2021 60th IEEE Conference on Decision and Control (CDC)
December 13-15, 2021. Austin, Texas

Self-Healing First-Order Distributed Optimization

Israel L. Donato Ridgley[1,4], Randy A. Freeman[1,3,4], and Kevin M. Lynch[2,3,4]

2021 60th IEEE Conference on Decision and Control (CDC) | 978-1-6654-3659-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CDC45484.2021.9683487

***Abstract*— We describe a parameterized family of first-order distributed optimization algorithms that enable a network of agents to collaboratively calculate a decision variable that minimizes the sum of cost functions at each agent. These algorithms are *self-healing* in that their convergence to the correct optimizer can be guaranteed even if they are initialized randomly, agents join or leave the network, or local cost functions change. We also present simulation evidence that our algorithms are self-healing in the case of dropped communication packets. Our algorithms are the first single-Laplacian methods for distributed convex optimization to exhibit all of these characteristics. We achieve self-healing by sacrificing internal stability, a fundamental trade-off for single-Laplacian methods.**

I. Introduction

The distributed optimization problem contains a network of n agents, wherein each agent calculates a decision vector that minimizes a global additive objective function of the form $f(\cdot) = \sum_i f_i(\cdot)$, where f_i denotes the local convex objective function known only to agent i. Specifically, each agent maintains a local estimate x_i of the global minimizer

$$x_{\text{opt}} = \arg\min_{\theta} \sum_i f_i(\theta), \tag{1}$$

which we assume is unique. The agents reach consensus $x_i = x_{\text{opt}}$ by computing the gradients of their local objective functions $\nabla f_i(x_i)$ and passing messages along the links of the communication network.

Distributed optimization problems of this form have broad application. For example, a distributed set of servers or sensors could perform a learning task (e.g., classification) using their local data without uploading the data to a central server for bandwidth, resiliency, or privacy reasons [1]. Swarms of robots can use distributed optimization to plan motions to solve the rendezvous problem [2].

The optimization of a collective cost function in a network setting has seen considerable interest over the last decade [3]–[10]. Recently, several authors have adapted methods from control theory to study distributed optimization algorithms as linear systems in feedback with uncertainties constrained by integral quadratic constraints (IQCs) [3], [11], [12]. These works have made it possible to more easily compare the various known algorithms across general classes of cost functions and graph topologies.

All authors are affiliated with Northwestern University, Evanston, IL 60208 USA (e-mail: israelridgley2023@u.northwestern.edu; freeman@northwestern.edu; kmlynch@northwestern.edu).

[1]Department of Electrical & Computer Engineering; [2]Department of Mechanical Engineering; [3]Northwestern Institute on Complex Systems; [4]Center for Robotics and Biosystems

This material is based upon work supported by the National Science Foundation under Grant No. CMMI-2024774.

The work [3] uses these techniques to describe several recent distributed optimization algorithms within a common framework, then describes a new algorithm (SVL) within that framework that achieves a superior worst-case convergence rate. However, all of the algorithms considered in [3] share a common undesirable trait: to reach the correct solution, their states must start on a particular subspace of the overall global state space and remain in it at every time step. If for any reason the state trajectories leave this subspace (e.g., incorrect initialization, dropped packets, computation errors, agents leaving the network, changes to objective functions due to continuous data collection), then the system will no longer converge to the minimizer. Such methods cannot automatically recover from disturbances or other faults that displace their trajectories from this subspace; in other words, they are not *self-healing*.

In this paper, we extend our results from dynamic average consensus estimators [13], [14] to design a family of distributed optimization algorithms whose trajectories need not evolve on a pre-defined subspace. We call such algorithms *self-healing*. In practice, this means that our algorithms can be arbitrarily initialized, agents can join or leave the network at will, and agents can change their objective functions as necessary, such as when they collect new data. An important consequence of the self-healing property is that our algorithm can be modified with a low-overhead packet-loss protocol which allows the algorithm to recover from lost or corrupted packets.

We refer to distributed optimization algorithms that communicate one or two variables (having the same vector dimension as the decision variable x_i) per time step as single- and double-Laplacian methods, respectively. Examples of single-Laplacian methods are SVL and NIDS, while examples of double-Laplacian methods are uEXTRA and DIGing [3], [5]–[8]. Our algorithms are the first self-healing single-Laplacian methods for convex optimization that converge to the exact (rather than an approximate) solution (see [13], [15] for the specific case of average consensus). They achieve self-healing by sacrificing internal stability, a fundamental trade-off for single-Laplacian methods. In particular, each agent will have an internal state that grows linearly in time in steady state, but because such growth is not exponential it will not cause any numerical issues unless the algorithm runs over a long time horizon. Double-Laplacian methods can achieve both internal stability and self-healing, but they require twice as much communication per time step and converge no faster than single-Laplacian methods [3], [14].

II. Preliminaries and Main Results

A. Notation and terminology

We adopt notation similar to that in [3]. Let $\mathbb{1}_n$ be the n-dimensional column vector of all ones, I_n be the identity matrix in $\mathbb{R}^{n\times n}$, and $\Pi_n = \frac{1}{n}\mathbb{1}\mathbb{1}^\intercal$ be the projection matrix onto the vector $\mathbb{1}_n$. We drop the subscript n when the size is clear from context. We refer to the one-dimensional linear subspace of $\mathbb{R}^n$ spanned by the vector $\mathbb{1}_n$ as the *consensus direction* or the *consensus subspace*. We refer to the $(n-1)$-dimensional subspace of $\mathbb{R}^n$ associated with the projection matrix $(I_n - \Pi_n)$ as the *disagreement direction* or subspace.

