A Push-Pull Gradient Method for Distributed Optimization in Networks

Shi Pu, Wei Shi, Jinming Xu, and Angelia Nedić

Abstract— **In this paper, we focus on solving a distributed convex optimization problem in a network, where each agent has its own convex cost function and the goal is to minimize the sum of the agents' cost functions while obeying the network connectivity structure. In order to minimize the sum of the cost functions, we consider a new distributed gradient-based method where each node maintains two estimates, namely, an estimate of the optimal decision variable and an estimate of the gradient for the average of the agents' objective functions. From the viewpoint of an agent, the information about the decision variable is pushed to the neighbors, while the information about the gradients is pulled from the neighbors (hence giving the name "push-pull gradient method"). The method unifies the algorithms with different types of distributed architecture, including decentralized (peer-to-peer), centralized (master-slave), and semi-centralized (leader-follower) architecture. We show that the algorithm converges linearly for strongly convex and smooth objective functions over a directed static network. In our numerical test, the algorithm performs well even for time-varying directed networks.**

I. Introduction

In this paper, we consider a system involving n agents whose goal is to collaboratively solve the following problem:

$$\min_{x\in\mathbb{R}^p} \ f(x) := \sum_{i=1}^{n} f_i(x), \tag{1}$$

where x is the global decision variable and each function $f_i : \mathbb{R}^p \to \mathbb{R}$ is convex and known by agent i only. The agents are embedded in a communication network, and their goal is to obtain an optimal and consensual solution through local neighbor communications and information exchange. This local exchange is desirable in situations where privacy needs to be preserved, or the exchange of a large amount of data is prohibitively expensive due to limited communication resources.

To solve problem (1) in a networked system of n agents, many algorithms have been proposed under various assumptions on the objective functions and the underlying network [1]–[26]. Centralized algorithms are discussed in [1], where extensive applications in learning can be found. Parallel, coordinated, and asynchronous algorithms are discussed in [2] and the references therein.

Our emphasis in the literature review is on the decentralized optimization since our approach builds on a new understanding of the decentralized consensus-based methods for directed communication networks. Most references, including [3]–[14], often restrict the underlying network connectivity structure, or more commonly require doubly stochastic mixing matrices. The work in [3] has been the first to demonstrate the linear convergence of an ADMM-based decentralized optimization scheme. Reference [4] uses a gradient difference structure in the algorithm to provide the first-order decentralized optimization algorithm which is capable of achieving the typical convergence rates of a centralized gradient method, while references [5], [6] deal with the second-order decentralized methods. By using Nesterov's acceleration, reference [7] has obtained a method whose convergence time scales linearly in the number of agents n, which is the best scaling with n currently known. More recently, for a class of so-termed dual friendly functions, papers [8], [9] have obtained an optimal decentralized consensus optimization algorithm whose dependency on the condition number[1] κ of the system's objective function $\sum_{i=1}^{n} f(x_i)$ achieves the best known scaling in the order of $O(\sqrt{\kappa})$. Work in [13], [14] investigates proximal-gradient methods which can tackle (1) with proximal friendly component functions. Paper [19] extends the work in [3] to handle asynchrony and delays. References [20], [21] consider a stochastic variant of problem (1) in asynchronous networks. A tracking technique has been recently employed to develop decentralized algorithms for tracking the average of the Hessian/gradient in second-order methods [6], allowing uncoordinated step-size [10], [11], handling non-convexity [12], and achieving linear convergence over time-varying graphs [22].

For directed graphs, to eliminate the need of constructing a doubly stochastic matrix in reaching consensus[2], reference [27] proposes the push-sum protocol. Reference [28] has been the first to propose a push-sum based distributed optimization algorithm for directed graphs. Then, based on the push-sum technique again, a decentralized subgradient method for time-varying directed graphs has been proposed and analyzed in [15]. Aiming to improve convergence for a smooth objective function and a fixed directed graph, work in [16], [17] modifies the algorithm from [4] with the push-sum technique, thus providing a new algorithm which converges linearly for a strongly convex objective function on a static graph. However, the algorithm has some stability

*This work was supported in parts by the NSF grant CCF-1717391 and by the ONR grant no. N00014-12-1-0998.

