Proximal Methods for Self-Healing and Exact Distributed Convex Optimization

Extended abstract

Randy A. Freeman
Electrical & Computer Engineering
Northwestern University
Evanston, IL USA
freeman@northwestern.edu

***Abstract*—Distributed convex optimization algorithms employ a variety of methods for achieving exact convergence to the global optimal value (modulo numerical precision): some use time-varying dynamics, some use dynamics on each edge rather than on each node of the communication graph, some use double the communication between nodes per optimization step, and some use a specific initialization that enforces the dynamics to evolve on a particular subspace. Each of these methods has its drawbacks. Using time-varying dynamics might require a global clock, and it might cause transients due to disturbances to take longer and longer to die out as time progresses. Using edge-based rather than node-based dynamics might increase the memory and computation costs, and it typically requires the communication graph to be undirected. Using double the communication per optimization step might increase the communication cost, and it might also slow down the convergence to the optimal value. Using a specific initialization to enforce a particular invariant subspace might render the algorithm unable to recover from faults or disturbances that perturb its dynamics from this subspace, resulting in convergence to the incorrect value. In this latter case we say that the algorithm is not *self-healing*.**

In our previous work [1] we considered strongly convex objective functions having Lipschitz continuous gradients, and we introduced a new first-order method for achieving exact convergence to the global optimal value. Our new algorithm has none of the above drawbacks, and in particular it is a self-healing algorithm. But, it does possess a peculiar internal instability: each node has states that grows linearly in time even as its output converges to the optimal value. In the present work, we consider general non-smooth and extended-real-valued convex objective functions (which can thus incorporate hard convex constraints). We present proximal algorithms that again employ our new, internally unstable method for achieving exact convergence.

***Index Terms*—distributed optimization, convex optimization, proximal methods**

I. Background

Several well-known first-order methods for distributed optimization, such as distributed (sub)gradient descent (DGD) [2] and diffusion [3], converge to a neighborhood of the optimal value when the step size is constant. As the choice for the step size decreases, typically so do both the size of the neighborhood and the convergence rate, thus creating a trade-off between accuracy and speed. While such a trade-off may be undesirable, both DGD and diffusion are *self-healing:* if a temporary disturbance or fault occurs in the distributed computation, the method will automatically resume convergence to its original stationary point without the need for any restarting or re-initialization.

To obtain convergence to the exact value in methods like DGD and diffusion, one can vary the step size with time, making it smaller and smaller as time progresses. In doing so, however, one creates a time-varying response to any disturbances or faults: it takes longer and longer to recover from disturbances as time progresses. Hence such time-varying methods might not be appropriate for applications in which disturbances and faults are common (such as swarm robotics).

By introducing additional states into the optimization dynamics, one can obtain methods that converge to the exact solution under constant step sizes, thus avoiding any accuracy/speed trade-offs. Such methods include EXTRA [4], DIGing [5], [6], NIDS [7], Exact Diffusion [8], SVL [9], DOGT [10], and Push-Pull [11] (and see [9] for other examples). These methods all evolve on a state space that is a foliation of invariant manifolds for the method dynamics, only one of which contains the optimal solution. Thus precise initialization on the correct manifold is a requirement for convergence to the exact solution. In other words, these exact methods are not self-healing.

The methods DGD, diffusion, SVL, NIDS, EXTRA, and Exact Diffusion all require each node to broadcast to its neighbors at each iteration a vector the size of the number of shared decision variables. Because other methods require more communication per iteration, we will say that these six methods are *communication-efficient.* There do exist methods for distributed optimization that are exact and self-healing with time-invariant dynamics (i.e., constant step sizes), but they are not communication-efficient. For example, the self-healing methods in [12] (which is also [13, Algorithm 1]) and [14] require twice as much communication per iteration as do the communication-efficient methods. Also, the edge-based methods in [15], [16] are exact and self-healing with time-invariant dynamics, but they do not work on directed communication graphs and they generate a total amount of network communication per iteration that grows linearly with the number of edges rather than with the number of nodes (and thus are generally not communication-efficient either).

This work was supported in part by a grant from the National Science Foundation (CMMI-2024774).

To the best of our knowledge, our gradient method in [1] is the first distributed optimization method that is exact, self-healing, and communication-efficient, has time-invariant dynamics, and can work on directed communication graphs. However, this method is restricted to smooth convex objective functions having Lipschitz-continuous gradients. In this extended abstract we present proximal versions of our method in [1] that we can apply to general nonsmooth and extended real-valued convex objective functions.

II. The problem

We suppose each node i in a network of n nodes has a local objective function $f_i : \mathbb{R}^c \to \mathbb{R} \cup \{\infty\}$, and we consider the optimization problem

$$\underset{\theta \in \mathbb{R}^c}{\text{minimize}} \quad \sum_{i=1}^{n} f_i(\theta)\,. \tag{1}$$

The nodes wish to solve this problem cooperatively in a distributed manner, i.e., without having to share the local data that comprise their local objective functions f_i. We assume the following:

- **(A1)** each f_i is convex and lower semicontinuous
- **(A2)** $\bigcap_{i=1}^n \operatorname{ri}\operatorname{dom}(f_i) \neq \varnothing$, and
- **(A3)** there is a unique solution $\theta^* \in \mathbb{R}^c$ to problem (1).

In **(A2)**, the set $\operatorname{dom}(f_i) = \{\theta \in \mathbb{R}^c : f_i(\theta) < \infty\}$ represents the effective domain of f_i and ri denotes the relative interior (i.e., the interior relative to the affine hull). Hence **(A2)** is an assumption that the local objective functions have a common "strictly feasible" point.