The variable z represents the complex frequency of the z-transform. Subscripts denote the agent index whereas superscripts denote the time index. The symbol $\otimes$ represents the Kronecker product. A^+ indicates the Moore-Penrose inverse of A. Symmetric quadratic forms $x^\intercal Ax$ are written as $[\star]^\intercal Ax$ to save space when x is long. The local decision variables are d-dimensional and represented as a row vector, i.e., $x_i \in \mathbb{R}^{1\times d}$, and the local gradients are a map $\nabla f_i : \mathbb{R}^{1\times d} \to \mathbb{R}^{1\times d}$. The symbol $||\cdot||$ refers to the Euclidean norm of vectors and the spectral norm of matrices.

We model a network of n agents participating in a distributed computation as a weighted digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, ..., n\}$ is the set of n nodes (or vertices) and $\mathcal{E}$ is the set of edges such that if $(i, j) \in \mathcal{E}$ then node i can receive information from j. We make use of the *weighted graph Laplacian* $\mathcal{L} \in \mathbb{R}^{n\times n}$ associated with $\mathcal{G}$ such that $-\mathcal{L}_{ij}$ is the weight on edge $(i, j) \in \mathcal{E}$, $\mathcal{L}_{ij} = 0$ when $(i, j) \notin \mathcal{E}$ and $i \neq j$, and the diagonal elements of $\mathcal{L}$ are $\mathcal{L}_{ii} = -\sum_{j\neq i} \mathcal{L}_{ij}$, so that $\mathcal{L}\mathbb{1} = 0$. We define $\sigma = ||I - \Pi - \mathcal{L}||$, which is a parameter related to the edge weights and the graph connectivity.

Throughout this work we stack variables and objective functions such that

$$x^k = \begin{bmatrix} x_1^k \\ \vdots \\ x_n^k \end{bmatrix} \in \mathbb{R}^{n\times d} \quad \text{and} \quad \nabla F(x^k) = \begin{bmatrix} \nabla f_1(x_1^k) \\ \vdots \\ \nabla f_n(x_n^k) \end{bmatrix} \in \mathbb{R}^{n\times d}.$$

B. Assumptions

(A1) Given $0 < m \leq L$, we assume that the local gradients are sector bounded on the interval (m, L), meaning that they satisfy the quadratic inequality

$$[\star]^\intercal \begin{bmatrix} -2mLI_d & (L+m)I_d \\ (L+m)I_d & -2I_d \end{bmatrix} \begin{bmatrix} (x_i - x_{\text{opt}})^\intercal \\ (\nabla f_i(x_i) - \nabla f_i(x_{\text{opt}}))^\intercal \end{bmatrix} \geq 0$$

for all $x_i \in \mathbb{R}^{1\times d}$, where x_{opt} satisfies $\sum_{i=1}^n \nabla f_i(x_{\text{opt}}) = 0$. We define the condition ratio as $\kappa = \frac{L}{m}$, which captures the variation in the curvature of the objective function.

(A2) The graph $\mathcal{G}$ is strongly connected.

(A3) The graph $\mathcal{G}$ is weight balanced, meaning that $\mathbb{1}^\intercal\mathcal{L} = 0$.

(A4) The weights of $\mathcal{G}$ are such that $\sigma = ||I - \Pi - \mathcal{L}|| < 1$.

Remark 1. Assumption **(A1)** is known as a *sector IQC* (for a more detailed description see [11]) and is satisfied when the local objective functions are m-strongly convex with L-Lipschitz continuous gradients.

Remark 2. Throughout this paper we assume without loss of generality that the dimension of the local decision and state variables is $d = 1$.

Remark 3. Under appropriate conditions on the communications network, the agents can self-balance their weights in a distributed way to satisfy **(A3)**; for example, they can use a scalar consensus filter like push-sum (see Algorithm 12 in [16]). Additionally, agents can enforce that their weighted in-degrees (and thus their weighted out-degrees) sum to less than one in order to satisfy **(A4)**.

C. Results

In the following sections we present a parameterized family of distributed, synchronous, discrete-time algorithms to be be run on each agent such that, under assumptions **(A1)**-**(A4)**, we achieve the following:

Accurate convergence: in the absence of disturbances or other faults, the local estimates x_i converge to the optimizer x_{opt} with a linear rate.

Self-healing: the system state trajectories need not evolve on a pre-defined subspace and will recover from events such as arbitrary initialization, temporary node failure, computation errors, or changes in local objectives.

Packet-loss protocol: if agents are permitted a state of memory for each of their neighbors, they can implement a packet-loss protocol that allows computations to continue in the event communication is temporarily lost. This extends the self-healing of the network to packet loss in a way that is not possible if the system state trajectories are required to evolve on a pre-defined subspace.

First we present the synthesis and analysis of our algorithm along with its performance relative to existing methods. Then we demonstrate via simulation that our algorithm still converges under high rates of packet loss.

