B. Related decentralized algorithms under different settings

Our setting of asynchrony is different from randomized single activation, which is assumed for the randomized gossip algorithm [15], [16]. Their setting activates only one edge at a time and does not allow any delay. That is, before each activation, computation and communication associated with previous activations must be completed, and only one edge in each neighborhood can be activated at any time. Likewise, our setting is different from randomized multi-activation such as [17], [18] for consensus averaging, and [19], [20] for consensus optimization, which activate multiple edges each time and still do not allow any delay. These algorithms can be alternatively viewed as asynchronous algorithms running in a sequence of varying subgraphs. Since each iteration waits for the slowest agent or longest communication, a certain coordinator or global clock is needed.

Our setting is also different from [23]–[26], in which other sources of asynchronous behavior in networks are allowed, such as different arrival times of data at the agents, random on/off activation of edges and neighbors, random on/off updates by the agents, and random link failures, etc. Although the results in these works provide notable evidence for the resilience of decentralized algorithms to network uncertainties, they do not consider delays.

We also distinguish our setting from the fixed communication delay setting [27], [28], where the information passing through each edge takes a fixed number of iterations to arrive. Different edges can have different such numbers, and agents can compute with only the information they have, instead of waiting. As demonstrated in [27], this setting can be transferred into no communication delay by replacing an edge with a chain of dummy nodes. Information passing through a chain of \( r \) dummy nodes simulates an edge with a \( r \)-iteration delay. The computation in this setting is still synchronous, so a coordinator or global clock is still needed. [27], [29] consider random communication delays in their setting. However, they are only suitable for consensus averaging, not generalized for the optimization problem (1).

Our setting is identical to the setting outlined in Section 2.6 of [30], the asynchronous decentralized ADMM. Our algorithm, however, handles composite functions and avoids solving possibly complicated subproblems. The recent paper [31] also considers both random computation and communication delays. However, as the focus of that paper is a quasi-Newton method, its analysis assumes twice continuously differentiable functions and thus excludes nondifferentiable functions \( s_i \) in our problem. In fact, [31] solves a different problem and cannot be directly used to solve our Problem (1) even if its objective functions are smooth. It can, however, solve an approximation problem of (1) with either reduced solution accuracy or low speed.

C. Contributions

This paper introduces an asynchronous, decentralized algorithm for problem (1) that provably converges to an optimal solution assuming that the next update is performed by a random agent and that communication is subject to arbitrary but bounded delays. If running synchronously, the proposed algorithm is as fast as the competitive PG-EXTRA algorithms except, for asynchrony, the proposed algorithm involves updating and communicating additional edge variable.

Our asynchronous setting is considerably less restrictive than the settings under which existing non-synchronous or non-deterministic decentralized algorithms are proposed. In our setting, the computation and communication of agents are uncoordinated. A global clock is not needed.

Technical contributions are also made. We borrow ideas from monotone operator theory and primal-dual operator splitting to derive the proposed algorithms in compact steps. To establish convergence under delays, we cannot follow the existing analysis of PG-EXTRA [2]; instead, motivated by [30], [32], a new non-Euclidean metric is introduced to absorb the delays. In particular, the foundation of the analysis is not the monotonicity of the objective value sequence. Instead, the proposed approach establishes monotonic conditional expectations of certain distances to the solution. We believe this new analysis can extend to a general set of primal-dual algorithms beyond decentralized optimization.

D. Notation

Each agent \( i \) holds a local variable \( x_i^k \in \mathbb{R}^p \), whose value at iteration \( k \) is denoted by \( x_i^{k \leftarrow k} \). We introduce variable \( X \) to stack all local variables \( X \) as:

\[
X := \begin{bmatrix}
(x^1)^\top \\
\vdots \\
(x^p)^\top
\end{bmatrix} \in \mathbb{R}^{n \times p}. \tag{2}
\]
In this paper, we denote the $j$-th row of a matrix $A$ as $A_{j}$. Therefore, for $X$ defined in (2), it holds that $X_{i} = (x^{i})^\top$, which indicates that the $i$-th row of $X$ is the local variable $x^{i} \in \mathbb{R}^{p}$ of agent $i$. Now we define functions
\[
s(X) := \sum_{i=1}^{n} s_{i}(x^{i}), \quad r(X) := \sum_{i=1}^{n} r_{i}(x^{i}),
\]
as well as
\[
f(X) := \sum_{i=1}^{n} f_{i}(x^{i}) = s(X) + r(X).
\]

The gradient of $s(X)$ is defined as
\[
\nabla s(X) := \begin{bmatrix}
\nabla s_{1}(x^{1})^\top \\
\vdots \\
\nabla s_{n}(x^{n})^\top 
\end{bmatrix} \in \mathbb{R}^{n \times p}.
\]

The inner product on $\mathbb{R}^{n \times p}$ is defined as $\langle X, \tilde{X} \rangle := \text{tr}(X^\top \tilde{X}) = \sum_{i=1}^{n} (x^{i})^\top \tilde{x}^{i}$, where $X, \tilde{X}$ are arbitrary matrices.

II. ALGORITHM DEVELOPMENT

Consider a strongly connected network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with agents $\mathcal{V} = \{1, 2, \cdots, n\}$ and undirected edges $\mathcal{E} = \{1, 2, \cdots, m\}$. By convention, all edges $(i, j) \in \mathcal{E}$ obey $i < j$. To each edge $(i, j) \in \mathcal{E}$, we assign a weight $w_{ij} > 0$, which is used by agent $i$ to scale the data $x^{i}$ it receives from agent $j$. Likewise, let $w_{ji} = w_{ij}$ for agent $j$. If $(i, j) \notin \mathcal{E}$, then $w_{ij} = w_{ji} = 0$. For each $i$, we let $\mathcal{N}_{i} := \{j|(i, j) \in \mathcal{E} \text{ or } (j, i) \in \mathcal{E}\} \cup \{i\}$ denote the set of neighbors of agent $i$ (including agent $i$ itself). We also let $\mathcal{E}_{i} := \{(i, j)|(i, j) \in \mathcal{E} \text{ or } (j, i) \in \mathcal{E}\}$ denote the set of all edges connected to $i$. Moreover, we also assume that there exists at least one agent index $i$ such that $w_{ii} > 0$.

Let $W = [w_{ij}] \in \mathbb{R}^{n \times n}$ denote the weight matrix, which is symmetric and doubly stochastic. Such $W$ can be generated through the maximum-degree or Metropolis-Hastings rules [33]. It is easy to verify that $\text{null}(I - W) = \text{span}\{1\}$.

The proposed algorithm involves a matrix $V$, which we now define. Introduce the diagonal matrix $D \in \mathbb{R}^{m \times m}$ with diagonal entries $D_{e,e} = \sqrt{w_{ej}/2}$ for each edge $e = (i, j)$. Let $C := [c_{e}] \in \mathbb{R}^{m \times n}$ be the incidence matrix of $\mathcal{G}$:
\[
c_{e} = \begin{cases} 
+1, & \text{if } e \text{ is the lower indexed agent connected to } \epsilon, \\
-1, & \text{if } e \text{ is the higher indexed agent connected to } \epsilon, \\
0, & \text{otherwise}.
\end{cases}
\]

With incidence matrix $C$, we further define
\[
V := DC \in \mathbb{R}^{m \times n},
\]
as the scaled incidence matrix. It is easy to verify:

Proposition 1 (Matrix factorization identity). When $W$ and $V$ is generated according the above description, we have $V^\top V = (I - W)/2$.

A. Problem formulation

Let us reformulate Problem 1. First, it is equivalent to
\[
\begin{aligned}
\text{minimize} & \quad \sum_{i=1}^{n} s_{i}(x^{i}) + \sum_{i=1}^{n} r_{i}(x^{i}), \\
\text{subject to} & \quad x^{1} = x^{2} = \cdots = x^{n}.
\end{aligned}
\]

Since $\text{null}(I - W) = \text{span}\{1\}$ and recall the definition of $X$ in (2), Problem 5 is equivalent to
\[
\begin{aligned}
\text{minimize} & \quad s(X) + r(X), \\
\text{subject to} & \quad (I - W)X = 0.
\end{aligned}
\]

By Proposition 1, we have $(I - W)X = 0 \Rightarrow V^\top VX = 0 \Rightarrow X^\top V^\top VX = 0 \Rightarrow VX = 0$. On the other hand, $VX = 0 \Rightarrow V^\top VX = 0 \Rightarrow (I - W)X = 0$. Therefore, we conclude that $(I - W)X = 0 \iff VX = 0$. Therefore, Problem (6) is further equivalent to
\[
\begin{aligned}
\text{minimize} & \quad s(X) + r(X), \\
\text{subject to} & \quad VX = 0.
\end{aligned}
\]

Now we denote
\[
Y := \begin{bmatrix}
\cdots & (y^{1})^\top \\
\vdots & \ddots & \\
\cdots & (y^{m})^\top
\end{bmatrix} \in \mathbb{R}^{m \times p}
\]
as the dual variable. Problem (7) can be reformulated into the saddle-point problem
\[
\max_{Y \in \mathbb{R}^{m \times p}} \min_{X \in \mathbb{R}^{n \times p}} s(X) + r(X) + \frac{1}{\alpha} \langle Y, VX \rangle,
\]

where $\alpha > 0$ is a constant parameter. Notice that a similar formulation using the incidence and Laplacian matrices was employed in [14] to derive primal-dual distributed optimization strategies over networks.