Shi Pu, Wei Shi, Jinming Xu and Angelia Nedić are with the School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287, USA. (emails: shipu3@asu.edu, wshi36@asu.edu, jinming.xu@asu.edu, Angelia.Nedich@asu.edu)

[1]The condition number of a smooth and strongly convex function is the ratio of its gradient Lipschitz constant and its strong convexity constant.

[2]Constructing a doubly stochastic matrix over a directed graph needs weight balancing which requires an independent iterative procedure across the network; consensus is a basic element in decentralized optimization.

issues, which have been resolved in [22] in a more general setting of time-varying directed graphs.

In this paper, we introduce a modified gradient-tracking algorithm for decentralized (consensus-based) optimization in directed graphs. Unlike the push-sum protocol, our algorithm uses a row stochastic matrix for the mixing of the decision variables, while it employs a column stochastic matrix for tracking the average gradients. Although motivated by a fully decentralized scheme, we will show that our algorithm can work both in fully decentralized networks and in two-tier networks. The contributions of this paper include the design of a new decentralized algorithm[3] and the establishment of its linear convergence for a static directed graph. We numerically evaluate our proposed algorithm for both static and time-varying graphs, and find that the algorithm is competitive as compared to the linearly convergent algorithm developed in [22].

The structure of this paper is as follows. We first provide notation and state basic assumptions in Subsection I-A. Then, we introduce our algorithm in Section II along with the intuition of its design and some examples explaining how it relates to (semi-)centralized and decentralized optimization. We establish the linear convergence of our algorithm in Section III, while in Section IV we conduct numerical test to verify our theoretical claims. Concluding remarks are given in Section V.

A. Notation and Assumptions

Throughout the paper, vectors default to columns if not otherwise specified. Let each agent $i \in \{1, 2, \ldots, n\}$ hold a local copy $x_i \in \mathbb{R}^p$ of the decision variable and an auxiliary variable $y_i \in \mathbb{R}^p$ tracking the average gradients, where their values at iteration k are denoted by $x_{i,k}$ and $y_{i,k}$, respectively. Let

$$\mathbf{x} := [x_1, x_2, \ldots, x_n]^\intercal \in \mathbb{R}^{n\times p},$$
$$\mathbf{y} := [y_1, y_2, \ldots, y_n]^\intercal \in \mathbb{R}^{n\times p}.$$

Define $F(\mathbf{x})$ to be an aggregate objective function of the local variables, i.e., $F(\mathbf{x}) := \sum_{i=1}^n f_i(x_i)$, and write

$$\nabla F(\mathbf{x}) := [\nabla f_1(x_1), \nabla f_2(x_2), \ldots, \nabla f_n(x_n)]^\intercal \in \mathbb{R}^{n\times p}.$$

Definition 1: Given an arbitrary vector norm $\|\cdot\|$ on $\mathbb{R}^n$, for any $\mathbf{x} \in \mathbb{R}^{n\times p}$, we define

$$\|\mathbf{x}\| := \left\| \left[\|\mathbf{x}^{(1)}\|, \|\mathbf{x}^{(2)}\|, \ldots, \|\mathbf{x}^{(p)}\| \right] \right\|_2,$$

where $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(p)} \in \mathbb{R}^n$ are columns of $\mathbf{x}$, and $\|\cdot\|_2$ represents the 2-norm.

A directed graph is a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices (nodes) and the edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ consists of ordered pairs of vertices. A directed tree is a directed graph where every vertex, except for the root, has only one parent. A spanning tree of a directed graph is a directed tree that connects the root to all other vertices in the graph (see [30]).

[3]While completing the paper, we became aware of a recent work discussing an algorithm that is similar to ours [29]. We have independently arrived to our method and results.

Given a nonnegative matrix $M \in \mathbb{R}^{n\times n}$, the directed graph induced by the matrix M is denoted by $\mathcal{G}_M = (\mathcal{V}_M, \mathcal{E}_M)$, where $\mathcal{V}_M = \{1, 2, \ldots, n\}$ and $(j, i) \in \mathcal{E}_M$ iff $M_{ij} > 0$. We let $\mathcal{R}_M$ denote the set of roots of all directed spanning trees in the graph $\mathcal{G}_M$.