We model the communication among the nodes in the network as a simple digraph in which a directed edge $i \to j$ from node i to node j exists if and only if node i can send information to node j. Thus the edge direction is the same as the communication direction. Each edge $i \to j$ has an associated weight $\mathrm{w}_{ij} \in \mathbb{R}$. We extend the set of weights to non-edges by setting $\mathrm{w}_{ij} = 0$ whenever $i \to j$ is not an edge of the digraph. The weighted in- and out-degrees d_i^{in} and d_i^{out} for node i are

$$d_i^{\text{in}} = \sum_{j=1}^{n} \mathrm{w}_{ji}\,, \qquad d_i^{\text{out}} = \sum_{j=1}^{n} \mathrm{w}_{ij}\,. \tag{2}$$

The corresponding weighted in- and out-Laplacian matrices for the digraph are

$$L_{\text{in}} = \operatorname{diag}\{d_1^{\text{in}}, \ldots, d_n^{\text{in}}\} - [\mathrm{w}_{ij}] \tag{3}$$
$$L_{\text{out}} = \operatorname{diag}\{d_1^{\text{out}}, \ldots, d_n^{\text{out}}\} - [\mathrm{w}_{ij}]\,, \tag{4}$$

where $[\mathrm{w}_{ij}]$ denotes the weighted adjacency matrix, namely, the $n \times n$ matrix whose entry in row i and column j is w_{ij}. Weights w_{ii} on self-loops cancel out in these Laplacians, so such weights are irrelevant. By construction we have $L_{\text{in}}^{\mathsf{T}} \mathbb{1}_n = L_{\text{out}} \mathbb{1}_n = 0$. We say the weights are *balanced* when $L_{\text{in}} = L_{\text{out}}$, namely, when $d_i^{\text{in}} = d_i^{\text{out}}$ for each node i.

Now suppose each node i has access to a local vector $v_i \in \mathbb{R}^c$, and define the global vector $v = (v_1, \ldots, v_n) \in \mathbb{R}^{cn}$. If edge weights are assigned at the receiving nodes, then each node i can calculate the ith vector component $y_i \in \mathbb{R}^c$ of $y = (L_{\text{in}}^{\mathsf{T}} \otimes I_c)v$ by receiving the vectors v_j from each of its in-neighbors j. In other words, multiplication by the transposed in-Laplacian $L_{\text{in}}^{\mathsf{T}} \otimes I_c$ is a distributed operation. Likewise, if edge weights are assigned at the transmitting nodes, then each node i can calculate the ith component $y_i \in \mathbb{R}^c$ of $y = (L_{\text{out}}^{\mathsf{T}} \otimes I_c)v$ by receiving the weighted vectors $\mathrm{w}_{ji} v_j$ from each of its in-neighbors j. It is easier to use L_{in} rather than L_{out} because nodes naturally know their in-degrees through receiving information from their in-neighbors, whereas discovering their out-degrees may involve extra communication.

We will make use of the orthogonal projection matrix

$$J = I_n - \tfrac{1}{n} \mathbb{1}_n \mathbb{1}_n^{\mathsf{T}}\,, \tag{5}$$

where $\mathbb{1}_n \in \mathbb{R}^n$ represents the vector of n ones. Note that J is also the weighted Laplacian matrix for a complete graph having n nodes and all edge weights equal to $1/n$. We make the following assumptions on the communication digraph:

- **(A4)** the weights are balanced (and we set $L = L_{\text{in}} = L_{\text{out}}$),
- **(A5)** there exists $\sigma \in [0, 1)$ such that $\|J - L\|_2 \leqslant \sigma$.

In practice we achieve **(A4)** by running a weight balancer in parallel with our optimization algorithms so that **(A4)** holds only in the limit, not at every time step. Because our algorithms are self-healing, such parallel weight balancing will not introduce any error in steady state. One can find various methods for balancing weights in [17]–[22]. If the weights are positive and the digraph is strongly connected, then we can always satisfy **(A5)** by scaling all weights by a sufficiently small positive constant. Here σ represents a bound on the distance between L and the Laplacian matrix J for a complete graph, and as such is acts as a measure of the connectivity of the digraph.

We next reformulate the problem by giving each node i a local copy x_i of the common decision variable θ, collecting $x = (x_1, \ldots, x_n) \in \mathbb{R}^{cn}$, and defining $F : \mathbb{R}^{cn} \to \mathbb{R} \cup \{\infty\}$ as

$$F(x) = \sum_{i=1}^{n} f_i(x_i)\,. \tag{6}$$

To obtain an optimization problem for F that is equivalent to the original one in (1), we first define two subspaces of $\mathbb{R}^{cn}$ as follows:

$$\mathcal{C} = \operatorname{Null}(J \otimes I_c) = \big\{(\mu_1, \ldots, \mu_n) \in \mathbb{R}^{cn} \;:\; \mu_i = \mu_j \;\forall i, j\big\} \tag{7}$$
$$\mathcal{C}^{\perp} = \operatorname{Col}(J \otimes I_c) = \big\{(\mu_1, \ldots, \mu_n) \in \mathbb{R}^{cn} \;:\; \textstyle\sum_{i=1}^n \mu_i = 0\big\}\,. \tag{8}$$

Here $\mathcal{C}$ represents the *consensus subspace* and has dimension c, whereas $\mathcal{C}^{\perp}$ represents the *disagreement subspace* and has dimension $c(n-1)$. The reformulated problem is

$$\begin{array}{ll} \underset{x \in \mathbb{R}^{cn}}{\text{minimize}} & F(x) \\ \text{subject to} & x \in \mathcal{C}\,, \end{array} \tag{9}$$

which from assumption **(A3)** has a unique solution given by $x^* = \mathbb{1}_n \otimes \theta^*$. It is straightforward to show that

$$x = x^* \quad \Leftrightarrow \quad x \in \mathcal{C} \text{ and } \partial F(x) \cap \mathcal{C}^{\perp} \neq \varnothing\,, \tag{10}$$

where $\partial F(x) = (\partial f_1(x_1), \ldots, \partial f_n(x_n)) \subseteq \mathbb{R}^{cn}$ denotes the convex subdifferential.