B. Synchronous algorithm

Problem 8 can be solved iteratively by the primal-dual algorithm that is adapted from [34],[35]:
\[
\begin{aligned}
X_{k+1} & = \text{prox}_{\alpha} \left[ X^{k} - \alpha \nabla s(X^{k}) - V^\top (2Y^{k+1} - Y^{k}) \right], \\
Y_{k+1} & = Y^{k} + VX^{k},
\end{aligned}
\]

where the proximal operator is defined as
\[
\text{prox}_{\alpha} (U) := \arg \min_{X \in \mathbb{R}^{n \times p}} \left\{ r(X) + \frac{1}{2\alpha} \| X - U \|^2 \right\}.
\]

Next, in the $X$-update, we eliminate $Y^{k+1}$ and use $I - 2V^\top V = W$ to arrive at:
\[
\begin{aligned}
X_{k+1} & = \text{prox}_{\alpha} \left[ W X^{k} - \alpha \nabla s(X^{k}) - V^\top Y^{k} \right], \\
Y_{k+1} & = Y^{k} + VX^{k},
\end{aligned}
\]

which computes $(Y^{k+1}, X^{k+1})$ from $(Y^{k}, X^{k})$. Algorithm [10] can run in a decentralized manner. To do so, we associate each row of the dual variable $Y$ with an edge $e = (i, j) \in \mathcal{E}$.

Algorithm [10] is essentially equivalent to PG-EXTRA developed in [2].
and for simplicity we let agent $i$ store and update the variable $y^e$ (the choice is arbitrary). We also define

$$
L_i := \{e = (i, j) \in E, \forall j \neq i \}, \tag{11}
$$

as the index set of dual variables that agent $i$ needs to update. Recall $V$ is the scaled incidence matrix defined in [4], where $W$ is the weight matrix associated with the network. The calculations of $VX, V^\top Y$ and $WX$ require communication, and other operations are local. For agent $i$, it updates its local variables $x^{i,k+1}$ and $\{y^{e,k+1}\}_{e \in L_i}$ according to

$$
\begin{aligned}
x^{i,k+1} &= \text{prox}_{\alpha e_i} \left( \sum_{j \in N_i} w_{ij} x^{j,k} - \alpha \nabla s_i(x^{i,k}) - \sum_{e \in E_i} v_{ei} y^{e,k} \right), \\
y^{e,k+1} &= y^{e,k} + (v_{ei} x^{i,k} - v_{ej} x^{j,k}), \quad \forall e \in L_i,
\end{aligned} \tag{12}
$$

where $v_{ei}$ and $v_{ej}$ are the $(e, i)$-th and $(e, j)$-th entries of the matrix $V$, respectively.

Algorithm 1 implements recursion (12) in the synchronous fashion, which requires two synchronization barriers in each iteration $k$. The first one holds computing until an agent receives all necessary input; after the agent finishes computing its variables, the second barrier prevents it from sending out information until all of its neighbors finish their computation (otherwise, the information intended for iteration $k + 1$ may arrive at a neighbor too early).

**Algorithm 1: Synchronous algorithm based on (12)**

**Input:** Starting point $\{x^{i,0}\}, \{y^{e,0}\}$. Set counter $k = 0$; while all agents $i \in V$ in parallel do

1. Wait until $\{x^{j,k}\}_{j \in N_i}$ and $\{y^{e,k}\}_{e \in E_i}$ are received;
2. Update $x^{i,k+1}$ according to (12);
3. Update $\{y^{e,k+1}\}_{e \in L_i}$ according to (12);
4. Wait until all neighbors finish computing;
5. Set $k \leftarrow k + 1$;
6. Send out $x^{i,k+1}$ and $\{y^{e,k+1}\}_{e \in L_i}$ to neighbors;

C. Asynchronous algorithm

In the asynchronous setting, each agent computes and communicates independently without any coordinator. Whenever an arbitrary agent finishes a round of its variables’ updates, we let the iteration index $k$ increase by 1 (see Fig. 1). As discussed in Sec. II both communication latency and uncoordinated computation result in delays. Let $\tau^k \in \mathbb{R}^n_+$ and $\delta^k \in \mathbb{R}^m_+$ be vectors of delays at iteration $k$. We define $X^{k-\tau}$ and $Y^{k-\delta}$ as delayed primal and dual variables occurring at iteration $k$:

$$
X^{k-\tau} := \begin{bmatrix}
(x^{1,k-\tau_1})^\top \\
\vdots \\
(x^{n,k-\tau_n})^\top
\end{bmatrix} \in \mathbb{R}^{n \times p}, \\
Y^{k-\delta} := \begin{bmatrix}
(y^{1,k-\delta_1})^\top \\
\vdots \\
(y^{m,k-\delta_m})^\top
\end{bmatrix} \in \mathbb{R}^{m \times p},
$$

where $\tau^k_j$ is the $j$-th element of $\tau_k$ and $\delta^k_j$ is the $j$-th element of $\delta^k$. In the asynchronous setting, recursion (10) is calculated with delayed variables, i.e.,

$$
\begin{aligned}
\tilde{X}^{k+1} &= \text{prox}_{\alpha \nabla S}(WX^{k-\tau} - \alpha \nabla S(X^{k-\tau})), \\
\tilde{Y}^{k+1} &= Y^{k-\delta} + V X^{k-\tau}.
\end{aligned} \tag{13}
$$

Suppose agent $i$ finishes update $k + 1$. To guarantee convergence, instead of letting $x^{i,k+1} = \tilde{x}^{i,k+1}$ and $y^{e,k+1} = \tilde{y}^{e,k+1}$ directly, we propose a relaxation step

$$
\begin{aligned}
x^{i,k+1} &= x^{i,k} + \eta_i (\tilde{x}^{i,k+1} - x^{i,k}), \\
y^{e,k+1} &= y^{e,k} + \eta_i (\tilde{y}^{e,k+1} - y^{e,k}), \quad \forall e \in E_i,
\end{aligned} \tag{14}
$$

where $\tilde{x}^{i,k+1} - x^{i,k}$ and $\tilde{y}^{e,k+1} - y^{e,k}$ behave as updating directions, and $\eta_i \in (0, 1)$ behaves as a step size. To distinguish from the step size $\alpha$, we call $\eta_i$ the relaxation parameter of agent $i$. Its value depends on how out of date agent $i$ receives information from its neighbors. Longer delays require a smaller $\eta_i$, which leads to slower convergence. Since the remaining agents have not finished their updates yet, it holds that

$$
\begin{aligned}
x^{i,k+1} &= x^{i,k} \quad \forall j \neq i, \\
y^{e,k+1} &= y^{e,k} \quad \forall e \not\in L_i. \tag{15}
\end{aligned}
$$

To write (14) and (15) compactly, we let $S^*_p : \mathbb{R}^{n \times p} \rightarrow \mathbb{R}^{n \times p}$ be the primal selection operator associated with agent $i$. For any matrix $A \in \mathbb{R}^{n \times p}$, $[S^*_p(A)]_{ij} = A_j$ if $j = i$; otherwise $[S^*_p(A)]_{ij} = 0$, where $[S^*_p(A)]_{ij}$ and $A_j$ denote the $j$-th row of matrix $S^*_p(A)$ and $A$, respectively. Similarly, we also let $S^*_d : \mathbb{R}^{m \times p} \rightarrow \mathbb{R}^{m \times p}$ be the dual selection operator associated with agent $i$. For any matrix $B \in \mathbb{R}^{m \times p}$, $[S^*_d(B)]_e = B_e$ if $e \in E_i$; otherwise $[S^*_d(B)]_e = 0$. With the above defined selection operators, (14) and (15) are equivalent to

$$
\begin{aligned}
x^{k+1} &= X^k + \eta \tilde{S}^*_p(\tilde{X}^{k+1} - X^{k-\tau}), \\
Y^{k+1} &= Y^k + \eta \tilde{S}^*_d(\tilde{Y}^{k+1} - Y^{k-\delta}). \tag{16}
\end{aligned}
$$

Recursions (13) and (16) constitute the asynchronous algorithm.

Similar to the synchronous algorithm, the asynchronous recursion (13) and (16) can also be implemented in a decentralized manner. When agent $i$ is activated at iteration $k$:

Compute:

$$
\begin{aligned}
\tilde{x}^{i,k+1} &= \text{prox}_{\alpha \nabla S}(\sum_{j \in N_i} w_{ij} x^{j,k-\tau_j} - \alpha \nabla S(x^{i,k-\tau_i})) \sum_{e \in E_i} v_{ei} y^{e,k-\delta_e}, \\
\tilde{y}^{e,k+1} &= y^{e,k} + (v_{ei} x^{i,k-\tau_i} + v_{ej} x^{j,k-\tau_j}), \quad \forall e \in E_i;
\end{aligned}
$$

Relaxed updates:

$$
\begin{aligned}
x^{i,k+1} &= x^{i,k} + \eta_i (\tilde{x}^{i,k+1} - x^{i,k}), \\
y^{e,k+1} &= y^{e,k} + \eta_i (\tilde{y}^{e,k+1} - y^{e,k}), \quad \forall e \in E_i. \tag{17}
\end{aligned}
$$

The entries of $X, Y$ not held by agent $i$ remain unchanged from $k$ to $k + 1$. Algorithm 2 summarizes the asynchronous updates.
Algorithm 2: Asynchronous algorithm based on (17)

Input: Starting point \( \{x^{(0)}\}, \{y^{(0)}\} \). Set counter \( k = 0 \);
while each agent \( i \) asynchronously do

Compute per (17) using the information it has available;
Send out \( x^{i,k+1} \) and \( \{y^{e,k+1}\}_{e \in \mathcal{E}} \), to neighbors;

D. Asynchronous proximal decentralized gradient descent

Sec. II-C has presented a decentralized primal-dual algorithm for problem (1). In [12], [13], [36], proximal gradient descent algorithms are derived to solve stochastic composite optimization problems in a distributed manner by networked agents. Using techniques similar to those developed in [12], [13], [36] we can likewise derive a proximal variant for the consensus gradient descent algorithm in [6]; see (21) below. Following Sec. II-C its asynchronous version will also be developed; see (22) below.