We use the following assumption on the functions f_i in (1).

Assumption 1: Each f_i is μ-strongly convex and its gradient is L-Lipschitz continuous, i.e., for any $x, x' \in \mathbb{R}^p$,

$$\begin{aligned} &\langle \nabla f_i(x) - \nabla f_i(x'), x - x' \rangle \geq \mu \|x - x'\|^2, \\ &\|\nabla f_i(x) - \nabla f_i(x')\| \leq L\|x - x'\|. \end{aligned} \tag{2}$$

Under Assumption 1, there exists a unique optimal solution $x^* \in \mathbb{R}^{1\times p}$ to problem (1).

II. A Push-Pull Gradient Method

The aggregated form of the proposed algorithm, termed push-pull gradient method (Push-Pull), works as follows: Initialize with any $\mathbf{x}_0$ and $\mathbf{y}_0 = \nabla F(\mathbf{x}_0)$, and update according to the following rule for $k \geq 0$,

$$\mathbf{x}_{k+1} = R(\mathbf{x}_k - \alpha \mathbf{y}_k), \tag{3a}$$
$$\mathbf{y}_{k+1} = C\left(\mathbf{y}_k + \nabla F(\mathbf{x}_{k+1}) - \nabla F(\mathbf{x}_k)\right), \tag{3b}$$

where $R, C \in \mathbb{R}^{n\times n}$. We make the following standard assumption on the matrices R and C.

Assumption 2: We assume that $R \in \mathbb{R}^{n\times n}$ is nonnegative[4] row-stochastic and $C \in \mathbb{R}^{n\times n}$ is nonnegative column-stochastic, i.e., $R\mathbf{1} = \mathbf{1}$ and $\mathbf{1}^\intercal C = \mathbf{1}^\intercal$.

Lemma 1: Under Assumption 2, the matrix R has a nonnegative left eigenvector $u^\intercal$ (w.r.t. eigenvalue 1) with $u^\intercal \mathbf{1} = n$, and the matrix C has a nonnegative right eigenvector v (w.r.t. eigenvalue 1) with $\mathbf{1}^\intercal v = n$ (see [31]).

The next condition ensures that 1 is a simple eigenvalue of both R and C.

Assumption 3: The diagonal entries of R and C are positive, i.e., $R_{ii} > 0$ and $C_{ii} > 0$ for all $i \in \mathcal{V}$.

Finally, we give the condition on the structures of $\mathcal{G}_R$ and $\mathcal{G}_C$. This assumption is weaker than requiring that both $\mathcal{G}_R$ and $\mathcal{G}_C$ are strongly connected.

Assumption 4: The graphs $\mathcal{G}_R$ and $\mathcal{G}_{C^T}$ each contain at least one spanning tree. Moreover, $\mathcal{R}_R \cap \mathcal{R}_{C^\intercal} \neq \emptyset$.

Supposing that we have a strongly connected communication graph $\mathcal{G}$, there are multiple ways to construct $\mathcal{G}_R$ and $\mathcal{G}_C$ satisfying Assumption 4. One trivial approach is to set $\mathcal{G}_R = \mathcal{G}_C = \mathcal{G}$. Another way is to pick at random $i_r \in \mathcal{V}$ and let $\mathcal{G}_R$ (respectively, $\mathcal{G}_C$) be a spanning tree (respectively, reversed spanning tree) contained in $\mathcal{G}$ with i_r as its root. Once graphs $\mathcal{G}_R$ and $\mathcal{G}_C$ are established, matrices R and C can be designed accordingly.

To provide some intuition for the development of this algorithm, let us consider the optimality condition for (1) in the following form:

$$\mathbf{x}^* \in \text{null}\{I - R\}, \quad \mathbf{1}^\intercal \nabla F(\mathbf{x}^*) = \mathbf{0}, \tag{4}$$

where R satisfies Assumption 2. Consider now the algorithm in (3). Suppose that the algorithm produces two sequences

[4]A matrix is nonnegative iff all its elements are nonnegative.