III. The algorithms

There are many optimization methods that are based on a splitting of the objective function into the sum of two terms (ADMM being a prime example), and we will employ such a splitting here. To be precise, each node i splits its local objective function f_i as the sum

$$f_i(x_i) = g_i(x_i) + h_i(A_i x_i + b_i)\,, \tag{11}$$

where $g_i : \mathbb{R}^c \to \mathbb{R} \cup \{\infty\}$, $h_i : \mathbb{R}^{m_i} \to \mathbb{R} \cup \{\infty\}$ for some dimension m_i, $A_i \in \mathbb{R}^{m_i \times c}$, and $b_i \in \mathbb{R}^{m_i}$. Note that m_i, g_i, h_i, A_i, and b_i need only be known to the i^{th} node. Also, the splitting in (11) is not unique, and different choices for the splitting will lead to different algorithms.

The presence of the matrix A_i in (11) is significant, as it often allows us to choose a function h_i having a proximal map that is straightforward to evaluate (and that may even be given by a simple closed-form expression). We could always define a new function $h_i^{\text{new}}(x_i) = h_i(A_i x_i + b_i)$ and thus assume $A_i = I$ and $b_i = 0$ without loss of generality; as explained in [23], however, the proximal map for h_i^{new} can be significantly more difficult to evaluate that that for the original h_i, largely eliminating any potential benefit of the splitting. Hence by allowing $A_i \neq I$, we obtain proximal methods more widely applicable than those such as NIDS [7] that require $A_i = I$.

We assume the splitting in (11) satisfies the following:

(A6) $\operatorname{rank}(A_i) = c$,

(A7) g_i and h_i are convex and lower semicontinuous, and

(A8) $\exists\, \theta \in \operatorname{ri}\operatorname{dom}(g_i)$ such that $A_i\theta + b_i \in \operatorname{ri}\operatorname{dom}(h_i)$.

Assumption **(A6)** incurs no loss of generality: indeed, if $\operatorname{rank}(A_i) < c$ then we can simply add rows to A_i to obtain a larger matrix A_i^{new} which has rank c. We likewise add entries to b_i to obtain b_i^{new}, and we define $h_i^{\text{new}}(v, \mu) = h_i(v)$ so that $h_i^{\text{new}}(A_i^{\text{new}} x_i + b_i^{\text{new}}) = h_i(A_i x_i + b_i)$. Assumption **(A8)** is a regularity condition on the splitting in (11) that will allow us to employ rules from subdifferential calculus.

To derive a formula for ∂F, we let $m = m_1 + \cdots + m_n$ and define $G : \mathbb{R}^{cn} \to \mathbb{R} \cup \{\infty\}$ and $H : \mathbb{R}^m \to \mathbb{R} \cup \{\infty\}$ as

$$G(x) = \sum_{i=1}^{n} g_i(x_i)\,, \qquad H(v) = \sum_{i=1}^{n} h_i(v_i)\,, \tag{12}$$

where $v = (v_1, \ldots, v_n) \in \mathbb{R}^m$ with $v_i \in \mathbb{R}^{m_i}$. Thus using (11) we can write F in (6) as

$$F(x) = G(x) + H(Ax + b)\,, \tag{13}$$

where $A \in \mathbb{R}^{m \times cn}$ denotes the block-diagonal matrix $A = \operatorname{blk\,diag}\{A_1, \ldots, A_n\}$ and $b = (b_1, \ldots, b_n) \in \mathbb{R}^m$. It follows from **(A8)** and the rules of subdifferential calculus [24] that

$$\partial f_i(x_i) = \partial g_i(x_i) + A_i^{\mathsf{T}} \partial h_i(A_i x_i + b_i)\,. \tag{14}$$

Because we have both $\partial G(x) = (\partial g_1(x_1), \ldots, \partial g_n(x_n)) \subseteq \mathbb{R}^d$ and $\partial H(Ax+b) = \big(\partial h_1(A_1x_1+b_1), \ldots, \partial h_n(A_nx_n+b_n)\big) \subseteq \mathbb{R}^m$, it follows from (13) and (14) that

$$\partial F(x) = \partial G(x) + A^{\mathsf{T}} \partial H(Ax + b)\,. \tag{15}$$

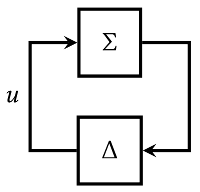

Fig. 1. Feedback form of distributed algorithms

Also, **(A6)** implies that the Moore-Penrose pseudoinverse A^+ is a left inverse of A and is given by

$$A^+ = (A^{\mathsf{T}}A)^{-1}A^{\mathsf{T}} = \operatorname{blk\,diag}\{A_1^+, \ldots, A_n^+\}\,, \tag{16}$$

where $A_i^+ = (A_i^{\mathsf{T}}A_i)^{-1}A_i^{\mathsf{T}}$ for each i.

Our iterative algorithms will involve sequences of vectors, and if v is such a sequence then we will let v^k denote its k^{th} element. Here k is the discrete time variable, which will take values in $\mathbb{N} = \{0, 1, 2, \ldots\}$.

We write our algorithms as the feedback connection of a discrete-time LTI system Σ and a memoryless nonlinear block Δ, as shown in Fig. 1. The system Σ consists of n parallel systems Σ_i (one for each node), and it has the state-space form

$$\begin{bmatrix} w^{k+1} \\ y^k \end{bmatrix} = \left[\begin{array}{c|c} \Sigma_A & \Sigma_B \\ \hline \Sigma_C & \Sigma_D \end{array}\right] \begin{bmatrix} w^k \\ u^k \end{bmatrix} \tag{17}$$

for appropriate matrices $(\Sigma_A, \Sigma_B, \Sigma_C, \Sigma_D)$. Here w^k is the global state at time k. The feedback system will have no algebraic loops, that is, $(\operatorname{id} - \Sigma_D\Delta)^{-1}$ will be a well-defined memoryless operator.