To derive the synchronous algorithm, we penalize the constraints of Problem (7) (which is equivalent to Problem (1)) and obtain an auxiliary problem:

\[
\minimize_{X \in \mathbb{R}^{n \times p}} s(X) + r(X) + \frac{1}{\alpha} \|VX\|^2, \tag{18}
\]

where \( \alpha > 0 \) is a penalty constant. By \( V^\top V = \frac{1}{2}(I - W) \) in Proposition 1 Problem (18) is exactly

\[
\minimize_{X \in \mathbb{R}^{n \times p}} s(X) + \frac{1}{2\alpha} X^\top (I - W)X + r(X), \tag{19}
\]

which has the smooth-plus-proximable form. Therefore, applying the proximal gradient method with step-size \( \alpha \) yields the iteration:

\[
x^{k+1} = \text{prox}_\alpha_{\text{or}} \left( x^k - \alpha \nabla s(x^k) - \frac{1}{\alpha} (I - W)x^k \right)
\]

\[
= \text{prox}_\alpha_{\text{or}} (Wx^k - \alpha \nabla s(x^k)), \tag{20}
\]

Recursion (20) is ready for decentralized implementation — each agent \( i \in \mathcal{V} \) performs

\[
x^{i,k+1} = \text{prox}_{\alpha_{1i}} \left( \sum_{j \in \mathcal{N}_i} w_{ij} x^{j,k} - \alpha \nabla s_i(x^{i,k}) \right), \tag{21}
\]

Since algorithm (21) solves the penalty problem (19) instead of the original problem (1), one must reduce \( \alpha \) during the iterations or use a small value. Recursion (21) is similar to the proximal diffusion gradient descent algorithm derived in [12], [13] in which each agent first applies local proximal gradient descent and then performs weighted averaging with neighbors. According to [11], [33], the diffusion strategy removes the asymmetry problem that can cause consensus implementations to become unstable for stochastic optimization problems.

The asynchronous version of (21) allows delayed variables and adds relaxation to (21); each agent \( i \) performs

\[
\tilde{x}^{i,k+1} = \text{prox}_{\alpha_{1i}} \left( \sum_{j \in \mathcal{N}_i} w_{ij} \tilde{x}^{j,k} - r_{ij} - \alpha \nabla s_i(\tilde{x}^{i,k} - r_{ij}) \right),
\]

\[
x^{i,k+1} = x^{i,k} + \eta_i (\tilde{x}^{i,k+1} - x^{i,k} - r_{ij}), \tag{22}
\]

Its convergence follows from treating the proximal gradient iteration as a fixed-point iteration and applying results from [30]. Unlike Algorithm 2 the asynchronous algorithm (22) uses only primal variables, so it is easier to implement. Like all other distributed gradient descent algorithms [6], [11]–[13], [36], however, algorithm (21) and its asynchronous version (22) must use diminishing step sizes to guarantee exact convergence, which causes slow convergence.

III. CONVERGENCE ANALYSIS

A. Preliminaries

In this subsection we first present several definitions, lemmas and theorems that underlie the convergence analysis.

First we introduce a new symmetric matrix

\[
G := \begin{bmatrix} I_n & V \\ V & I_m \end{bmatrix} \in \mathbb{R}^{m+n}.
\]

As we will discuss later, \( G \) is an important auxiliary matrix that helps establish the convergence properties of the synchronous Algorithm 1 and asynchronous Algorithm 2. Meanwhile, it also determines the range of step size \( \alpha \) that enables the algorithms to converge. Recall that the weight matrix \( W \) associated with the network is symmetric and doubly stochastic; besides, there exists at least one agent \( i \) such that \( w_{ii} > 0 \). Under these conditions, it is shown in [33] that \( W \) has an eigenvalue 1 with multiplicity 1, and all other eigenvalues are strictly inside the unit circle. With such \( W \), we can show that \( G \) is positive definite.

Lemma 1. It holds that \( G > 0 \).

Proof: According to the Schur Complement condition on positive definiteness, we know \( G > 0 \iff I > 0, I - V^\top V > 0 \). Recall Proposition 1 and \( \lambda_{\text{min}}(W) > -1 \), it holds that \( I - V^\top V = \frac{1}{2}(I + W) > 0 \), which proves \( G > 0 \).

To analyze the convergence of the proposed Algorithms 1 and 2 we will need a classic result from the nonexpansive operator theory. In the following we provide related definitions and preliminary results.

Definition 1. Let \( \mathcal{H} \) be a finite dimensional vector space (e.g. \( \mathbb{R}^p \) or \( \mathbb{R}^{n \times p} \)) equipped with a certain inner product \( \langle \cdot, \cdot \rangle \) and its induced norm \( \|x\| = \sqrt{\langle x, x \rangle}, \forall x \in \mathcal{H} \). An operator \( \mathcal{P} : \mathcal{H} \to \mathcal{H} \) is called nonexpansive if

\[
\|Px - Py\| \leq \|x - y\|, \forall x, y \in \mathcal{H}.
\]

Definition 2. For any operator \( \mathcal{P} : \mathcal{H} \to \mathcal{H} \) (not necessarily nonexpansive), we define \( \text{Fix} \mathcal{P} := \{x \in \mathcal{H} | x = \mathcal{P}x\} \) to be the set of fixed points of operator \( \mathcal{P} \).

Definition 3. When the operator \( \mathcal{P} \) is nonexpansive and \( \beta \in (0, 1) \), the combination operator

\[
\mathcal{Q} := (1 - \beta)I + \beta \mathcal{P}
\]

is defined as the averaged operator associated with \( \mathcal{P} \).

With Definition 2 and 3 it is easy to verify \( \text{Fix} \mathcal{Q} = \text{Fix} \mathcal{P} \).

The next classical theorem states the convergence property of an averaged operator.
Theorem 1 (KM iteration [37], [38]). Let $P : \mathcal{H} \to \mathcal{H}$ be a nonexpansive operator and $\beta \in (0, 1)$. Let $Q$ be the averaged operator associated with $P$. Suppose that the set $\text{Fix}Q$ is nonempty. From any starting point $z^0$, the iteration

$$z^{k+1} = Qz^k = (1 - \beta)z^k + \beta Pz^k$$

produces a sequence $(z^k)_{k\geq 0}$ converging to a point $z^* \in \text{Fix}Q$ (and hence also to a point $z^* \in \text{Fix}P$).

For the remainder of the paper, we consider a specific Hilbert space $\mathcal{H}$ for the purpose of analysis.

Definition 4. Define $\mathcal{H}$ to be the Hilbert space $\mathbb{R}^{(n+m)\times p}$ endowed with the inner product $(Z, \bar{Z})_A := \text{tr}(Z^T A \bar{Z})$ and the norm $\|Z\|_A := \sqrt{(Z, \bar{Z})_A}$, where $A \in \mathbb{R}^{(n+m)\times (n+m)}$ is a positive definite matrix, and $Z \in \mathbb{R}^{(n+m)\times p}$.

B. Convergence of Algorithm 1

In this subsection we prove the convergence of Algorithm 1. We first re-derive the iteration (9) using operator splitting techniques and identify it as an averaged operator. Next, we apply Theorem 1 and establish its convergence property.

Assumption 1.

1) The functions $s_i$ and $r_i$ are closed, proper and convex.
2) The functions $s_i$ are differentiable and satisfy:

$$\|\nabla s_i(x) - \nabla s_i(\bar{x})\| \leq L_i \|x - \bar{x}\|, \ \forall x, \bar{x} \in \mathbb{R}^p,$$

where $L_i > 0$ is the Lipschitz constant.
3) The parameter $\alpha$ in synchronous algorithm [12] and asynchronous algorithm [17] satisfies $0 < \alpha < 2\rho_{\text{min}}/L_i$, where $L_i := \max_i L_i$.