III. Synthesis of Self-Healing Distributed Optimization Algorithms

A. Canonical first-order methods

As a motivation for our algorithms, we use the canonical form first described in [17] and later used as the SVL template [3]. When the communication graph is constant, many single-Laplacian methods such as SVL, EXTRA and Exact Diffusion can be described in this form [3], [4], [9], [10], [17]. Algorithms representable by the SVL template can also be expressed as a state space system G in feedback with an uncertain and nonlinear block containing the objective function gradients $\nabla F(\cdot)$ and the Laplacian $\mathcal{L}$ shown in Figure 1, where

$$G = \left[\begin{array}{c|c|c} A & B_u & B_v \\ \hline C_x & D_{xu} & D_{xv} \\ \hline C_y & D_{yu} & D_{yv} \end{array}\right] = \left[\begin{array}{cc|c|c} 1 & \beta & -\alpha & -\gamma \\ 0 & 1 & 0 & -1 \\ \hline 1 & 0 & 0 & -\delta \\ \hline 1 & 0 & 0 & 0 \end{array}\right] \otimes I_n. \quad (2)$$

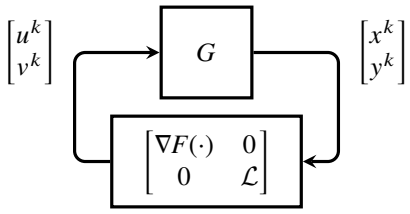

Fig. 1. Distributed optimization algorithms represented as a feedback interconnection of an LTI system G and an uncertain block containing the gradients and the graph Laplacian.

The LTI system G has two states w_1^k and w_2^k, inputs u^k and v^k, and outputs x^k and y^k such that

$$u_i^k = \nabla f_i(x_i^k), \quad v_i^k = \sum_{j=1}^{n} \mathcal{L}_{ij}^k y_j^k. \tag{3}$$

We would like to alert the reader to a small notational difference between our work and [3]: in this work, the variable x is the input to the gradients and the variable y is the input to the Laplacian (as shown in Figure 1), whereas in [3] y is the input to the gradients and z is the input to the Laplacian. (We cannot use z because we already use it as the frequency variable of the z-transform.)

Algorithms described by Figure 1 contain two discrete-time integrators in the LTI block G: one integrator is necessary to force the steady state error to zero, and the other is responsible for the agents coming to consensus. In the SVL template, the output of the graph Laplacian feeds into the integrator responsible for consensus.

Algorithms representable by the SVL template, and more broadly all existing first-order methods with a single Laplacian, require that the system trajectories evolve on a pre-defined subspace. From our work with average consensus estimators [13], [14], we know that these drawbacks arise from the positional order of the Laplacian and integrator blocks. When the Laplacian feeds into the integrator, the output of the Laplacian cannot drive the integrator state away from the consensus subspace, which leads to an observable but uncontrollable mode. If the integrator state is initialized on the consensus subspace, or it is otherwise disturbed there, the estimate of the optimizer will contain an uncorrectable error. Switching the order of the Laplacian and integrator renders the integrator state controllable but causes it to become inherently unstable because the integrator output in the consensus direction is disconnected from the rest of the system. We exploit this trade-off to develop self-healing distributed optimization algorithms with only a single Laplacian.

B. Factorization and integrator location

To switch the order of the Laplacian and the integrator, we first factor an integrator out of the $G(z)$ block of Figure 1,

$$G(z) = \begin{bmatrix} \dfrac{-\alpha}{z-1} & -\dfrac{\delta z^2 + (\gamma - 2\delta)z + (\beta + \delta - \gamma)}{(z-1)^2} \\ \dfrac{-\alpha}{z-1} & -\dfrac{\gamma z + (\beta - \gamma)}{(z-1)^2} \end{bmatrix} \otimes I_n \tag{4}$$

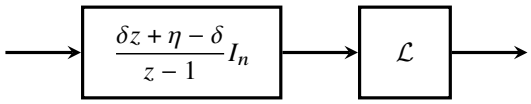

Fig. 2. The output of the integrator now feeds into the Laplacian, converting an uncontrollable and observable mode in the original SVL template to a controllable and unobservable one.

$$= \begin{bmatrix} \dfrac{-\alpha}{z-1} & -\dfrac{z-1+\zeta}{z-1} \\ \dfrac{-\alpha}{z-1} & \dfrac{-\gamma z - (\beta - \gamma)}{(z-1)(\delta z + \eta - \delta)} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \dfrac{\delta z + \eta - \delta}{z-1} \end{bmatrix} \otimes I_n, \tag{5}$$

where

$$\eta = \gamma - \delta\zeta \ \text{ and } \ \zeta = \begin{cases} \dfrac{\beta}{\gamma} & \text{if } \delta = 0 \\ \dfrac{\gamma - \sqrt{\gamma^2 - 4\beta\delta}}{2\delta} & \text{otherwise.} \end{cases} \tag{6}$$

Swapping the order of the component matrices yields our new family of algorithms (where G_s replaces G):

$$G_s(z) = \begin{bmatrix} 1 & 0 \\ 0 & \dfrac{\delta z + \eta - \delta}{z-1} \end{bmatrix} \begin{bmatrix} \dfrac{-\alpha}{z-1} & -\dfrac{z-1+\zeta}{z-1} \\ \dfrac{-\alpha}{z-1} & \dfrac{-\gamma z - (\beta - \gamma)}{(z-1)(\delta z + \eta - \delta)} \end{bmatrix} \otimes I_n$$

$$= \begin{bmatrix} -\alpha \dfrac{1}{z-1} & -\dfrac{z-1+\zeta}{z-1} \\ -\alpha \dfrac{\delta z + \eta - \delta}{(z-1)^2} & -\dfrac{\gamma z + \beta - \gamma}{(z-1)^2} \end{bmatrix} \otimes I_n. \tag{7}$$

Now the output of the integrator feeds directly into the Laplacian, as depicted in Figure 2. We assume that our parameter choices satisfy

$$\gamma^2 \geq 4\beta\delta \tag{8}$$

so that the zeros of G_s remain real and thus the system can be implemented with real-valued signals. The corresponding distributed algorithm is described in Algorithm 1, where w_1 and w_2 are the internal states of G_s, and the compact state space form is

$$G_s = \left[\begin{array}{cc|cc} 1 & 0 & -\alpha & -\zeta \\ 1 & 1 & 0 & -1 \\ \hline 1 & 0 & 0 & -1 \\ \hline \delta & \eta & 0 & 0 \end{array}\right] \otimes I_n. \tag{9}$$

Remark 4. The factorization in (5) is not unique; we chose it because it leads to a method still having only two internal states per agent. There may be other useful factorizations.