At every time step k, each LTI system Σ_i has two vector states $w_{i1}^k, w_{i2}^k \in \mathbb{R}^{m_i}$, four vector inputs $u_{ij}^k \in \mathbb{R}^{m_i}$, and four vector outputs $y_{ij}^k \in \mathbb{R}^{m_i}$, where $j = 1, \ldots, 4$. We collect the local signals for each node into the vectors $w_j^k, u_j^k, y_j^k \in \mathbb{R}^m$ by defining $w_1^k = (w_{11}^k, \ldots, w_{n1}^k)$ and likewise for the other signals. We further collect the states, the inputs, and the outputs as $w^k = (w_1^k, w_2^k) \in \mathbb{R}^{2m}$, $u^k = (u_1^k, \ldots, u_4^k) \in \mathbb{R}^{4m}$, and $y^k = (y_1^k, \ldots, y_4^k) \in \mathbb{R}^{4m}$. The state-space matrices for the system Σ are

$$\left[\begin{array}{c|c} \Sigma_A & \Sigma_B \\ \hline \Sigma_C & \Sigma_D \end{array}\right] = \left[\begin{array}{cc|cccc} 1 & 0 & b_{11} & b_{12} & 1 & b_{14} \\ 0 & 1 & 0 & 0 & 1 & 0 \\ \hline c_{11} & 1 & 0 & 0 & 0 & 0 \\ c_{21} & 0 & d_{21} & 0 & d_{23} & 0 \\ c_{31} & 0 & d_{31} & d_{32} & 0 & d_{34} \\ 1 & 0 & 0 & 0 & d_{43} & 0 \end{array}\right] \otimes I_m \tag{18}$$

for parameters $c_{ij}, d_{ij} \in \mathbb{R}$. The nonlinear block $u^k = \Delta(y^k)$

is given by the memoryless mappings

$$u_1^k = \Delta_1(y_1^k) = A^{+\mathsf{T}}(L^\mathsf{T} \otimes I_c)A^+ y_1^k \tag{19}$$

$$u_2^k = \Delta_2(y_2^k) = \begin{cases} \mathrm{gd}_{\alpha G}^A(y_2^k) & \text{(algorithm 1)} \\ \mathrm{prox}_{\alpha G}^A(y_2^k) & \text{(algorithm 2)} \end{cases} \tag{20}$$

$$u_3^k = \Delta_3(y_3^k) = \mathrm{prox}_{\beta H}(y_3^k + b) - b \tag{21}$$

$$u_4^k = \Delta_4(y_4^k) = AA^+ y_4^k\,. \tag{22}$$

where α and β represent strictly positive step sizes. Hence Δ is of the diagonal form $\Delta = \mathrm{diag}\{\Delta_1, \dots, \Delta_4\}$. The nonlinear maps in (20)–(21) are given by

$$\mathrm{gd}_{\alpha G}^A(v) = A^{+\mathsf{T}}\left[A^\mathsf{T} v - \alpha \nabla G(A^+ v)\right] \tag{23}$$

$$\mathrm{prox}_{\alpha G}^A(v) = A \cdot \arg\min_z \left[\alpha G(z) + \tfrac{1}{2}\|Az - v\|^2\right] \tag{24}$$

$$\mathrm{prox}_{\beta H}(v) = \arg\min_z \left[\beta H(z) + \tfrac{1}{2}\|z - v\|^2\right] \tag{25}$$

There are two choices for Δ_2 in (20) leading to two different algorithms: the first uses the weighted gradient descent map $\mathrm{gd}_{\alpha G}^A$ whereas the second uses the weighted proximal map $\mathrm{prox}_{\alpha G}^A$. Both algorithms use the standard proximal map $\mathrm{prox}_{\beta H}$ for Δ_3. Note that algorithm 1 uses the gradient of G and thus requires each g_i to be everywhere differentiable. Also, we must take care when using algorithm 2 that our splitting results in each g_i being simple enough (e.g., quadratic) to make the evaluation of $\mathrm{prox}_{\alpha G}^A$ straightforward. The only part of Σ and Δ that involves communication between nodes is Δ_1 (where the Laplacian matrix L appears); all other computations can be performed locally in parallel on each node.

The "readout" signal x_{out}^k that will converge to optimal solution x^* under suitable stability conditions is

$$x_{\text{out}}^k = A^+ u_3^k\,. \tag{26}$$

Our algorithms are not internally stable: the signals w_2^k and y_1^k grow linearly in k in steady state, even as x_{out}^k converges to the optimal solution x^*. This mild internal instability is the price we pay for gaining the self-healing property without losing exactness, time invariance, or communication efficiency.

There are 12 parameters in the matrices Σ_B, Σ_C, and Σ_D, which together with the step sizes α and β represent a total of 14 algorithm parameters. However, we cannot freely choose all 14 parameters. First, our parameter choices must guarantee the well-posedness of the feedback loop in Fig. 1. For this reason we assume that $d_{23}d_{32} = d_{34}d_{43} = 0$, which together with the diagonal form of the nonlinear block Δ ensures that $(\mathrm{id} - \Sigma_D\Delta)^{-1}$ is a well-defined operator on $\mathbb{R}^{4m}$. This operator consists of various compositions of linear maps and the nonlinearities $\mathrm{gd}_{\alpha G}^A$ (or $\mathrm{prox}_{\alpha G}^A$) and $\mathrm{prox}_{\beta H}$. In particular, Δ and $(\mathrm{id} - \Sigma_D\Delta)^{-1}$ are Lipschitz continuous in algorithm 2 (and also in algorithm 1 when ∇G is Lipschitz).

Second, our parameter choices must guarantee that the readout signal x_{out}^k converges to the optimal solution x^* as $k \to \infty$, regardless of the initial state w^0. This guarantee comes in two parts: *stability*, which is the existence of a limit for x_{out}^k, and *correctness*, which is this limit being equal to x^*.