Since $r$ and $s$ are convex, the solution to the saddle point problem (8) is the same as the solution to the following Karush-Kuhn-Tucker (KKT) system:

$$0 \in \begin{bmatrix} \nabla s & 0 & 0 \\ 0 & \frac{\partial r}{\partial x} & 0 \\ 0 & 0 & -\frac{1}{\alpha}V^T \\ \frac{1}{\alpha}V & 0 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \quad (23)$$

which can be written as

$$0 \in AZ + BZ, \quad \text{with} \quad Z := \begin{bmatrix} X \\ Y \end{bmatrix} \in \mathbb{R}^{(n+m)\times p}.$$

For any positive definite matrix $M \in \mathbb{R}^{(n+m)\times (n+m)}$, we have

$$0 \in AZ + BZ \iff MZ - AZ \in MZ + BZ \quad (24)$$
$$\iff Z - M^{-1}AZ \in Z + M^{-1}BZ$$
$$\iff Z = (I + M^{-1}B)^{-1}(I - M^{-1}A)Z,$$

Note that $(I + M^{-1}B)^{-1}$ is well-defined and single-valued; it is called resolvent [39]. From now on, we let

$$M := \frac{1}{\alpha}G > 0 \quad (25)$$

and define the operator

$$T := (I + M^{-1}B)^{-1}(I - M^{-1}A). \quad (26)$$

From (23), (24) and the definition of operator $T$ in (26), we conclude that the solutions to the KKT system (23) coincide with the fixed points of $T$.

Now we consider the fixed-point iteration

$$Z^{k+1} = T Z^k = (I + M^{-1}B)^{-1}(I - M^{-1}A)Z^k, \quad (27)$$

which reduces to

$$MZ^k - AZ^k \in MZ^k + BZ^k. \quad (28)$$

Substituting $M$ defined in (25) into (28) and multiplying $\alpha$ to both sides, we will achieve

$$X^{k+1} + BV - \alpha \nabla s(X^k) = X^{k+1} + 2V^T Y^{k+1} + \alpha \partial r(X^{k+1}),$$
$$Y^{k+1} + V X^{k+1} = Y^{k+1} + V X^{k+1} - V X^{k+1},$$

which, by cancellation, is further equivalent to Recursion (9).

Until now, we have rewritten the primal-dual algorithm (9) as a fixed-point iteration (27) with operator $T$ defined in (26). Besides, we also established that $\text{Fix}T$ coincide with the solutions to the KKT system (23). What still needs to be proved is that $(Z^k)$ generated through the fixed-point iteration (27) will converge to $\text{Fix}T$. Below is a classic important result of the operator $T$. For a proof, see [39].

Theorem 2. Under Assumption [7] and the above-defined norm with $M = G/\alpha$, there exists a nonexpansive operator $O$ and some $\beta \in (0, 1)$ such that the operator $T$ defined in (26) satisfies $T = (1 - \beta)I + \beta O$. Hence, $T$ is an averaged operator.

The convergence of the synchronous update (9) follows directly from Theorems 1 and 2.

Corollary 1. Under Assumption 1, Algorithm 1 produces $Z^k$ that converges to a solution $Z^*$ to Problem (8), which is also a saddle point to Problem (8). Therefore, $X^*$ is a solution to Problem (1).

Corollary 1 states that if we run algorithm 1 in the synchronous fashion, the agents’ local variables will converge to a consensus solution to the problem (1).

C. Convergence of Algorithm 2

Algorithm 2 is an asynchronous version that involves random variables. In this subsection we will establish its almost sure (a.s.) convergence (i.e., with probability 1). The analysis is inspired by [30] and [32].

1) Algorithm reformulation: It was shown in Sec. III-B that Algorithm 1 can be rewritten as a fixed-point iteration (27). Now we let $T_i$ be the operator corresponding to the update performed by agent $i$, i.e., $T_i$ only updates $x^i$ and $\{y^j\}_{j \in L_i}$ (the definition of $L_i$ can be referred to [11]) and leaves the other variables unchanged. We define

$$L_i = \{i\} \cup \{n + e \mid e \text{ is the edge index of } (i, j) \in L_i\}$$

to be the set of all row indices of $Z$ that agent $i$ needs to update. The operator $T_i : \mathbb{R}^{(m+n)\times p} \to \mathbb{R}^{(m+n)\times p}$ is defined as follows. For any $Z \in \mathbb{R}^{(m+n)\times p}$, $T_i Z \in \mathbb{R}^{(m+n)\times p}$ is a matrix with

$$\{T_i Z\}_{j} := \begin{cases} \{TZ\}_{j}, & \text{if } j \in L_i; \\ Z_{j}, & \text{if } j \notin L_i. \end{cases} \quad (29)$$
where $Z_j$ (\(TZ\)) and (\(T_iZ\)) are the \(j\)-th row of Z, T Z and \(T_iZ\), respectively (see the notation section \[1\]). Moreover, we define

\[
S := I - T, \quad S_i := I - T_i. \tag{30}
\]

Based on the definition of the operator \(T_i\), for any \(Z \in \mathbb{R}^{(m+n) \times p}\), \(S_i Z \in \mathbb{R}^{(m+n) \times p}\) is also a matrix with

\[
(S_i Z)_{j} := \begin{cases} (Z - T_i Z)_{j}, & \text{if } j \in I_i; \\ 0, & \text{if } j \notin I_i. \end{cases} \tag{31}
\]

We also define the delayed variable used in iteration \(k\) as

\[
\hat{Z}^k := \left[ X^{k -} \beta \right] \in \mathbb{R}^{(m+n) \times p}. \tag{32}
\]

Suppose agent \(i_k\) is activated at iteration \(k\), with definitions \([29]\) and \([32]\), it can be verified that recursions \([13]\) and \([16]\) are equivalent to:

\[
\begin{cases} \hat{Z}^k = T_{i_k} \hat{Z}^k, \\ Z^{k+1} = Z^k - \eta_{i_k} \left( \hat{Z}^k - \hat{Z}^k \right). \end{cases} \tag{33}
\]

From recursion \([33]\) we further have

\[
Z^{k+1} = Z^k - \eta_{i_k} (\hat{Z}^k - T_{i_k} \hat{Z}^k) + \eta_{i_k} S_{i_k} \hat{Z}^k. \tag{34}
\]

Therefore, the asynchronous update algorithm \([17]\) can be viewed as a stochastic coordinate-update iteration based on delayed variables \([30]\), \([32]\), with each coordinate update taking the form of \([34]\).

2) Relation between \(\hat{Z}^k\) and \(Z^k\): The following assumption introduces a uniform upper bound for the random delays.

**Assumption 2.** At any iteration \(k\), the delays \(\tau_j^k, j = 1, 2, \ldots, n\) and \(\delta^k, e = 1, 2, \ldots, m\) defined in \([17]\) have a uniform upper bound \(\tau > 0\).

As we are finishing this paper, the recent work \([40]\) has relaxed the above assumption by associating step sizes to potentially unbounded delays. However, we still keep this assumption in this paper for simplicity. Under Assumption \([2]\), the relation between \(\hat{Z}^k\) and \(Z^k\) can be characterized in the following lemma \([30]\).

**Lemma 2.** Under Assumption \([2]\) it holds that

\[
\hat{Z}^k = Z^k + \sum_{d \in J(k)} (Z^d - Z^{d+1}), \tag{35}
\]

where \(J(k) \subseteq \{k - 1, \ldots, k - \tau\}\) is an index set.

The proof of relation \([35]\) can be referred to \([30]\).

3) Convergence analysis: Now we introduce an assumption about the activation probability of each agent.

**Assumption 3.** For any \(k > 0\), let \(i_k\) be the index of the agent that is responsible for the \(k\)-th completed update. It is assumed that each \(i_k\) is a random variable. The random variable \(i_k\) is independent of \(i_1, i_2, \ldots, i_{k-1}\) as

\[
P(i_k = i) =: q_i > 0,
\]

where \(q_i\)’s are constants.

This assumption is satisfied under either of the following scenarios: (i) every agent \(i\) is activated following an independent Poisson process with parameter \(\lambda_i\), and any computation occurring at agent \(i\) is instant, leading to \(q_i = \lambda_i / \left(\sum_{i=1}^{n} \lambda_i\right)\); (ii) every agent \(i\) runs continuously, and the duration of each round of computation follows the exponential distribution \(\exp(\beta_i)\), leading to \(q_i = \beta_i^{-1} / \left(\sum_{i=1}^{n} \beta_i^{-1}\right)\). Scenarios (i) and (ii) often appear as assumptions in the existing literature.

**Definition 5.** Let \((\Omega, \mathcal{F}, P)\) be the probability space we work with, where \(\Omega, \mathcal{F}\) and \(P\) are the sample space, \(\sigma\)-algebra and the probability measure, respectively. Define

\[
Z^k := \sigma(Z^0, \tilde{Z}^0, Z^1, \tilde{Z}^1, \ldots, Z^k, \tilde{Z}^k)
\]

to be the \(\sigma\)-algebra generated by \(Z^0, \tilde{Z}^0, \ldots, Z^k, \tilde{Z}^k\).

**Assumption 4.** Throughout our analysis, we assume

\[
P(i_k = i|Z^k) = P(i_k = i) = q_i, \quad \forall i, k.
\]

**Remark 1.** Assumption \([4]\) indicates that the index responsible for the \(k\)-th completed update, \(i_k\), is independent of the delays in different rows of \(Z^k\). Although not always practical, it is a key assumption for our proof to go through. One of the cases for this assumption to hold is when every row of \(Z^k\) always has the maximum delay \(\tau\). In reality, what happens is between this worst case and the no-delay case. Besides, Assumption \([4]\) is also a common assumption in the recent literature of stochastic asynchronous algorithms; see \([30]\) and the references therein.