Remark 5. In contrast to algorithms like SVL, Algorithm 1 does not require specific initial conditions and system trajectories are not restricted to a pre-defined subspace. If agents change their local objective functions or drop out of the computation, the system does not need to be reset and the system will converge to the new minimizer. In the case of agents dropping out, the connection topology must still be strongly connected, otherwise Algorithm 1 with a weight balancer will converge to the minimizer for only a subset of the objective functions and consensus across the network will not be achieved.

Algorithm 1: Self-Healing Distributed Gradient Descent

Initialization:
Each agent $i \in \{1, ..., n\}$ chooses $w_{1i}^0, w_{2i}^0 \in \mathbb{R}^{1\times d}$ arbitrarily. $\mathcal{L} \in \mathbb{R}^{n\times n}$ is the graph Laplacian.
for $k = 0, 1, 2, ...$ **do**
  for $i \in \{1, ..., n\}$ **do**
    Local communication
    $y_i^k = \delta w_{1i}^k + \eta w_{2i}^k$
    $v_i^k = \sum_{j=1}^n \mathcal{L}_{ij} y_j^k$
    Local gradient computation
    $x_i^k = w_{1i}^k - v_i^k$ // Update the optimizer estimate.
    $u_i^k = \nabla f_i(x_i^k)$
    Local state update
    $w_{1i}^{k+1} = w_{1i}^k - \alpha u_i^k - \zeta v_i^k$
    $w_{2i}^{k+1} = w_{1i}^k + w_{2i}^k - v_i^k$
  end
end
return x_i

IV. Stability and Convergence Rates Using IQCs

A. Projection onto the disagreement subspace

As written, our family of algorithms is internally unstable. We use the projection matrix $(I-\Pi)$ to eliminate the instability from the global system without affecting x^k. This procedure is a centralized calculation that cannot be implemented in a distributed fashion, but it allows us to analyze the convergence properties of the distributed algorithm.

Consider the steady-state values $(w_1^\star, x^\star, u^\star, v^\star)$ and suppose w_2^k contains a component in the $\mathbb{1}$ direction. Then that component does not affect the aforementioned values because it is an input to the Laplacian $\mathcal{L}$ (and lies in its nullspace); however, it grows linearly in time due to the w_2 update. Thus the system has an internal instability that is unobservable from the output of the bottom block in Figure 1. Since the component of w_2^k in the consensus direction is unobservable to the variables (w_1^k, x^k, u^k, v^k), we can throw it away without affecting their trajectories. Using the transformation $\hat{w}_2^k = (I-\Pi)w_2^k$, our state updates become

$$w_1^{k+1} = w_1^k - \alpha u^k - \zeta v^k \tag{10}$$
$$\hat{w}_2^{k+1} = (I-\Pi)w_1^k + (I-\Pi)\hat{w}_2^k - (I-\Pi)v^k \tag{11}$$
$$x^k = w_1^k - v^k \tag{12}$$
$$\hat{y}^k = \delta w_1^k + \eta \hat{w}_2^k \tag{13}$$
$$u^k = \nabla F(x^k) \tag{14}$$
$$v^k = \mathcal{L}\hat{y}^k, \tag{15}$$

where y^k was replaced with $\hat{y}^k$ in (13) and (15) to accommodate $\hat{w}_2^k$. These updates lead to the state-space system

$$G_m = \left[\begin{array}{cc|cc} I & 0 & -\alpha I & -\zeta I \\ I-\Pi & I-\Pi & 0 & -(I-\Pi) \\ \hline I & 0 & 0 & -I \\ \hline \delta I & \eta I & 0 & 0 \end{array}\right]. \tag{16}$$

B. Existence and optimality of a fixed point

Now that we have eliminated the inherent instability of the global system, we can state the following about the fixed points:

Theorem 1. *For the system described by G_m, there exists at least one fixed point $(w_1^\star, \hat{w}_2^\star, x^\star, \hat{y}^\star, u^\star, v^\star)$, and any such fixed point has $x^\star$ in the consensus subspace such that $x_i^\star = x_{\text{opt}}$ for all $i \in \{1, \ldots, n\}$, i.e., any fixed point of the system is optimal.*

Proof. First, assume that the fixed point $(w_1^\star, \hat{w}_2^\star, x^\star, \hat{y}^\star, u^\star, v^\star)$ exists. To prove that the variable $x^\star$ lies in the consensus direction, we show that $(I-\Pi)x^\star = 0$. From (11) and (12) we have that

$$(I-\Pi)w_1^\star = (I-\Pi)v^\star \tag{17}$$
$$(I-\Pi)x^\star = (I-\Pi)w_1^\star - (I-\Pi)v^\star \tag{18}$$
$$= 0. \tag{19}$$

Thus $x_i^\star = x_j^\star$ for all $i, j \in \{1, \ldots, n\}$. Next we show that $x_i^\star = x_{\text{opt}}$. From (10) then plugging in (15), we have

$$-\alpha u^\star - \zeta v^\star = 0 \tag{20}$$
$$u^\star = -\frac{\zeta}{\alpha} v^\star = -\frac{\zeta}{\alpha}\mathcal{L}\hat{y}^\star \tag{21}$$
$$\mathbb{1}^\top u^\star = -\frac{\zeta}{\alpha}\mathbb{1}^\top \mathcal{L}\hat{y}^\star \tag{22}$$
$$\sum_{i=1}^n u_i^\star = 0 \tag{23}$$
$$\rightarrow \sum_{i=1}^n \nabla f_i(x_i^\star) = 0 \tag{24}$$
$$\rightarrow x_i^\star = x_{\text{opt}} \;\forall\; i \in \{1, \ldots, n\}. \tag{25}$$

Thus any fixed point is optimal.