We say that the algorithm parameters are *correct* when they guarantee well-posedness ($d_{23}d_{32} = d_{34}d_{43} = 0$), have strictly positive step sizes α and β, and satisfy a collection of *correctness constraints*. For algorithm 1, these correctness constraints are:

$$c_{31} \neq 0\,, \qquad c_{31} \neq -d_{34} \tag{27}$$

$$\beta \neq \alpha(1 + c_{31}d_{43})\,, \qquad b_{11} \neq \frac{\alpha d_{31}}{\beta - \alpha(1 + c_{31}d_{43})} \tag{28}$$

$$b_{12} = \frac{\alpha d_{32} - \beta}{\beta - \alpha(1 + c_{31}d_{43})}\,, \qquad b_{14} = \frac{\alpha(c_{31} + d_{34})}{\beta - \alpha(1 + c_{31}d_{43})} \tag{29}$$

$$\mathrm{rank}\begin{bmatrix} b_{11} & b_{12} & 1 & b_{14} \\ d_{21} & 0 & d_{23} - c_{21}d_{43} - 1 & c_{21} \end{bmatrix} = 1\,. \tag{30}$$

To satisfy these constraints, we can choose the parameters as follows. First we choose d_{34} and d_{43}, noting that at least one of them must be zero (so together they represent only one independent parameter). Then we choose any α, β, b_{11}, c_{11}, c_{31}, and d_{31} that satisfy the inequality constraints in (27)–(28). This results in seven independent parameter choices, and we must choose the remaining six parameters in one of the following two ways. The first way is to choose $c_{21} = d_{21} = d_{32} = 0$, $d_{23} = 1$, and b_{12} and b_{14} according to (29). The second way, which requires $\alpha \neq \beta$, is to choose $b_{12} = d_{23} = 0$, $d_{32} = \beta/\alpha$, b_{14} according to (29), and

$$c_{21} = \frac{\alpha(c_{31} + d_{34})}{\alpha - \beta}\,, \qquad d_{21} = -b_{11}(1 + c_{21}d_{43})\,. \tag{31}$$

In either case, the resulting set of correct parameters is a seven-dimensional semialgebraic subset of $\mathbb{R}^{14}$.

For algorithm 2, the correctness constraints are:

$$b_{11} = b_{14} = 0\,, \qquad b_{12} = -1 \tag{32}$$

$$\alpha(c_{31} + d_{34}) + \beta c_{21} = 0\,, \qquad \alpha d_{32} + \beta d_{23} = \alpha + \beta \tag{33}$$

$$c_{31} + d_{34} \neq 0\,, \qquad d_{21}(c_{31} + d_{34}) \neq c_{21}d_{31}\,. \tag{34}$$

To satisfy these constraints, we can choose the parameters as follows. First we choose d_{34} and d_{43}, noting that at least one of them must be zero. Then we choose any c_{31} such that $c_{31} + d_{34} \neq 0$. Next we choose any α, β, c_{11}, and d_{21}, and then we solve (33) for c_{21}, d_{23}, and d_{32} (noting that at least one of d_{23} and d_{32} must be zero). Finally. we choose any d_{31} that satisfies the right-hand inequality in (34). So again, the set of correct parameters is a seven-dimensional semialgebraic subset of $\mathbb{R}^{14}$. However, if we choose $d_{34} = 0$ then u_4 becomes disconnected from the feedback loop and the value of d_{43} becomes irrelevant; hence in this case the effective parameter space has dimension six.

The stability analysis is beyond the scope of this extended abstract, and thus we provide only an overview. Given a desired linear convergence rate, a set of correct algorithm parameters, the connectivity parameter σ from **(A5)**, bounds on the singular values of the matrices A_i, and other bounds related to the g_i parts of the split local objective functions, we can determine whether x_{out}^k converges to the optimal solution x^* at the desired linear rate by solving an LMI feasibility problem. The size of this LMI is independent of n (the number of nodes), m (the number of rows in A), and c (the number of decision variables). We have verified that for a wide variety of scenarios, there exist algorithm parameters such that this LMI is feasible for some linear convergence rate.

If the splitting in (11) is such that $A_i = I_c$ and $h_i \equiv 0$ for each i, then algorithm 1 reduces to [1, Algorithm 1] under appropriate parameter choices. Let δ, η, and ζ be the algorithm parameters from [1] (together with the step size α). Consider the following parameter values: $c_{21} = d_{21} = d_{32} = d_{34} = 0$, $b_{12} = b_{14} = c_{31} = -1$, $d_{23} = d_{43} = 1$, and

$$\beta = \alpha\,, \qquad b_{11} = \eta(\zeta - 1) \tag{35}$$

$$c_{11} = -\delta/\eta\,, \qquad d_{31} = -\eta\,. \tag{36}$$

If $\eta\zeta \neq 0$ (which is needed in [1] for the convergence proof), then these choices are correct for algorithm 1. Now $A = I_{cn}$ and $H \equiv 0$ imply $\Delta_3 = \Delta_4 = \mathrm{id}$, which means $u_3^k = -w_1^k - \eta u_1^k$ and $u_4^k = -\eta u_1^k$ for all k. We define the signals (v^k, x^k, y^k) from [1] as $v^k = \eta u_1^k$, $y^k = \eta y_1^k = -\delta w_1^k + \eta w_2^k$, and $x^k = y_2^k = u_3^k = -w_1^k - v^k$. These result in the state update equations

$$\begin{aligned} w_1^{k+1} &= w_1^k + b_{11} u_1^k - u_2^k + u_3^k - u_4^k \\ &= w_1^k + \alpha \nabla F(x^k) + \zeta v^k \end{aligned} \tag{37}$$

$$w_2^{k+1} = -w_1^k + w_2^k - \eta u_1^k = -w_1^k + w_2^k - v^k\,. \tag{38}$$

Now $v^k = (L^\mathsf{T} \otimes I_c) y^k$ because $A = I_{cn}$, so we recover [1, Algorithm 1] if we scale the state variable w_1^k by -1, namely, if we replace w_1^k with $-w_1^k$. An example of the parameters from [1] are $\delta = \eta = 0.5$ and $\zeta = 1$, which are inspired by the NIDS [7] and Exact Diffusion [8] methods. These choices result in the following list of parameters for our algorithm 1: $b_{11} = c_{21} = d_{21} = d_{32} = d_{34} = 0$, $b_{12} = b_{14} = c_{11} = c_{31} = -1$, $d_{23} = d_{43} = 1$, and $d_{31} = -0.5$, with step size $\alpha = \beta$.