Now we are ready to prove the convergence of Algorithm \([2]\). For simplicity, we define

\[
\hat{Z}^{k+1} := Z^k - \eta S \hat{Z}^k, \quad q_{\text{min}} := \min\{q_i\} > 0, \tag{36}
\]

\[
\lambda_{\text{max}} := \lambda_{\text{max}}(M), \quad \lambda_{\text{min}} := \lambda_{\text{min}}(M),
\]

where \(\eta > 0\) is some constant, and \(M > 0\) is defined in \([25]\). Note that \(Z^{k+1}\) is the full update as if all agents synchronously compute at the \(k\)-th iteration. It is used only for analysis.

**Lemma 3.** Define \(Q := I - P\). When \(P : \mathbb{R}^{(m+n) \times p} \to \mathbb{R}^{(m+n) \times p}\) is nonexpansive under the norm \(\|\cdot\|_A\), we have

\[
\langle Z - \bar{Z}, QZ - Q\bar{Z}\rangle_A \geq \frac{1}{2} \|QZ - Q\bar{Z}\|_A^2, \tag{37}
\]

where \(Z, \bar{Z} \in \mathbb{R}^{(m+n) \times p}\) are two arbitrary matrices.

**Proof:** By nonexpansiveness of \(P\), we have

\[
\|PZ - P\bar{Z}\|_A^2 \leq \|Z - \bar{Z}\|_A^2, \quad \forall Z, \bar{Z} \in \mathbb{R}^{(m+n) \times p}. \tag{38}
\]

Notice that

\[
\|PZ - P\bar{Z}\|_A^2 = \|Z - \bar{Z} - (QZ - Q\bar{Z})\|_A^2 = \|Z - \bar{Z}\|_A^2 - 2\langle Z - \bar{Z}, QZ - Q\bar{Z}\rangle_A + \|QZ - Q\bar{Z}\|_A^2. \tag{39}
\]

Substituting \([39]\) into \([38]\), we achieve \([37]\).

Let \(\kappa\) be its condition number of matrix \(G\). Since \(M = G/\alpha\), \(\kappa\) is also the condition number of \(M\). The following lemma establishes the relation between \(S_i\) and \(S\).

\[
\|S_i - S\|_A \leq \kappa, \quad \forall S_i, S \in \mathbb{R}^{(m+n) \times p}.
\]
Lemma 4. Recall the definition of $S, S_i$ and $M$ in (30) and (25). It holds that
\begin{equation}
\sum_{i=1}^{n} S_i \tilde{Z}_i^k = S \tilde{Z}_i^k, \quad \sum_{i=1}^{n} ||S_i \tilde{Z}_i^k||^2_M \leq \kappa ||S \tilde{Z}_i^k||^2_M. \tag{40}
\end{equation}

Proof: The first part comes immediately from the definition of $S$ and $S_i$ in (30) and (34). For the second part,
\begin{equation}
\sum_{i=1}^{n} ||S_i \tilde{Z}_i^k||^2_M \leq \sum_{i=1}^{n} \lambda_{\text{max}} ||S_i \tilde{Z}_i^k||^2_M = \lambda_{\text{max}} ||S \tilde{Z}_i^k||^2_M
\end{equation}

\[ \leq \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} ||S \tilde{Z}_i^k||^2_M = \kappa ||S \tilde{Z}_i^k||^2_M. \]

The next lemma shows that the conditional expectation of the distance between $Z^{k+1}$ and any $Z^* \in \text{Fix} T$ for given $Z^k$ has an upper bound that depends on $Z^k$ and $Z^*$ only. From now on, the relaxation parameters will be set as
\begin{equation}
\eta_i = \frac{\eta}{n \eta_{\text{ini}}}, \tag{41}
\end{equation}

where $\eta > 0$ is the constant appearing at (36).

Lemma 5. Let $\{Z^k\}_{k \geq 0}$ be the sequence generated by Algorithm 2. Then for any $Z^* \in \text{Fix} T$, we have
\begin{equation}
\mathbb{E}(||Z^{k+1} - Z^*||^2_M | Z^k)
\leq ||Z^k - Z^*||^2_M + \frac{\xi}{n} \sum_{k - \tau \leq d \leq k - \tau + 1} ||Z^{d} - Z^{d+1}||^2_M
\end{equation}

\[ + \frac{1}{n} \left( \frac{\xi + \frac{\kappa}{n \eta_{\text{ini}}} - \frac{1}{\eta}}{\xi} \right) ||Z^k - Z^{k+1}||^2_M, \tag{42}
\]

where $\mathbb{E}(\cdot | Z^k)$ denotes conditional expectation on $Z^k$ and $\xi$ is an arbitrary positive number.

Proof: We have
\begin{equation}
\mathbb{E}(||Z^{k+1} - Z^*||^2_M | Z^k)
= \mathbb{E}(\|Z^k - \frac{\eta}{n \eta_{\text{ini}}} S_i \tilde{Z}_i^k - Z^*\|^2_M | Z^k)
\end{equation}

\[ = \mathbb{E} \left( \left\| \frac{2\eta}{n \eta_{\text{ini}}} (S_i \tilde{Z}_i^k, Z^* - Z^k)_M + \frac{\eta^2}{n^2 q_{\text{ini}}} ||S_i \tilde{Z}_i^k||^2_M \right\| Z^k \right) \]

\[ + ||Z^k - Z^*||^2_M, \tag{43}
\]

\[ \leq \frac{2\eta}{n} \sum_{i=1}^{n} (S_i \tilde{Z}_i^k, Z^* - Z^k)_M + \frac{\eta^2}{n^2} \sum_{i=1}^{n} q_{\text{ini}} ||S_i \tilde{Z}_i^k||^2_M
\]

\[ + ||Z^k - Z^*||^2_M, \tag{44}
\]

where the equality (a) holds because of Assumptions 3 and 4.

On the other hand, note that
\begin{equation}
\sum_{i=1}^{n} \frac{1}{q_{\text{ini}}} ||S_i \tilde{Z}_i^k||^2_M \leq \frac{1}{q_{\text{min}}} \sum_{i=1}^{n} ||S_i \tilde{Z}_i^k||^2_M \leq \frac{\kappa}{q_{\text{min}}} \sum_{i=1}^{n} ||S \tilde{Z}_i^k||^2_M
\end{equation}

\[ \leq \frac{\kappa}{\eta^2 q_{\text{min}}} ||Z^k - Z^{k+1}||^2_M, \tag{45}
\]

and
\begin{equation}
\left< S \tilde{Z}_i^k, Z^* - Z^k \right>_M
\leq \left< Z^k - Z^{k+1}, Z^k - Z^{k+1} \right>_M
\end{equation}

\[ + \frac{1}{2\eta} \sum_{d \in \mathcal{J}(k)} \left( \frac{1}{\xi} ||Z^k - Z^{k+1}||^2_M + \frac{1}{\xi} ||Z^k - Z^{d+1}||^2_M \right) \]

\[ + \frac{\xi}{2\eta} \sum_{d \in \mathcal{J}(k)} ||Z^k - Z^{d+1}||^2_M, \tag{46}
\]

where the inequality (b) follows from Young’s inequality and the fact that $SZ^* = (I - \mathcal{H})Z^* = 0$ since $Z^*$ is a fixed point of $\mathcal{T}$. Substituting (44) and (45) into (43) and using $\mathcal{J}(k) \subseteq \{k - 1, \ldots, k - \tau\}$ we get the desired result.

The term $\frac{\xi}{n} \sum_{k - \tau < d < k} ||Z^d - Z^{d+1}||^2$ in the inequality (42) appears because of the delay. Next we stack $\tau + 1$ iterates together to form a new vector and introduce a new metric in order to absorb these terms.

Let $\mathcal{H}^{\tau + 1} = \prod_{i=0}^{\tau + 1} \mathcal{H}$ be a product space (Recall $\mathcal{H}$ denotes $\mathbb{R}^{(m+n) \times p}$, see Definition 4). For any $(Z_0, \ldots, Z_{\tau}), (\bar{Z}_0, \ldots, \bar{Z}_\tau) \in \mathcal{H}^{\tau + 1}$, we let $\langle \cdot, \cdot \rangle$ be the induced inner product, i.e., $(Z_0, \ldots, Z_{\tau}, (\bar{Z}_0, \ldots, \bar{Z}_\tau)) := \sum_{i=0}^{\tau} (Z_i, \bar{Z}_i)_M = \sum_{i=0}^{\tau} \text{tr}(Z_i^t M \bar{Z}_i)$. Define a matrix $U' \in \mathbb{R}^{(\tau + 1) \times (\tau + 1)}$ as
\begin{equation}
U' := U_1 + U_2,
\end{equation}

where
\begin{equation}
U_1 := \begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{bmatrix},
\end{equation}

\begin{equation}
U_2 := \sqrt{\frac{q_{\text{min}}}{\kappa}} \begin{bmatrix}
\tau & -\tau & \cdots & \cdots \\
-\tau & 2\tau - 1 & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots \\
0 & -2 & 3 & -1 \\
\end{bmatrix},
\end{equation}