$\{\mathbf{x}_k\}$ and $\{\mathbf{y}_k\}$ converging to some points $\mathbf{x}_\infty$ and $\mathbf{y}_\infty$, respectively. Then from (3a) and (3b) we would have

$$(I-R)(\mathbf{x}_\infty - \alpha\mathbf{y}_\infty) + \alpha\mathbf{y}_\infty = 0, \quad (I-C)\mathbf{y}_\infty = 0. \quad (5)$$

If $\mathrm{span}\{I-R\}$ and $\mathrm{null}\{I-\mathbf{C}\}$ are disjoint[5], from (5) we would have $\mathbf{x}_\infty \in \mathrm{null}\{I-R\}$ and $\mathbf{y}_\infty = \mathbf{0}$. Hence $\mathbf{x}_\infty$ satisfies the first optimality condition in (4). Then by induction we know $\mathbf{1}^\intercal \nabla F(\mathbf{x}_\infty) = \mathbf{1}^\intercal \mathbf{y}_\infty = \mathbf{0}$, which is exactly the second optimality condition in (4).

The structure of the algorithm in (3) is similar to that of the DIGing algorithm proposed in [22] with the mixing matrices distorted (doubly stochastic matrices split into a row-stochastic matrix and a column-stochastic matrix). The $\mathbf{x}$-update can be seen as an inexact gradient step with consensus, while the $\mathbf{y}$-update can be viewed as a gradient tracking step. Such an asymmetric R-C structure design has already been used in the literature of average consensus [32]. However, we can not analyze the proposed optimization algorithm using linear dynamical systems since we have a nonlinear dynamics due to the gradient terms.

We now show how the proposed algorithm (3) unifies different types of distributed architecture. For the fully decentralized case, suppose we have a graph $\mathcal{G}$ that is undirected and connected. Then R and C can be chosen as symmetric matrices, in which case the proposed algorithm degrades to the one considered in [22]; if the graph is directed and strongly connected, we can set $\mathcal{G}_R = \mathcal{G}_C = \mathcal{G}$ and design the weights for R and C correspondingly.

To illustrate the less straightforward situation of (semi)-centralized networks, let us give a simple example. Consider a four-node star network composed by $\{1,2,3,4\}$ where node 1 is situated at the center and nodes 2, 3, and 4 are (bidirectionally) connected with node 1 but not connected to each other. In this case, the matrix R in our algorithm can be chosen as

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0.5 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0.5 & 0.5 & 0.5 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

For a graphical illustration, the corresponding network topologies of $\mathcal{G}_R$ and $\mathcal{G}_C$ are shown in Fig. 1. The central

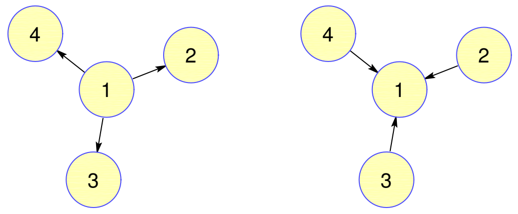

Fig. 1. The left is $\mathcal{G}_R$ and the right is $\mathcal{G}_C$.

node 1 pushes (diffuses) information regarding $x_{1,k}$ to the neighbors (in this case the entire network) through $\mathcal{G}_R$, while the others can only passively infuse the information from node 1. At the same time, node 1 pulls (collects) information regarding $y_{i,k}$ $(i = 2,3,4)$ from the neighbors through $\mathcal{G}_C$, while the other nodes can only actively comply with the request from node 1. This motivates the algorithm's name *push-pull gradient method*. Although nodes 2, 3, and 4 are updating their y_i's accordingly, these quantities do not have to contribute to the optimization procedure and will die out geometrically fast due to the weights in the last three rows of C. Consequently, in this special case, the local step-size α for agents 2, 3, and 4 can be set to 0. Without loss of generality, suppose $f_1(x) = 0, \forall x$. Then the algorithm becomes a typical centralized algorithm for minimizing $\sum_{i=2}^4 f_i(x)$ where the master node 1 utilizes the slave nodes 2, 3, and 4 to compute the gradient information in a distributed way.