IV. Example: distributed support-vector machine

Suppose each node i in the network has access to s_i data samples $\{d_{i1}, \dots, d_{is_i}\}$ belonging to a data set D, with each sample having a binary label $y_{ij} \in \{-1, 1\}$, and suppose the nodes would like to use their data to build a binary classifier. To accomplish this, the nodes wish to solve

$$\underset{\theta \in \mathbb{R}^c}{\text{minimize}} \;\; \frac{1}{s} \sum_{i=1}^{n} \sum_{j=1}^{s_i} \ell\big(y_{ij} \phi(d_{ij})^\mathsf{T} \theta\big) + \mu_1 \|\theta\|_1 + \tfrac{1}{2}\mu_2 \|\theta\|^2\,, \tag{39}$$

where $\phi : D \to \mathbb{R}^c$ is a feature map, ℓ is the hinge loss function $\ell(r) = \max\{0, 1 - r\}$, $\theta \in \mathbb{R}^c$ is the model parameter vector, $s = s_1 + \cdots + s_n$ is the total number of samples, and $\mu_1, \mu_2 \geqslant 0$ are the parameters for "elastic net" regularization. Once the nodes have a solution θ^* to (39), then the binary label y for a new data vector d_{new} is simply $y = \operatorname{sgn}(\phi(d_{\text{new}})^\mathsf{T} \theta^*)$.

We multiply the objective in (39) by s so that the local objective functions are

$$f_i(x_i) = \sum_{j=1}^{s_i} \ell\big(y_{ij} \phi(d_{ij})^\mathsf{T} x_i\big) + s_i \mu_1 \|x_i\|_1 + \tfrac{1}{2} s_i \mu_2 \|x_i\|^2 \tag{40}$$

We split (40) as in (11) as follows:

$$g_i(x_i) = \tfrac{1}{2} \kappa \mu_2 \|x_i\|^2 \tag{41}$$

$$h_i(v_i) = s_i \mu_1 \|v_{i0}\|_1 + \tfrac{1}{2}(s_i - \kappa)\mu_2 \|v_{i0}\|^2 + \sum_{j=1}^{s_i} \ell(v_{ij})\,, \tag{42}$$

where $v_i = (v_{i0}, v_{i1}, \dots, v_{is_i}) \in \mathbb{R}^{m_i}$, $m_i = c + s_i$, $v_{i0} \in \mathbb{R}^c$, $v_{ij} \in \mathbb{R}$ for $j = 1, \dots, s_i$, and κ is a parameter chosen such that $0 < \kappa \leqslant \min\{s_1, \dots, s_n\}$. If we define A_i as

$$A_i = \begin{bmatrix} I_c \\ \Xi_i^\mathsf{T} \end{bmatrix} \qquad \in \mathbb{R}^{m_i \times c} \tag{43}$$

$$\Xi_i = \begin{bmatrix} y_{i1}\phi(d_{i1}) & \dots & y_{is_i}\phi(d_{is_i}) \end{bmatrix} \qquad \in \mathbb{R}^{c \times s_i} \tag{44}$$

then by construction this gives $f_i(x_i) = g_i(x_i) + h_i(A_i x_i)$. In this case $\mathrm{gd}_{\alpha g_i}^{A_i}$ and $\mathrm{prox}_{\alpha g_i}^{A_i}$ are straightforward to evaluate (and are in fact linear maps):

$$\mathrm{gd}_{\alpha g_i}^{A_i}(y) = \big[A_i - \alpha\kappa\mu_2 A_i^{+\mathsf{T}}\big] A_i^+ y \tag{45}$$

$$\mathrm{prox}_{\alpha g_i}^{A_i}(y) = A_i \big(\alpha\kappa\mu_2 I_c + A_i^\mathsf{T} A_i\big)^{-1} A_i^\mathsf{T} y\,. \tag{46}$$

Moreover, $\mathrm{prox}_{\beta h_i}$ is easy to evaluate because

$$\mathrm{prox}_{\beta\ell}(r) = \begin{cases} r & \text{when } r > 1 \\ 1 & \text{when } 1 - \beta \leqslant r \leqslant 1 \\ r + \beta & \text{when } r < 1 - \beta\,. \end{cases} \tag{47}$$

Thus both algorithms 1 and 2 apply to this splitting. We have yet to investigate which of these two algorithms might provide better performance for this problem.

V. Example: distributed matrix completion

Let $\mathbb{M}$ be a real matrix space (i.e., the set of all matrices of a fixed size having real entries). A noisy measurement of a matrix $M \in \mathbb{M}$ is a pair $(P, \mathcal{I})$, where $P : \mathbb{M} \to \mathbb{R}$ is a linear mapping and $\mathcal{I} \subseteq \mathbb{R}$ is a nonempty closed interval such that $P(M) \in \mathcal{I}$. In the standard formulation, P simply selects a particular entry of the matrix. Now suppose each node i has access to a collection $\{(P_{ij}, \mathcal{I}_{ij})\}_{j=1}^{s_i}$ of s_i noisy measurements of some unknown matrix $M \in \mathbb{M}$. Our goal is to find a matrix $X \in \mathbb{M}$ of minimum rank that is consistent with all of the measurements, namely, that satisfies $P_{ij}(X) \in \mathcal{I}_{ij}$ for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant s_i$. We obtain a convex relaxation of this problem by replacing $\operatorname{rank}(X)$ with the nuclear norm $\|X\|_*$ and adding a quadratic regularization term:

$$\begin{aligned} \underset{X \in \mathbb{M}}{\text{minimize}} \quad & \|X\|_* + \tfrac{1}{2}\mu \|X - M_0\|_F^2 \\ \text{subject to} \quad & P_{ij}(X) \in \mathcal{I}_{ij} \quad \forall i \in \{1, \dots, n\}, \\ & \qquad\qquad\qquad\;\; \forall j \in \{1, \dots, s_i\}\,, \end{aligned} \tag{48}$$

where $\|\cdot\|_F$ denotes the Frobenius norm and M_0 is some *a priori* estimate of the unknown matrix M. Recall that the characteristic function char_S of a set S is

$$\mathrm{char}_S(r) = \begin{cases} 0 & \text{when } r \in S \\ \infty & \text{otherwise} \end{cases} \tag{49}$$