and let $U = U' \otimes I$ where $\otimes$ represents the Kronecker product and $I$ is the identity operator. For a given $(A_0, \ldots, A_{\tau}) \in \mathcal{H}^{\tau + 1}$,
\begin{equation}
(B_0, \ldots, B_{\tau}) = U(A_0, \ldots, A_{\tau})
\end{equation}
is given by:
\[
B_0 = A_0 + \tau \sqrt{\frac{q_{\min}}{\kappa}} (A_0 - A_1),
\]
\[
B_i = \sqrt{\frac{q_{\min}}{\kappa}} [(i - \tau - 1)A_{i-1} + (2\tau - 2i + 1)A_i + (i - \tau)A_{i+1}],
\]
\[
B_\tau = \sqrt{\frac{q_{\min}}{\kappa}} (A_\tau - A_{\tau - 1}).
\]
(46)

where the index \(i\) for \(B_i\) is from 1 to \(\tau + 1\). The linear operator \(U\) is a self-adjoint and positive definite since \(U^*\) is symmetric and positive definite. We define \(\langle \cdot, \cdot \rangle_U = \langle \cdot, U \cdot \rangle\) as the \(U\)-weighted inner product and \(\| \cdot \|_U\) as the induced norm. We further let
\[
Z^k = (Z^k, Z^{k-1}, \ldots, Z^{k-\tau}) \in \mathcal{H}^{\tau + 1}, \ k \geq 0,
\]
\[
Z^* = (Z^*, \ldots, Z^*) \in \mathcal{H}^{\tau + 1},
\]
where \(Z^k = Z^0\) for \(k < 0\). We have
\[
\|Z^k - Z^*\|^2_U \geq \frac{q_{\min}}{\kappa} \sum_{d=k-\tau}^{k-1} (d-(k-\tau)+1) \|Z^d - Z^{d+1}\|^2_M.
\]
(47)

and the following fundamental inequality.

**Theorem 3 (Fundamental inequality).** Let \(\{Z^k\}_{k \geq 0}\) be the sequence generated by Algorithm 2. Then for any \(Z^* = (Z^*, \ldots, Z^*)\), it holds that
\[
\mathbb{E}(\|Z^{k+1} - Z^*\|^2_U | Z^k) \geq \frac{q_{\min}}{\kappa \sqrt{q_{\min}/\kappa}} \|Z^{k+1} - Z^k\|^2_M.
\]
(48)

**Proof:** Let \(\xi = n \sqrt{q_{\min}/\kappa}\). We have
\[
\mathbb{E}(\|Z^{k+1} - Z^*\|^2_U | Z^k) \geq \frac{q_{\min}}{\kappa \sqrt{q_{\min}/\kappa}} \|Z^{k+1} - Z^k\|^2_M
\]
(49)

\[
\geq \frac{q_{\min}}{\kappa \sqrt{q_{\min}/\kappa}} \|Z^{k+1} - Z^k\|^2_M
\]

**Remark 2 (Stochastic Fejér monotonicity [41]).** From (48), suppose
\[
0 < \eta < \frac{n q_{\min}}{2 \tau \sqrt{q_{\min}/\kappa} + \kappa},
\]
(49)

then we have
\[
\mathbb{E}(\|Z^{k+1} - Z^*\|^2_U | Z^k) \leq \|Z^k - Z^*\|^2_U,
\]
\(i.e., the sequence \{Z^k\}_{k \geq 0}\) is stochastically Fejér monotone, which means the covariance of \(Z^k\) is non-increasing as the iteration \(k\) evolves.

Based on Theorem 3, we have the following corollary:

**Corollary 2. When \(\eta\) satisfies (49), we have**

1) \(\sum_{k=0}^{\infty} \|Z^k - Z^{k+1}\|^2_M < \infty\) a.s.;

2) the sequence \{\|Z^k - Z^*\|^2_U\}_{k \geq 0}\) converges to a \([0, +\infty)\)-valued random variable a.s.

**Proof:** The condition (49) implies that \(\frac{1}{n \sqrt{q_{\min}/\kappa}} - \frac{2 \tau \sqrt{q_{\min}/\kappa}}{n q_{\min}} > 0\). Applying (42) Theorem 1 to (48) directly gives the two conclusions.

The following lemma establishes the convergence properties of the sequences \{\(Z^k\)\}_{k \geq 1}\) and \(\{Z^k\}_{k \geq 1}\).

**Lemma 6. Define \(S^* = \{(Z^*, \ldots, Z^*)| Z^* \in \text{Fix}\mathcal{T} \}, let \(Z^k\)\}_{k \geq 0} \subset \mathcal{H}\) be the sequence generated by Algorithm 2 with \(\eta \in (0, q_{\max})\) for certain \(q_{\max}\) satisfying (49). Let \(Z^k\) be the set of cluster points of \{\(Z^k\)\}_{k \geq 0}. Then, \(Z^k\) converges in \(S^*\) a.s.

**Proof:** We take several steps to complete the proof of this lemma.

(i) Firstly, from Corollary 2 we have \(Z^k - Z^{k+1} \to 0\) a.s.

Since \(\|Z^k - Z^{k+1}\|_F \leq \frac{\sqrt{\kappa}}{n q_{\min}} \|Z^k - Z^{k+1}\|_M\) (c.f. (44)), we have \(\|Z^k - Z^{k+1}\| \to 0\) a.s. Then from (35), we have \(\hat{Z} - Z^k \to 0\) a.s.

(ii) From Corollary 2 we have that \(\|Z^k - Z^*\|^2_U\) converges a.s., and so does \(\|Z^k - Z^*\|_U\) converges a.s. Hence, we have \(\lim_{k \to \infty} \|Z^k - Z^*\|_U = \gamma\) a.s., where \(\gamma\) is a \([0, +\infty)\)-valued random variable. Hence, \(\|Z^k - Z^*\|_U\) must be bounded a.s., and so is \(\{Z^k\}_{k \geq 0}\).

(iii) We claim that there exists \(\Omega \in F\) such that \(P(\Omega) = 1\) and, for every \(\omega \in \Omega\) and every \(Z^* \in S^*\), \(\|Z^k(\omega) - Z^*\|_U\) converges.

The proof follows directly from [41] Proposition 2.3 (iii). It is worth noting that \(\Omega\) in the statement works for all \(Z^* \in S^*\), namely, \(\Omega\) does not depend on \(Z^*\).

(iv) By (i), there exists \(\Omega \in F\) such that \(P(\Omega) = 1\) and \(Z^k(w) - Z^{k+1}(w) \to 0, \forall w \in \mathcal{W}\). For any \(w \in \Omega\), let \(Z^k(w)\) be a convergent subsequence of \(\{Z^k(w)\}_{k \geq 0}\), i.e., \(Z^k(\omega) \to Z\), where \(Z^k(\omega) = (Z^{k_1}(\omega), \ldots, Z^{k_k-(\omega)})\) and \(Z = (u^0, \ldots, u^*).\) If \(Z^k(\omega) \to Z\) implies \(Z^{k-i}(\omega) \to u^i, \forall j\), then \(Z^{k-i}(\omega) \to Z^{k-i}(\omega) \to 0\). Furthermore, observing \(\eta > 0\), we have
\[
\lim_{k \to \infty} Z^{k-i}(\omega) - T Z^{k-i}(\omega) = 0.
\]
(50)
From the triangle inequality and the nonexpansiveness of $\mathcal{T}$, it follows that
\[
\|Z_{k}^{i}(\omega) - T Z_{k}^{i}(\omega)\|_{M} \\
= \|Z_{k}^{i}(\omega) - \hat{Z}_{k}^{i}(\omega) + \hat{Z}_{k}^{i}(\omega) - T \hat{Z}_{k}^{i}(\omega) + T Z_{k}^{i}(\omega)\|_{M} \\
\leq \|Z_{k}^{i}(\omega) - \hat{Z}_{k}^{i}(\omega)\|_{M} + \|\hat{Z}_{k}^{i}(\omega) - T \hat{Z}_{k}^{i}(\omega)\|_{M} \\
+ \|T \hat{Z}_{k}^{i}(\omega) - T Z_{k}^{i}(\omega)\|_{M} \\
\leq 2\|Z_{k}^{i}(\omega) - \hat{Z}_{k}^{i}(\omega)\|_{M} + \|\hat{Z}_{k}^{i}(\omega) - T \hat{Z}_{k}^{i}(\omega)\|_{M} \\
\leq 2\sum_{i\in\mathcal{E}(k)} \|Z_{k}^{i}(\omega) - Z_{k+1}^{i}(\omega)\|_{M} + \|\hat{Z}_{k}^{i}(\omega) - T \hat{Z}_{k}^{i}(\omega)\|_{M}.
\]
By (III-C3) and (III-C5), we have from the above inequality that $\lim_{k\to\infty} \|Z_{k}^{i}(\omega) - \hat{Z}_{k}^{i}(\omega)\|_{M} = 0$. Now the demiclosedness principle [38] Theorem 4.17 implies $u^{0} \in \text{Fix } \mathcal{T}$, and this implies the statement of the lemma.

From Lemma 9 and Opial’s Lemma [43], we have the convergence theorem of the asynchronous algorithm.

**Theorem 4.** Let $(Z_{k})_{k\geq 0} \subset \mathcal{H}$ be the sequence generated by Algorithm 2 if $\eta \in (0, \eta_{\text{max}})$ for certain $\eta_{\text{max}}$ satisfying (49). If Assumptions 3, 2, and 4 hold, then $(Z_{k})_{k\geq 0}$ converges to a Fix$\mathcal{T}$-valued random variable a.s.