Taking the above as an example for explaining the semi-centralized case, it is worth nothing that node 1 can be replaced by a strongly connected subnet in $\mathcal{G}_R$ and $\mathcal{G}_C$, respectively. Correspondingly, nodes 2, 3, and 4 can all be replaced by subnets as long as the information from the master layer in these subnets can be diffused to all the slave layer agents in $\mathcal{G}_R$, while the information from all the slave layer agents can be diffused to the master layer in $\mathcal{G}_C$. Specific requirements on connectivities of slave subnets can be understood by using the concept of rooted trees. We refer to the nodes as leaders if their roles in the network are similar to the role of node 1; and the other nodes are termed as followers. Note that after the replacement of the individual nodes by subnets, the network structure in all subnets are decentralized, while the relationship between leader subnet and follower subnets is master-slave. This is why we refer to such an architecture as semi-centralized.

Remark 1: There can be multiple variants of the proposed algorithm depending on whether the Adapt-then-Combine (ATC) strategy [33] is used in the $\mathbf{x}$-update and/or the $\mathbf{y}$-update (see Remark 3 in [22] for more details). Our following analysis can be easily adapted for these variants. We have also tested one of the variants in Section IV.

III. Convergence Analysis

In this section, we study the convergence properties of the proposed algorithm. We first define the following variables:

$$\bar{x}_k := \frac{1}{n}u^\intercal \mathbf{x}_k, \quad \bar{y}_k := \frac{1}{n}\mathbf{1}^\intercal \mathbf{y}_k.$$

Our strategy is to bound $\|\bar{x}_{k+1} - x^*\|_2$, $\|\mathbf{x}_{k+1} - \mathbf{1}\bar{x}_{k+1}\|_R$ and $\|\mathbf{y}_{k+1} - v\bar{y}_{k+1}\|_C$ in terms of linear combinations of their previous values, where $\|\cdot\|_R$ and $\|\cdot\|_C$ are specific norms to be defined later. In this way we establish a linear system of inequalities which allows us to derive the convergence results. The proof technique was inspired by [23], [24].

A. Preliminary Analysis

From the algorithm (3) and Lemma 1, we have

$$\bar{x}_{k+1} = \frac{1}{n}u^\intercal R(\mathbf{x}_k - \alpha\mathbf{y}_k) = \bar{x}_k - \frac{\alpha}{n}u^\intercal \mathbf{y}_k, \quad (6)$$

and

$$\bar{y}_{k+1} = \bar{y}_k + \frac{1}{n}\mathbf{1}^\intercal \left(\nabla F(\mathbf{x}_{k+1}) - \nabla F(\mathbf{x}_k)\right). \quad (7)$$

[5]This is indeed a consequence of Assumption 4.

With the initialization $\mathbf{y}_0 = \nabla F(\mathbf{x}_0)$, we obtain by induction that

$$\bar{y}_k = \frac{1}{n}\mathbf{1}^\intercal \nabla F(\mathbf{x}_k), \quad \forall k. \tag{8}$$

Let us further define $g_k := \frac{1}{n}\mathbf{1}^\intercal \nabla F(\mathbf{1}\bar{x}_k)$. Then, we obtain from relation (6) that

$$\begin{aligned}\bar{x}_{k+1} &= \bar{x}_k - \frac{\alpha}{n}u^\intercal\left(\mathbf{y}_k - v\bar{y}_k + v\bar{y}_k\right)\\ &= \bar{x}_k - \frac{\alpha}{n}u^\intercal v\bar{y}_k - \frac{\alpha}{n}u^\intercal\left(\mathbf{y}_k - v\bar{y}_k\right)\\ &= \bar{x}_k - \alpha' g_k - \alpha'(\bar{y}_k - g_k) - \frac{\alpha}{n}(u-\mathbf{1})^\intercal\left(\mathbf{y}_k - v\bar{y}_k\right),\end{aligned} \tag{9}$$

where

$$\alpha' := \frac{\alpha}{n}u^\intercal v. \tag{10}$$

We will show later that Assumption 4 ensures $\alpha' > 0$.