Next, to construct a fixed point we define

$$\begin{aligned} x^\star &= \mathbb{1}x_{\text{opt}}, \qquad u^\star = \nabla f(x^\star) \\ v^\star &= -\frac{\alpha}{\zeta}u^\star, \quad w_1^\star = x^\star + v^\star. \end{aligned} \tag{26}$$

Then $\hat{w}_2^\star$ is the solution to the equation

$$\zeta\eta\mathcal{L}\hat{w}_2^\star = -\alpha(I - \delta\mathcal{L})u^\star. \tag{27}$$

Since $\hat{w}_2^k = \mathcal{L}^+\mathcal{L}w_2^k$ (i.e., $\hat{w}_2^k$ is in the row space of $\mathcal{L}$), we write $\hat{w}_2^\star$ in closed form as

$$\hat{w}_2^\star = \frac{\alpha}{\zeta\eta}\mathcal{L}^+(\delta L - I)u^\star. \tag{28}$$

Finally, setting $\hat{y}^\star = \delta w_1^\star + \eta\hat{w}_2^\star$ completes the proof. □

Remark 6. If the graph is switching but converges in time such that the limit of the sequence of Laplacians exists, as with a weight balancer, then a solution to (27) still exists and an optimal fixed point can still be found. Furthermore, the proof techniques in the following section still hold for switching Laplacians (see [3] for more information).

C. Convergence

Following the approaches in [3], [11], [12], we prove stability using a set of linear matrix inequalities. First we split our modified system from (16) into consensus and disagreement components. We define

$$A_m = A_p \otimes \Pi + A_q \otimes (I - \Pi) \tag{29}$$

$$B_{mu} = B_{pu} \otimes \Pi + B_{qu} \otimes (I - \Pi) \tag{30}$$

$$B_{mv} = B_{pv} \otimes \Pi + B_{qv} \otimes (I - \Pi) \tag{31}$$

$$\begin{aligned} A_p &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad B_{pu} = \begin{bmatrix} -\alpha \\ 0 \end{bmatrix}, \quad B_{pv} = \begin{bmatrix} -\zeta \\ 0 \end{bmatrix} \\ A_q &= \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad B_{qu} = \begin{bmatrix} -\alpha \\ 0 \end{bmatrix} \quad B_{qv} = \begin{bmatrix} -\zeta \\ -1 \end{bmatrix}. \end{aligned} \tag{32}$$

We also define the matrices

$$M_0 = \begin{bmatrix} -2mL & L+m \\ L+m & -2 \end{bmatrix} \text{ and } M_1 = \begin{bmatrix} \sigma^2 - 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{33}$$

Notice that M_0 is associated with the sector bound from **(A1)** and that M_1 is associated with the $(1-\sigma, 1+\sigma)$ sector bound on $\mathcal{L}$ with inputs from the disagreement subspace.

We now make a statement analogous to Theorem 10 in [3].

Theorem 2. *If there exists $P, Q \in \mathbb{R}^{2\times 2}$, $\lambda_0, \lambda_1 \in \mathbb{R}$, and $\rho \in (0,1)$, with $P, Q \succ 0$ and $\lambda_0, \lambda_1 \geq 0$ such that*

$$[\star]^\intercal \left[\begin{array}{cc|c} P & 0 & 0 \\ 0 & -\rho^2 P & 0 \\ \hline 0 & 0 & \lambda_0 M_0 \end{array}\right] \left[\begin{array}{cc} A_p & B_{pu} \\ I & 0 \\ \hline C_x & D_{xu} \\ 0 & I \end{array}\right] \leq 0, \tag{34}$$

$$[\star]^\intercal \left[\begin{array}{cc|c|c} Q & 0 & 0 & 0 \\ 0 & -\rho^2 Q & 0 & 0 \\ \hline 0 & 0 & \lambda_0 M_0 & 0 \\ \hline 0 & 0 & 0 & \lambda_1 M_1 \end{array}\right] \left[\begin{array}{ccc} A_q & B_{qu} & B_{qv} \\ I & 0 & 0 \\ \hline C_x & D_{xu} & D_{xv} \\ 0 & I & 0 \\ \hline C_y & D_{yu} & D_{yv} \\ 0 & 0 & I \end{array}\right] \leq 0, \tag{35}$$

then the following is true for the trajectories of G_m:

$$\left\| \begin{bmatrix} w_1^k - w_1^\star \\ \hat{w}_2^k - \hat{w}_2^\star \end{bmatrix} \right\| \leq \sqrt{\text{cond}(T)}\rho^k \left\| \begin{bmatrix} w_1^0 - w_1^\star \\ \hat{w}_2^0 - \hat{w}_2^\star \end{bmatrix} \right\| \tag{36}$$

for a fixed point $(w_1^\star, \hat{w}_2^\star, x^\star, \hat{y}^\star, u^\star, v^\star)$, where $T = P \otimes \Pi_n + Q \otimes (I_n - \Pi_n)$ and $\text{cond}(T) = \frac{\lambda_{\max}(T)}{\lambda_{\min}(T)}$ is the condition number of T. Thus the output x^k of Algorithm 1 converges to the optimizer with the linear rate ρ.