We will encode the hard interval constraints $P_{ij}(X) \in \mathcal{I}_{ij}$ in (48) by adding the terms $\mathrm{char}_{\mathcal{I}_{ij}}\big(P_{ij}(X)\big)$ to the objective function. We let the decision variable $\theta = \operatorname{vec}(X) \in \mathbb{R}^c$ be a vectorization of X, where $c = \dim(\mathbb{M})$. If we scale the objective in (48) by n, then the problem is of the form (1) with

$$\begin{aligned} f_i(\theta) = \|X(\theta)\|_* + \tfrac{1}{2}\mu \|X(\theta) - M_0\|_F^2 & \\ + \sum_{j=1}^{s_i} \mathrm{char}_{\mathcal{I}_{ij}}\big(P_{ij}(X(\theta))\big)\,, & \end{aligned} \tag{50}$$

where $X(\theta)$ denotes the reassembly of θ into the matrix X. Because each P_{ij} is linear, there exist vectors $p_{ij} \in \mathbb{R}^c$ such that $P_{ij}(X(\theta)) = p_{ij}^\top \theta$. If we define the matrix A_i as

$$A_i = \begin{bmatrix} I_c \\ p_{i1}^\top \\ \vdots \\ p_{is_i}^\top \end{bmatrix}, \tag{51}$$

then $f_i(x_i) = g_i(x_i) + h_i(A_i x_i)$ with

$$g_i(x_i) = \tfrac{1}{2}\mu \|X(x_i) - M_0\|_F^2 \tag{52}$$

$$h_i(v_i) = \|X(v_{i0})\|_* + \sum_{j=1}^{s_i} \operatorname{char}_{\mathcal{I}_{ij}}(v_{ij}), \tag{53}$$

where $v_i = (v_{i0}, v_{i1}, \ldots, v_{is_i}) \in \mathbb{R}^{m_i}$, $m_i = c + s_i$, $v_{i0} \in \mathbb{R}^c$, and $v_{ij} \in \mathbb{R}$ for $j = 1, \ldots, s_i$. Again $\operatorname{gd}_{\alpha g_i}^{A_i}$ and $\operatorname{prox}_{\alpha g_i}^{A_i}$ are straightforward to evaluate:

$$\operatorname{gd}_{\alpha g_i}^{A_i}(y) = \left[A_i - \alpha\mu A_i^{+\top}\right] A_i^{+} y + \alpha\mu A_i^{+\top} \cdot \operatorname{vec}(M_0) \tag{54}$$

$$\operatorname{prox}_{\alpha g_i}^{A_i}(y) = A_i\left(\alpha\mu I_c + A_i^\top A_i\right)^{-1}\left[A_i^\top y + \alpha\mu \cdot \operatorname{vec}(M_0)\right]. \tag{55}$$

Computing the proximal map for βh_i involves singular value thresholding, i.e., computing all singular values of a matrix in $\mathbb{M}$ that are larger than β. Again both algorithms 1 and 2 apply to this splitting, and we have yet to investigate which of these two algorithms might provide better performance for this problem.

VI. Simulations of example problems

We tested our algorithm 2 on particular instances of the support-vector machine and matrix completion problems. The communication digraph was the same for each problem: a strongly connected digraph with $n = 16$ nodes generated at random using the Erdős-Rényi method with an 0.3 edge probability. For the support-vector machine, we used $s = 118$ labeled data points in $\mathbb{R}^2$ from the COSMO.jl documentation [25] (distributed at random among the nodes), together with a 6th-order polynomial feature map $\phi : \mathbb{R}^2 \to \mathbb{R}^{28}$ and regularization parameters $\mu_1 = 0.01$ and $\mu_2 = 0.1$. For the matrix completion problem, the unknown matrix M was a randomly generated $16{\times}100$ matrix having rank 5 and entries in the range 0 to 10. The ith node had between 1 and 10 measurements of entries from the ith row of M, with a total of $s = 80$ measurements of M distributed among the nodes. The intervals $\mathcal{I}_{ij}$ for the measurements were random subintervals of $[0, 10]$. The *a priori* estimate M_0 of M was a matrix of all 5's, and the regularization parameter was $\mu = 0.01$.

We chose the parameters for algorithm 2 as shown in Tables I and II. To demonstrate the self-healing property of the algorithm, we chose the initial state vector w^0 at random. We see from Fig. 2 that the error indeed converges to zero at a linear rate for each of the two problems.

VII. Future work

There is yet much to do, besides providing the stability analysis not present in this extended abstract. First, we must find a systematic way to choose the algorithm parameters, not only to guarantee convergence but also (if possible) to optimize the convergence rate. Second, we need to include additional edge states as in [1] to make the algorithms robust to intermittent communication failures (e.g., packet drops). Third, we need to investigate how robust our algorithms are to errors or noise in the calculations of the gradient descent and proximal maps. Finally, as these are first-order methods, we need to determine whether we can accelerate their convergence by including additional dynamics states (e.g., momentum terms).

Parameter	Value	Parameter	Value
α	1.0	c_{31}	−1.0
β	2.0	d_{21}	−0.1
b_{11}	0.0	d_{23}	1.5
b_{12}	−1.0	d_{31}	−0.6
b_{14}	0.0	d_{32}	0.0
c_{11}	−6.0	d_{34}	0.0
c_{21}	0.5	d_{43}	0.0

TABLE I
Parameter choices for the support-vector machine.

Parameter	Value	Parameter	Value
α	1.6	c_{31}	−1.0
β	2.0	d_{21}	−0.1
b_{11}	0.0	d_{23}	1.8
b_{12}	−1.0	d_{31}	−0.6
b_{14}	0.0	d_{32}	0.0
c_{11}	−1.0	d_{34}	0.0
c_{21}	0.8	d_{43}	0.0

TABLE II
Parameter choices for the matrix completion problem.