This theorem guarantees that, if we run the asynchronous algorithm 2 with an arbitrary starting point $Z^{0}$, then the sequence $(Z_{k})$ produced will converge to one of the solutions to problem 3 almost surely. From the upper bound of $\eta_{\text{max}}$, we can see that we must relax the update more if maximum delay $\tau$ becomes larger, or if the matrix $M$ becomes more ill-conditioned. The relative computation speed of the slowest agent will also affect the relaxation parameter.

**D. New step size rules**

For both the synchronous and asynchronous algorithms, we require the step size $\alpha$ to be less than $2\rho_{\text{min}}/L$ (Assumption 1). Both $\rho_{\text{min}}$ and $L$ involve global properties across the network: $\rho_{\text{min}}$ is related to the property of the matrix $V$ and $L$ is related to all the functions $s_{i}$. Unless they are known a priori, we need to apply a consensus algorithm to obtain them.

In this section we introduce a step size $\alpha_{i}$ for each agent $i$ that depends only on its local properties. The following theorem is a consequence of combining results of [44] Lemma 10 and the monotone operator theory.

**Theorem 5.** Let the iteration $Z_{k+1} = T Z_{k}$ be defined as
\[
\begin{align*}
{x}_{i,k+1} &= \text{prox}_{\alpha_{i}}\left(\sum_{j \in \mathcal{N}_{i}} w_{i,j} x_{j,k} - \alpha_{i} \nabla s_{i}(x_{i,k}) - \sum_{e \in \mathcal{E}_{i}} v_{e,k} y_{e,k}\right), \quad \forall i, \\
y_{e,k+1} &= y_{e,k} + \left(\sum_{e \in \mathcal{E}_{i}} v_{e,k} x_{e,k}^{i} \right), \quad \forall e \in (i, j) \in \mathcal{E}_{i},
\end{align*}
\]
with $\alpha_{i} = \frac{\mu_{i}}{L_{i} + \frac{1}{\gamma} \|w_{i}\|_{1}}$ for an arbitrary $0 < \gamma < 2$. Then under assumption 7 the operator $T$ is an averaged operator.

Because this new operator $T$ is still an averaged operator, the convergence results, Corollary 1 and Theorem 4 still hold.

**IV. NUMERICAL EXPERIMENTS**

In this part we simulate the performance of the proposed asynchronous algorithm [17]. We will compare it with its synchronous counterpart [12] (synchronous PG-EXTRA) as well as the proximal gradient descent algorithms (synchronous algorithm [21] and asynchronous algorithm [22]).

In the following experimental settings, we generate the network as follows. The location of the $n$ agents are randomly generated in a $30 \times 30$ area. If any two agents are within a distance of 15 units, they are regarded as neighbors and one edge is assumed to connect them. Once the network topology is generated, the associated weighting matrix $W$ is produced according to the Metropolis-Hastings rule [43].

In all simulations, we let each agent start a new round of update, with available neighboring variables which involves delays in general, instantly after the finish of last one. To guarantee each agent to be activated i.i.d., the computation time of agent $i$ is sampled from an exponential distribution $\exp(1/\mu_{i})$. For agent $i$, $\mu_{i}$ is set as $2 + |\mu|$ where $\mu$ follows the standard normal distribution $\mathcal{N}(0, 1)$. The communication times is also simulated. The communication time between agents are independently sampled from $\exp(1/0.6)$. After the computation time is generated, the probability $q_{i}$ can be computed accordingly.

In all curves in the following figures, the relative error $\frac{\|X^{k} - X^{*}\|_{F}}{\|X^{*}\|_{F}}$ against time is plotted, where $X^{*}$ is the exact solution to (1).

**A. Decentralized compressed sensing**

For decentralized compressed sensing, each agent $i \in \{1, 2, \ldots, n\}$ holds some measurements: $b_{i} = A_{i}x + e_{i} \in \mathbb{R}^{m_{i}}$, where $A_{i} \in \mathbb{R}^{m_{i} \times p}$ is a sensing matrix, $x \in \mathbb{R}^{p}$ is the common unknown sparse signal, and $e_{i}$ is i.i.d. Gaussian noise. The goal is to recover $x$. The number of measurements $\sum_{i=1}^{n} m_{i}$ may be less than the number of unknowns $p$, so we solve the $\ell_{1}$-regularized least squares:

\[
\min_{x} \frac{1}{n} \sum_{i=1}^{n} s_{i}(x) + r_{i}(x),
\]

where $s_{i}(x) = \frac{1}{2} \|A_{i}x - b_{i}\|_{2}^{2}$, $r_{i}(x) = \theta_{i} \|x\|_{1}$, and $\theta_{i}$ is the regularization parameter with agent $i$.

The tested network has 10 nodes and 14 edges. We set $m_{i} = 3$ for $i = 1, \ldots, 10$ and $p = 50$. The entries of $A_{i}, e_{i}$ are independently sampled from the standard normal distribution $\mathcal{N}(0, 1)$, and $A_{i}$ is normalized so that $\|A_{i}\|_{2} = 1$. The signal $x$ is generated randomly with 20% nonzero elements. We set the regularization parameters $\theta_{i} = 0.01$. The step sizes of all the four algorithms are tuned by hand and are nearly optimal. The step size $\alpha$ for both the primal synchronous algorithm [21] and the asynchronous algorithm [22] are set to be 0.05. This choice is a compromise between convergence speed and accuracy. The relaxation parameters for the asynchronous algorithm [22] are chosen to be $\eta_{i} = 0.036/\|e_{i}\|$. The step size $\alpha$ for both synchronous PG-EXTRA Algorithm 1 and asynchronous Algorithm 2 are set to be 1. The relaxation parameters for asynchronous Algorithm 2 are chosen to be $\eta_{i} = 0.0288/\|e_{i}\|$. From Fig. 2 we can see that asynchronous primal algorithm [22] is faster than its synchronous version [21], but both algorithms are far slower than synchronous PG-EXTRA Algorithm 1 and asynchronous Algorithm 2. The latter two algorithms exhibit linear convergence and Algorithm 2 converges significantly faster. Within the same period (roughly 2760ms), the two asynchronous
TABLE I: Sampled computation time in the 1st iteration.

<table>
<thead>
<tr>
<th>edge</th>
<th>time (ms)</th>
<th>(1.2)</th>
<th>(1.10)</th>
<th>(1.8)</th>
<th>(2.3)</th>
<th>(2.5)</th>
<th>(2.8)</th>
<th>(2.9)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.489</td>
<td>1.425</td>
<td>0.024</td>
<td>1.191</td>
<td>2.862</td>
<td>2.140</td>
<td>0.091</td>
</tr>
<tr>
<td>edge</td>
<td>time (ms)</td>
<td>(3.6)</td>
<td>(4.6)</td>
<td>(4.7)</td>
<td>(4.9)</td>
<td>(6.7)</td>
<td>(7.8)</td>
<td>(8.10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.429</td>
<td>0.018</td>
<td>2.359</td>
<td>2.233</td>
<td>0.003</td>
<td>1.952</td>
<td>2.412</td>
</tr>
<tr>
<td>edge</td>
<td>time (ms)</td>
<td>(2.1)</td>
<td>(10.1)</td>
<td>(8.1)</td>
<td>(3.4)</td>
<td>(5.2)</td>
<td>(8.2)</td>
<td>(9.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.626</td>
<td>1.665</td>
<td>1.672</td>
<td>1.828</td>
<td>0.509</td>
<td>4.392</td>
<td>0.617</td>
</tr>
<tr>
<td>edge</td>
<td>time (ms)</td>
<td>(6.5)</td>
<td>(6.4)</td>
<td>(7.4)</td>
<td>(9.4)</td>
<td>(7.6)</td>
<td>(8.7)</td>
<td>(10.8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.385</td>
<td>0.887</td>
<td>1.152</td>
<td>0.744</td>
<td>2.716</td>
<td>0.649</td>
<td>3.031</td>
</tr>
</tbody>
</table>

TABLE II: Sampled communication time in the 1st iteration.

algorithm finishes 21 times as many rounds of computation and communication as the synchronous counterparts, due to the elimination of waiting time.

To better illustrate the reason why the asynchronous Algorithm 2 is much faster than the synchronous Algorithm 1, we list the computation and communication time during the first iteration in Tables I and II respectively. In Table II each edge corresponds to two rounds of communication time: communications from agent i to j and from j to i. From Table II it is observed that the longest computation time is 1.152 ms. From Table II it is observed that the longest communication time is 4.592 ms. Therefore, the duration for the first iteration in the synchronous Algorithm 1 is 1.152 + 4.592 = 5.744 ms. However, in the asynchronous Algorithm 2, each agent will start a new iteration right after the finish of its previous update. The average duration for the first iteration is $\frac{\sum_{i=1}^{10} t_i}{10} = 0.443$ ms. Therefore, during the first iteration the asynchronous Algorithm 2 is 5.744/0.443 $\approx$ 13 times faster than the synchronous Algorithm 1. The data listed in Tables I and II illustrates the necessity to remove the idle time appearing in the synchronous algorithm.