Proof. Equation (36) follows directly from Theorem 4 of [11]. Since the states of G_m are converging at a linear rate ρ, the rest of the signals in the system (including x^k) converge to an optimal fixed point at the same rate. Additionally, the trajectories of G_m and G_s (Algorithm 1) are the same, save for $\hat{w}_2^k$ and $\hat{y}^k$, so x^k in Algorithm 1 also converges to the optimizer with linear rate ρ. □

To test the performance of our algorithm, we used the parameters $\beta = 0.5, \gamma = 1, \delta = 0.5$. These parameters were inspired by the NIDS/Exact Diffusion parameters presented in [17]; however, we have done no work to find parameters that optimize the convergence rate. We then solved the LMIs (34) and (35) using Convex.jl [18] with the MOSEK solver [19], performing a bisection search on ρ to find the minimum worst-case convergence rate for a given κ, σ, and α. We used Brent's method from Optim.jl [20] to determine the optimal α. We plot our results for $\kappa = 10$ in Figure 3 and include the results for SVL (reproduced from [3]) for comparison. Our algorithm with these parameter choices achieves the same performance as NIDS for the NIDS parameter choice $\mu = 1$ as shown in [3]. The worst-case convergence rate of our algorithm is subject to the same lower bound, $\rho \geq \max(\frac{\kappa-1}{\kappa+1}, \sigma)$, found in [3].

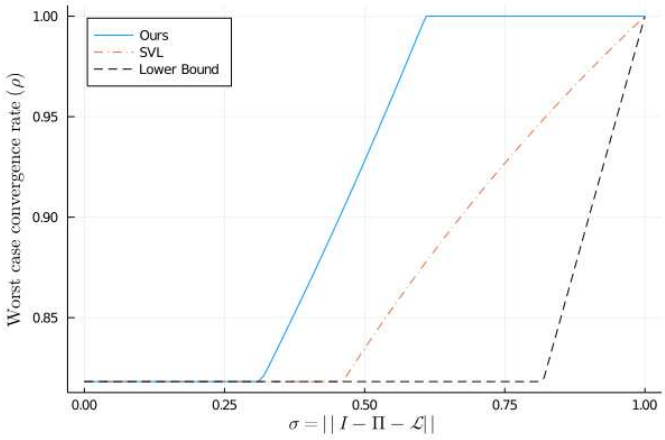

Fig. 3. Performance of our algorithm compared with SVL for $\kappa = 10$ and $\sigma \in [0, 1)$. Our NIDS-inspired parameter choices result in performance identical to that of NIDS in [3]. We have not made any attempts to choose "optimal" parameters like those of SVL.

Remark 7. In Algorithm 1, the signal x^k converges to the optimizer with linear rate ρ but the internal signals w_2^k and y^k grow only linearly. Therefore, w_2^k and y^k will not become large enough to cause overflow errors until well after the algorithm has converged to the optimizer.

Remark 8. We tested the convergence rates for our algorithm with Zames-Falb IQCs in place of Sector IQCs but saw no improvement.

V. Self-Healing Despite Packet Loss

A. Packet-loss protocol

We next give our agents some additional memory so that they can substitute previously transmitted values when a packet is lost. Each agent $i \in \{1, \ldots, n\}$ maintains an edge state e_{ij}^k for each $j \in \mathcal{N}_{\text{in}}(i)$ (the set of neighbors who transmit to i). Whenever agent i receives a message from agent j, it updates the state e_{ij} accordingly; however, if at time k no message from neighbor j is received, agent i must estimate what would have likely been transmitted. One potential strategy is to substitute in the last message received, but because y_j is growing linearly in quasi steady state, this naive strategy would ruin steady-state accuracy. Instead we must account for the linear growth present in our algorithm,

Algorithm 2: Self-Healing Distributed Gradient Descent with Packet-loss protocol

Initialization: Each agent $i \in \{1, ..., n\}$ chooses $w_{1i}^0, w_{2i}^0 \in \mathbb{R}^{1\times d}$ arbitrarily. $\mathcal{L} \in \mathbb{R}^{n\times n}$ is the graph Laplacian. All e_{ij} are initialized the first time a message is received from a neighbor.

for $k = 0, 1, 2, ...$ **do**
  for $i \in \{1, ..., n\}$ **do**
    Local communication
    $y_i^k = \delta w_{1i}^k + \eta w_{2i}^k$
    for $j \in \mathcal{N}_{in}(i)$ **do**
      if *Packet from* j *received by* i **then**
        $e_{ij}^k = y_j^k$
      else
        $e_{ij}^k = \eta x_i^{k-1} + e_{ij}^{k-1}$
      end
    end
    $v_i^k = \sum_{j=1}^n \mathcal{L}_{ij} e_{ij}^k$
    Local gradient computation
    $x_i^k = w_{1i}^k - v_i^k$ // Update the optimizer estimate.
    $u_i^k = \nabla f_i(x_i^k)$
    Local state update
    $w_{1i}^{k+1} = w_{1i}^k - \alpha u_i^k - \zeta v_i^k$
    $w_{2i}^{k+1} = w_{1i}^k + w_{2i}^k - v_i^k$
  end
end
return x_i

which we can do by analyzing the quantity $y_j^k - y_j^{k-1}$ at the quasi fixed point $(w_1^\star, x^\star, u^\star, v^\star)$:

$$y_j^k - y_j^{k-1} = \delta(w_{1j}^\star - w_{1j}^\star) + \eta(w_{2j}^k - w_{2j}^{k-1}) \quad (37)$$
$$= \eta(w_{1j}^\star - v_j^\star) \quad (38)$$
$$= \eta x_j^\star \approx \eta x_i^k \quad (39)$$

Therefore, when a packet is not received by a neighbor, agent i scales its estimate of the optimizer and adds it to its previously received (or estimated) message. The packet-loss protocol is summarized in Algorithm 2. By construction, the modifications included in Algorithm 2 will not alter the quasi fixed points of Algorithm 1, though we do not have a stability condition like Theorem 2 to present at this time. Instead, we show simulation evidence that Algorithm 2 does indeed converge, and packet loss does not appear to have a substantial impact on the convergence rate, even when the rate of packet loss is large. In the absence of dropped packets, the state trajectories of Algorithm 2 are equivalent to those of Algorithm 1.