In view of (3) and Lemma 1, using (6) we have

$$\begin{aligned}\mathbf{x}_{k+1} - \mathbf{1}\bar{x}_{k+1} &= R(\mathbf{x}_k - \alpha\mathbf{y}_k) - \mathbf{1}\bar{x}_k + \frac{\alpha}{n}\mathbf{1}u^\intercal\mathbf{y}_k\\ &= R(\mathbf{x}_k - \mathbf{1}\bar{x}_k) - \alpha\left(R - \frac{\mathbf{1}u^\intercal}{n}\right)\mathbf{y}_k\\ &= \left(R - \frac{\mathbf{1}u^\intercal}{n}\right)(\mathbf{x}_k - \mathbf{1}\bar{x}_k) - \alpha\left(R - \frac{\mathbf{1}u^\intercal}{n}\right)(\mathbf{y}_k - \mathbf{1}\bar{y}_k),\end{aligned} \tag{11}$$

and from (7) we obtain

$$\begin{aligned}\mathbf{y}_{k+1} - v\bar{y}_{k+1} &= C\mathbf{y}_k - v\bar{y}_k\\ &\quad + \left(C - \frac{v\mathbf{1}^\intercal}{n}\right)(\nabla F(\mathbf{x}_{k+1}) - \nabla F(\mathbf{x}_k))\\ &= \left(C - \frac{v\mathbf{1}^\intercal}{n}\right)(\mathbf{y}_k - v\bar{y}_k)\\ &\quad + \left(C - \frac{v\mathbf{1}^\intercal}{n}\right)(\nabla F(\mathbf{x}_{k+1}) - \nabla F(\mathbf{x}_k)).\end{aligned} \tag{12}$$

B. Supporting Lemmas

Before proceeding to the main results, we state a few useful lemmas.

Lemma 2: Under Assumption 1, there holds

$$\|\bar{y}_k - g_k\|_2 \le \frac{L}{\sqrt{n}}\|\mathbf{x}_k - \mathbf{1}\bar{x}_k\|_2, \tag{13}$$

$$\|g_k\|_2 \le L\|\bar{x}_k - x^*\|_2. \tag{14}$$

In addition, when $\alpha' \le 2/(\mu + L)$, we have

$$\|\bar{x}_k - \alpha' g_k - x^*\|_2 \le (1 - \alpha'\mu)\|\bar{x}_k - x^*\|_2, \quad \forall k. \tag{15}$$

Proof: See [34] (extended version of this paper) Appendix VI-A. ■

Lemma 3: Suppose Assumption 2 holds, and assume that $\mathcal{R}_R \neq \emptyset$ and $\mathcal{R}_{C^\intercal} \neq \emptyset$. Then, $\mathcal{R}_R \cap \mathcal{R}_{C^\intercal} \neq \emptyset$ iff $u^\intercal v > 0$.

Proof: See [34] Appendix VI-B. ■

Lemma 3 explains why Assumption 4 is essential for the Push-Pull algorithm (3) to work. Without the condition, $\alpha' = 0$ by its definition in (10).

Lemma 4: Suppose Assumptions 2-4 hold. Let ρ_R and ρ_C be the spectral radii of $(R - \mathbf{1}u^\intercal/n)$ and $(C - v\mathbf{1}^\intercal/n)$, respectively. Then, we have $\rho_R < 1$ and $\rho_C < 1$.

Proof: See [34] Appendix VI-C. ■

Lemma 5: There exist matrix norms $\|\cdot\|_R$ and $\|\cdot\|_C$ such that $\sigma_R := \|R - \frac{\mathbf{1}u^\intercal}{n}\|_R < 1$, $\sigma_C := \|C - \frac{v\mathbf{1}^\intercal}{n}\|_C < 1$, and σ_R and σ_C are arbitrarily close to ρ_R and ρ_C, respectively.

Proof: See [31, Lemma 5.6.10] and the discussions thereafter. ■

In the rest of this paper, with a slight abuse of notation, we do not distinguish between the vector norms on $\mathbb{R}^n$ and their induced matrix norms.