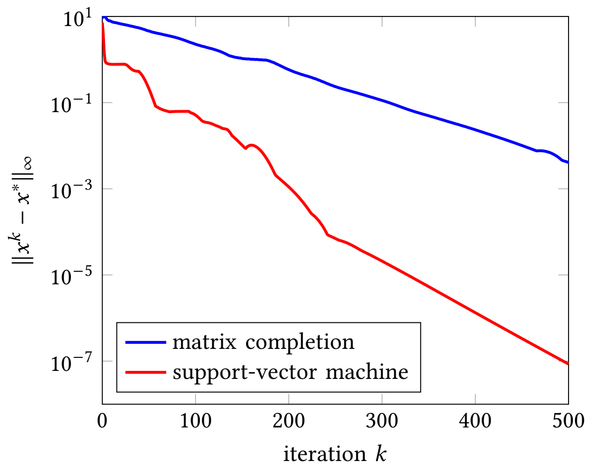

Fig. 2. An illustration of the linear convergence of algorithm 2 applied to particular instances of the example problems. The algorithm states were initialized at random.

References

[1] I. L. Donato Ridgley, R. A. Freeman, and K. M. Lynch, "Self-healing first-order distributed optimization," in *Proceedings of the 60th IEEE Conference on Decision and Control*, (Austin, Texas), Dec. 2021, pp. 3850–3856.

[2] A. Nedić and A. Ozdaglar, "Distributed subgradient

methods for multi-agent optimization," *IEEE Trans. Automat. Contr.*, vol. 54, no. 1, pp. 48–61, 2009.

[3] J. Chen and A. H. Sayed, "Diffusion adaptation strategies for distributed optimization and learning over networks," *IEEE Trans. Signal Process.*, vol. 60, no. 8, pp. 4289–4305, 2012.

[4] W. Shi, Q. Ling, G. Wu, and W. Yin, "EXTRA: An exact first-order algorithm for decentralized consensus optimization," *SIAM J. Optim.*, vol. 25, no. 2, pp. 944–966, 2015.

[5] A. Nedić, A. Olshevsky, and W. Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM J. Optim.*, vol. 27, no. 4, pp. 2597–2633, 2017.

[6] G. Qu and N. Li, "Harnessing smoothness to accelerate distributed optimization," *IEEE Trans. Control Netw. Syst.*, vol. 5, no. 3, pp. 1245–1260, 2018.

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

[8] K. Yuan, B. Ying, X. Zhao, and A. H. Sayed, "Exact diffusion for distributed optimization and learning—Part I: Algorithm development," *IEEE Trans. Signal Process.*, vol. 67, no. 3, pp. 708–723, 2019.

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

[10] A. Daneshmand, G. Scutari, and V. Kungurtsev, "Second-order guarantees of distributed gradient algorithms," *SIAM J. Optim.*, vol. 30, no. 4, pp. 3029–3068, 2020.

[11] S. Pu, W. Shi, J. Xu, and A. Nedić, "Push-pull gradient methods for distributed optimization in networks," *IEEE Trans. Automat. Contr.*, vol. 66, no. 1, pp. 1–16, 2021.

[12] D. Meng, M. Fazel, and M. Mesbahi, "Proximal alternating direction method of multipliers for distributed optimization on weighted graphs," in *Proceedings of the 54th IEEE Conference on Decision and Control*, (Osaka, Japan), 2015, pp. 1396–1401.

[13] Y. Yu, B. Açikmeşe, and M. Mesbahi, "Bregman parallel direction method of multipliers for distributed optimization via mirror averaging," *IEEE Contr. Syst. Lett.*, vol. 2, no. 2, pp. 302–306, 2018.

[14] A. Makhdoumi and A. Ozdaglar, "Convergence rate of distributed ADMM over networks," *IEEE Trans. Automat. Contr.*, vol. 62, no. 10, pp. 5082–5095, 2017.

[15] E. Wei and A. Ozdaglar, "Distributed alternating direction method of multipliers," in *Proceedings of the 51st IEEE Conference on Decision and Control*, (Maui, Hawaii), 2012, pp. 5445–5450.

[16] N. Bastianello, M. Todescato, R. Carli, and L. Schenato, "Distributed optimization over lossy networks via relaxed Peaceman-Rachford splitting: A robust ADMM approach," in *Proceedings of the 2018 European Control Conference*, (Limassol, Cyprus), 2018, pp. 477–482.

[17] B. Gharesifard and J. Cortés, "Distributed strategies for generating weight-balanced and doubly stochastic digraphs," *Eur. J. Control*, vol. 18, no. 6, pp. 539–557, 2012.

[18] Z. Qu, C. Li, and F. Lewis, "Cooperative control with distributed gain adaptation and connectivity estimation for directed networks," *Int. J. Robust Nonlinear Contr.*, vol. 24, no. 3, pp. 450–476, 2014.

[19] A. I. Rikos, T. Charalambous, and C. N. Hadjicostis, "Distributed weight balancing over digraphs," *IEEE Trans. Control Netw. Syst.*, vol. 1, no. 2, pp. 1–12, 2014.

[20] T. Charalambous, M. G. Rabbat, M. Johansson, and C. N. Hadjicostis, "Distributed finite-time computation of digraph parameters: Left-eigenvector, out-degree and spectrum," *IEEE Trans. Control Netw. Syst.*, vol. 3, no. 2, pp. 137–148, 2016.

[21] A. Gusrialdi and Z. Qu, "Distributed estimation of all the eigenvalues and eigenvectors of matrices associated with strongly connected digraphs," *IEEE Contr. Syst. Lett.*, vol. 1, no. 2, pp. 328–333, 2017.

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

[23] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[24] B. S. Mordukhovich and N. M. Nam, "Geometric approach to convex subdifferential calculus," *Optimization*, vol. 66, no. 6, pp. 839–873, 2017.

[25] M. Garstka, M. Cannon, and P. Goulart, "COSMO: A conic operator splitting method for convex conic problems," *J. Optimiz. Theory Appl.*, vol. 190, no. 3, pp. 779–810, 2021.