Remark 9. Algorithm 2 can be modified to include a forgetting factor q. If agent i does not receive a packet from neighbor j in q time steps, then agent i assumes that the communication link has been severed and clears e_{ij} from memory.

B. Classification example

To test the performance of our algorithm under packet loss, we solved a classification problem using the COSMO chip dataset [21]. The problem setup is as follows: a network of n agents would like to collaboratively compute a binary classifier that identifies whether or not a computer chip will pass a quality assurance test using data gathered independently by each agent in the network; furthermore, they would like to do so in a distributed fashion without sharing their datasets. In order to simulate this problem, we divide the COSMO chip dataset into n local subsets and denote agent i's set of local data indices as S_i. To improve the performance of their classifier, the agents employ a polynomial embedding where each 2-dimensional data point $d_j = [d_{j1}, d_{j2}]$ is embedded in a 28-dimensional space given by

$$M(d_j) = [1, d_{j1}, d_{j2}, d_{j1}^2, d_{j1}d_{j2}, d_{j2}^2, d_{j1}^3, \ldots, d_{j1}d_{j2}^5, d_{j2}^6].$$

The agents use the logistic loss function with L_2-regularization, yielding local cost functions given by

$$f_i(x_i) = \sum_{j\in S_i} \log(1 + e^{-l_j x_i^\intercal M(d_j)}) + \frac{1}{n}||x_i||^2, \quad (40)$$

where $l_j \in \{-1, 1\}$ is the label of data point j, and each agent computes the label of unseen data by using the operation $l = \text{sgn}(x_i^\intercal M(d))$. Using the cost in (40), the corresponding sector bound (m, L) is approximated as $m = \frac{2}{n}$ and

$$L_i \leq \left\|\frac{2}{n}I + \frac{1}{4}M_i^\intercal M_i\right\|, \quad L = \max_i L_i, \quad (41)$$

where the rows of M_i are $M(d_j)$ for $j \in S_i$. The agents' connection topology is described by an $n = 7$ node directed ring lattice, shown in Figure 4, such that $(i, j) \in \mathcal{E}$ when $j \in \{i + 1, i + 3, i + 5\} \mod n$. All edge weights in the graph are set to 1/4 and $\sigma = ||I - \Pi - \mathcal{L}|| = 0.562$.

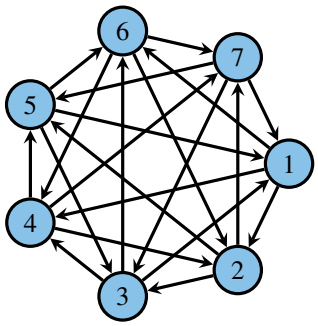

Fig. 4. The directed network topology for the classification example. All edge weights are 1/4.

Using (m, L) and σ, we computed the optimal step size α for our algorithm using Brent's method and computed the SVL parameters as detailed in [3]. We then simulated both Algorithm 2 and SVL with and without packet loss and took the maximum error between the distributed algorithms and a centralized solution found using Convex.jl and MOSEK. We ran Algorithm 2 using random initial conditions on the interval $[0, 1]$ and the SVL algorithm using zero initial conditions. For the packet loss run of SVL, we held the previous message on each edge so that the fixed points would be unaffected. At each time step, packets had a 30% chance of being lost, independent of each other and time. The results of these simulations are shown in Figure 5. In this scenario, Algorithm 2 with

lossy channels still converges to the optimum at a similar rate as Algorithm 2 with lossless channels, despite the high rate of packet loss that causes SVL to converge with high error.

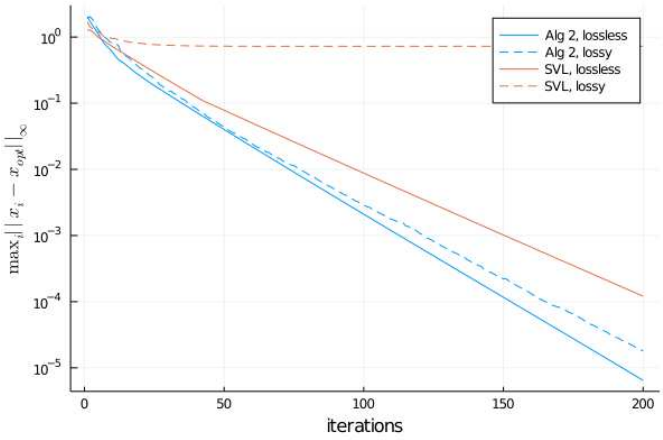

Fig. 5. Simulation of Algorithm 2 and SVL in lossless as well as lossy channels. The lossy channels are modeled with an independent 30% packet loss. Error is the maximum error.