Since both synchronous and asynchronous proximal-gradient-descent-type algorithms are much slower compared to the synchronous PG-EXTRA Algorithm 1 and asynchronous Algorithm 2 in the following two simulations we just show convergence performance of PG-EXTRA Algorithm 1 and asynchronous Algorithm 2.

B. Decentralized sparse logistic regression

In this subsection the tested problem is decentralized sparse logistic regression. Each agent $i \in \{1,2,\ldots,n\}$ holds local data samples $\{h_j^i,d_j^i\}_{j=1}^{m_i}$, where the subscript $i$ indicates the agent index and subscript $j$ indicates the data index. $h_j^i \in \mathbb{R}^p$ is a feature vector and $d_j^i \in \{-1,+1\}$ is the corresponding label. $m_i$ is the number of local samples kept by agent $i$. All agents in the network will cooperatively solve the sparse logistic regression problem

$$\min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{m} [s_i(x) + r_i(x)],$$

where $s_i(x) = \frac{1}{m_i} \sum_{j=1}^{m_i} \ln(1+\exp(-d_j^i (h_j^i)^\top x))$, $r_i(x) = \theta_i \|x\|_1$.

In the simulation, we set $n = 10$, $p = 50$, and $m_i = 3$ for all $i$. For local data samples $\{h_j^i,d_j^i\}_{j=1}^{m_i}$ at agent $i$, each $h_j^i$ is generated from the standard normal distribution $N(0,1)$. The step sizes of both algorithms are tuned by hand and are nearly optimal. The step sizes $\alpha$ for both synchronous PG-EXTRA Algorithm 1 and asynchronous Algorithm 2 are chosen to be $\eta_1 = 0.0224$. From Fig. 3 we can see that both algorithms exhibit convergence almost linearly and that Algorithm 2 converges significantly faster. In our experiments, within the same period, the asynchronous algorithm finishes 21 times as many rounds of computation and communication as that performed by the synchronous algorithm, due to the elimination of waiting time.

C. Decentralized low-rank matrix completion

Consider a low-rank matrix $A = [A_1,\ldots,A_n] \in \mathbb{R}^{N \times K}$ of rank $r \ll \min\{N,K\}$. In a network, each agent $i$ observes some entries of $A_i \in \mathbb{R}^{N \times K}$, $\sum_{i=1}^{n} K_i = K$. The set of observations is $\Omega = \cup_{i=1}^{n} \Omega_i$. To recover the unknown entries of $A$, we introduce a public matrix $X \in \mathbb{R}^{N \times r}$, which is
known to all agents, and a private matrix \( Y = [Y^1, \ldots, Y^n] \in \mathbb{R}^{r \times K} \), where each \( Y^i \in \mathbb{R}^{N \times K_i} \) corresponds to \( A_i \) and is held by agent \( i \). The subscript \( i \) indicates the agent index. We reconstruct \( A = XY \), which is at most rank \( r \), by recovering \( X \) and \( Y \) in a decentralized fashion. The problem is formulated as follows (see [45] for reference).

\[
\begin{align*}
\text{minimize} & \quad X, \{Y^i\}_{i=1}^n, \{K_i\}_{i=1}^n \quad \frac{1}{2} \sum_{i=1}^n \|XY^i - Z^i\|^2_F, \\
\text{subject to} & \quad (Z^i)_{ab} = (A_i)_{ab}, \forall (a, b) \in \Omega_i, \quad (50)
\end{align*}
\]

where \( Z^i \in \mathbb{R}^{N \times K_i} \) is an auxiliary matrix, and \( (Z^i)_{ab} \) is the \((a, b)\)-th element of \( A_i \).

1) **Synchronous algorithm:** [45] proposes a decentralized algorithm to solve Problem \((50)\). Let each agent hold \( X^i \) as a local copy of the public matrix \( X \), the algorithm is:

**Step 1:** Initialization. Agent \( i \) initializes \( X^{i,0} \) and \( Y^{i,0} \) as random matrices, \( Z^{i,0} \) is also initialized as a random matrix with \( (Z^{i,0})_{ab} = (A_i)_{ab} \) for any \((a, b) \in \Omega_i\).

**Step 2:** Update of \( X^i \). Each agent \( i \) updates \( X^{i,k+1} \) by solving the following average consensus problem:

\[
\begin{align*}
\text{minimize} & \quad \{X^i\}_{i=1}^n \quad \frac{1}{2} \sum_{i=1}^n \|X^i - Z^{i,k}(Y^{i,k})^\top\|^2_F, \\
\text{subject to} & \quad X^1 = X^2 = \cdots = X^n. \quad (51)
\end{align*}
\]

**Step 3:** Update of \( Y^i \). Each agent \( i \) updates \( Y^{i,k+1} \) as follows:

\[
Y^{i,k+1} = [X^{i,k+1} + P_{\Omega_i}(A_i - X^{i,k+1}Y^{i,k+1})].
\]

**Step 4:** Update of \( Z_i \). Each agent \( i \) updates \( Z_i^{k+1} = X^{i,k+1}Y^{i,k+1} + P_{\Omega_i}(A_i - X^{i,k+1}Y^{i,k+1}) \),

where \( P_{\Omega}(-) \) is defined as follows: for any matrix \( A \), if \((a, b) \in \Omega \), then \( [P_{\Omega}(A)]_{ab} = A_{ab} \); otherwise \( [P_{\Omega}(A)]_{ab} = 0 \).

In problem \((51)\) appearing at Step 2, we can let \( r_i(X^i) = \frac{1}{2} \|X^i - Z^{i,k}(Y^{i,k})^\top\|^2_F \) and \( s_i(Y^i) = 0 \) and hence problem \((51)\) falls into the general form of problem \((1)\), for which we can apply the synchronous PG-EXTRA Algorithm \([1]\) to solve it. We introduce matrices \( \{Q^e, \forall e \in E\} \) as dual variables. Instead of solving \((51)\) exactly, we just run \((12)\) once for each iteration \( k \). Therefore, Step 2 becomes

**Step 2': Update of \( X^i \).** Each agent \( i \) updates \( X^{i,k+1} \) by

\[
\begin{align*}
X^{i,k+1} &= \frac{1}{\alpha + 1} \sum_{j \in A_i} w_{ij} X^{j,k} - \sum_{e \in E_i} v_e Q^e + \alpha Z^{i,k}(Y^{i,k})^\top, \\
Q^e_{k+1} &= Q^e_k + \sum_{i,j} \left( v_e X^{i,k} + v_e X^{j,k} \right), \quad \forall e = (i, j) \in E_i.
\end{align*}
\]

2) **Asynchronous algorithm:** For the asynchronous algorithm, agent \( i \), when activated, updates \( X_i \) and \( \{Q^e, \forall e \in E_i\} \) according to Step 2' using the neighboring information it has available, then performs Step 3 and Step 4 to update its private matrices \( Y_i \) and \( Z_i \), and sends out the updated \( X_i \) and \( Q^e \) to neighbors.

We test our algorithms on a network with 20 nodes and 41 edges. In the simulation, we generate a rank 4 matrix \( A \in \mathbb{R}^{40 \times 140} \), hence each \( A_i \in \mathbb{R}^{40 \times 7} \). To generate \( A \), we first produce \( E \in \mathbb{R}^{40 \times 4} \sim N(0,1) \) and \( F \in \mathbb{R}^{140 \times 4} \sim N(0,1) \). We also produce a diagonal matrix \( D \in \mathbb{R}^{4 \times 4} \sim N(0,1) \). With \( E, F \) and \( D \), we let \( A = EDF^\top \). A total of 80% entries of \( A \) are known. To generate \( \Omega \), we first sample its entries independently from the uniform distribution \( U(0,1) \). If any entry of \( \Omega \) is less than 0.8, it will be set as 1, which indicates this entry is known; otherwise it will be set as 0.

We run the synchronous PG-EXTRA Algorithm \([1]\) and the proposed asynchronous primal-dual Algorithm \([2]\) and plot the relative error \( \|Z^{i,k} - {\Omega}(A_i)\|_F \) against time, as depicted in Fig. 4. The step sizes of both algorithms are chosen to be \( \alpha = 0.1 \). The relaxation parameters for asynchronous Algorithm \([2]\) are chosen to be \( \eta_k = \frac{0.101}{k} \). The performance of the algorithms are similar to the previous experiments. In this 20-node network, within the same period, the asynchronous algorithm finishes 29 times as many rounds of computation and communication as those finished by synchronous Algorithm \([1]\).

**V. CONCLUSIONS**

This paper developed an asynchronous, decentralized algorithm for consensus optimization. The agents in this algorithm can compute and communicate in an uncoordinated fashion; local variables are updated with possibly out-of-date information. Mathematically, the developed algorithm extends the existing algorithm PG-EXTRA by adding an explicit dual variable for each edge and to take asynchronous steps. The convergence of our algorithm is established under certain statistical assumptions. Although not all assumptions are satisfied in practice, the algorithm is practical and efficient. In particular, step size parameters are fixed and depend only on local information.

Numerical simulations were performed on both convex and nonconvex problems, and synchronous and asynchronous algorithms were compared. In addition, we introduced an asynchronous algorithm without dual variables by extending an existing algorithm and included it in the simulation. All simulation results clearly show the advantages of the developed asynchronous primal-dual algorithm.

**REFERENCES**