VI. Summary and Future Work

In this paper, we demonstrated the existence of a parameterized family of first-order algorithms for distributed optimization that do not require system trajectories to evolve on a pre-defined subspace, despite having a single communicated variable. These algorithms are self-healing; they do not require the system to be initialized precisely and will recover from events such as agents dropping out of the network or changes to objective functions that might otherwise introduce uncorrectable errors. Furthermore, our algorithms can be augmented with our packet-loss protocol, thereby allowing the system to converge to the optimizer even in the presence of heavily lossy communication channels. Our algorithms converge with a linear rate to the optimizer but contain an internal instability that grows linearly in time; however, this instability is unlikely to cause issues unless run over long time horizons.

There is much left to investigate: we need to characterize the properties of other factorizations of $G(z)$ in (4), and possible factorizations of algorithms that are not subsumed by the SVL template. We need to explore the parameter space of the algorithm presented in this paper and, particularly, investigate if an optimization like that used to find the SVL parameters can be carried out. We need to devise a formal proof that Algorithm 2 still converges in the presence of packet loss. Finally, we will consider the important practical issue of adapting our algorithm to the case of asynchronous updates.

References

[1] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines.," *Journal of Machine Learning Research*, vol. 11, no. 5, 2010.

[2] G. G. Rigatos, "Distributed gradient and particle swarm optimization for multi-robot motion planning," English, *Robotica*, vol. 26, no. 3, pp. 357–370, May 2008, Copyright - Cambridge University Press; Last updated - 2015-08-15.

[3] A. Sundararajan, B. Van Scoy, and L. Lessard, "Analysis and design of first-order distributed optimization algorithms over time-varying graphs," *IEEE Transactions on Control of Network Systems*, vol. 7, no. 4, pp. 1597–1608, 2020.

[4] W. Shi, Q. Ling, G. Wu, and W. Yin, "Extra: An exact first-order algorithm for decentralized consensus optimization," *SIAM Journal on Optimization*, vol. 25, no. 2, pp. 944–966, 2015. eprint: https://doi.org/10.1137/14096668X. [Online]. Available: https://doi.org/10.1137/14096668X.

[5] Z. Li, W. Shi, and M. Yan, "A decentralized proximal-gradient method with network independent step-sizes and separated convergence rates," *IEEE Transactions on Signal Processing*, vol. 67, no. 17, pp. 4494–4506, 2019.

[6] D. Jakovetić, "A unification and generalization of exact distributed first-order methods," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 1, pp. 31–46, 2019.

[7] A. Nedić, A. Olshevsky, and W. Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM Journal on Optimization*, vol. 27, no. 4, pp. 2597–2633, 2017. eprint: https://doi.org/10.1137/16M1084316. [Online]. Available: https://doi.org/10.1137/16M1084316.

[8] G. Qu and N. Li, "Harnessing smoothness to accelerate distributed optimization," *IEEE Transactions on Control of Network Systems*, vol. 5, no. 3, pp. 1245–1260, 2018.

[9] K. Yuan, B. Ying, X. Zhao, and A. H. Sayed, "Exact diffusion for distributed optimization and learning—part i: Algorithm development," *IEEE Transactions on Signal Processing*, vol. 67, no. 3, pp. 708–723, 2019.

[10] ——, "Exact diffusion for distributed optimization and learning—part ii: Convergence analysis," *IEEE Transactions on Signal Processing*, vol. 67, no. 3, pp. 724–739, 2019.

[11] L. Lessard, B. Recht, and A. Packard, "Analysis and design of optimization algorithms via integral quadratic constraints," *SIAM J. Optim.*, vol. 26, no. 1, pp. 57–95, 2016.

[12] A. Sundararajan, B. Hu, and L. Lessard, "Robust convergence analysis of distributed optimization algorithms," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2017, pp. 1206–1212.

[13] I. L. Donato Ridgley, R. A. Freeman, and K. M. Lynch, "Private and hot-pluggable distributed averaging," *IEEE Control Systems Letters*, vol. 4, no. 4, pp. 988–993, 2020.

[14] S. S. Kia, B. Van Scoy, J. Cortés, R. A. Freeman, K. M. Lynch, and S. Martínez, "Tutorial on dynamic average consensus: The problem, its applications, and the algorithms," *IEEE Control Systems Magazine*, vol. 39, no. 3, pp. 40–72, Jun. 2019.

[15] C. N. Hadjicostis, N. H. Vaidya, and A. D. Domínguez-García, "Robust distributed average consensus via exchange of running sums," *IEEE Transactions on Automatic Control*, vol. 61, no. 6, pp. 1492–1507, 2016.

[16] C. N. Hadjicostis, A. D. Domínguez-García, and T. Charalambous, *Distributed Averaging and Balancing in Network Systems*, ser. Foundations and Trends (R) in Systems and Control. Now Publishers, 2018, vol. 13.

[17] A. Sundararajan, B. Van Scoy, and L. Lessard, "A canonical form for first-order distributed optimization algorithms," in *2019 American Control Conference (ACC)*, 2019, pp. 4075–4080.

[18] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd, "Convex optimization in Julia," *SC14 Workshop on High Performance Technical Computing in Dynamic Languages*, 2014.

[19] MOSEK ApS, *MOSEK Optimizer API for C 8.0.0.81*, 2017. [Online]. Available: http://docs.mosek.com/8.0/capi/index.html.

[20] P. K. Mogensen and A. N. Riseth, "Optim: A mathematical optimization package for Julia," *Journal of Open Source Software*, vol. 3, no. 24, p. 615, 2018.

[21] M. Garstka, M. Cannon, and P. Goulart, "COSMO: A conic operator splitting method for large convex problems," in *European Control Conference*, Naples, Italy, 2019.