Lemma 6: Given an arbitrary norm $\|\cdot\|$, for any $W \in \mathbb{R}^{n\times n}$ and $\mathbf{x} \in \mathbb{R}^{n\times p}$, we have $\|W\mathbf{x}\| \le \|W\|\|\mathbf{x}\|$. For any $w \in \mathbb{R}^{n\times 1}$ and $x \in \mathbb{R}^{1\times p}$, we have $\|wx\| = \|w\|\|x\|_2$.

Proof: See [34] Appendix VI-D. ■

Lemma 7: There exist constants $\delta_{C,R}, \delta_{C,2}, \delta_{R,C}, \delta_{R,2} > 0$ such that for all $\mathbf{x} \in \mathbb{R}^{n\times p}$, we have $\|\mathbf{x}\|_C \le \delta_{C,R}\|\mathbf{x}\|_R$, $\|\mathbf{x}\|_C \le \delta_{C,2}\|\mathbf{x}\|_2$, $\|\mathbf{x}\|_R \le \delta_{R,C}\|\mathbf{x}\|_C$, and $\|\mathbf{x}\|_R \le \delta_{R,2}\|\mathbf{x}\|_2$. In addition, with a proper rescaling of the norms $\|\cdot\|_R$ and $\|\cdot\|_C$, we have $\|\mathbf{x}\|_2 \le \|\mathbf{x}\|_R$ and $\|\mathbf{x}\|_2 \le \|\mathbf{x}\|_C$.

Proof: The result follows from the equivalence relation of all norms on $\mathbb{R}^n$ and Definition 1. ■

C. Main Results

The following lemma establishes a linear system of inequalities that bound $\|\bar{x}_{k+1} - x^*\|_2$, $\|\mathbf{x}_{k+1} - \mathbf{1}\bar{x}_k\|_R$ and $\|\mathbf{y}_{k+1} - v\bar{y}_k\|_C$.

Lemma 8: Under Assumptions 1-4, when $\alpha' \le 2/(\mu+L)$, we have the following linear system of inequalities:

$$\begin{bmatrix}\|\bar{x}_{k+1} - x^*\|_2\\ \|\mathbf{x}_{k+1} - \mathbf{1}\bar{x}_{k+1}\|_R\\ \|\mathbf{y}_{k+1} - v\bar{y}_{k+1}\|_C\end{bmatrix} \le A\begin{bmatrix}\|\bar{x}_k - x^*\|_2\\ \|\mathbf{x}_k - \mathbf{1}\bar{x}_k\|_R\\ \|\mathbf{y}_k - v\bar{y}_k\|_C\end{bmatrix}, \tag{16}$$

where the inequality is to be taken component-wise, and elements of the transition matrix $A = [a_{ij}]$ are given by:

$$\begin{bmatrix}a_{11}\\ a_{21}\\ a_{31}\end{bmatrix} = \begin{bmatrix}1 - \alpha'\mu\\ \alpha\sigma_R\|v - \mathbf{1}\|_R L\\ \alpha\sigma_C\delta_{C,2}\|Rv\|_2 L^2\end{bmatrix},$$

$$\begin{bmatrix}a_{12}\\ a_{22}\\ a_{32}\end{bmatrix} = \begin{bmatrix}\frac{\alpha' L}{\sqrt{n}}\\ \sigma_R\left(1 + \alpha\|v - \mathbf{1}\|_R\frac{L}{\sqrt{n}}\right)\\ \sigma_C\delta_{C,2}L\left(\|R - I\|_2 + \alpha\|Rv\|_2\frac{L}{\sqrt{n}}\right)\end{bmatrix},$$

$$\begin{bmatrix}a_{13}\\ a_{23}\\ a_{33}\end{bmatrix} = \begin{bmatrix}\frac{\alpha\|u - \mathbf{1}\|_2}{n}\\ \alpha\sigma_R\delta_{R,C}\\ \sigma_C\left(1 + \alpha\delta_{C,2}\|R\|_2 L\right)\end{bmatrix}.$$

Proof: See [34] Appendix VI-E. ■

In light of Lemma 8, $\|\bar{x}_k - x^*\|_2$, $\|\mathbf{x}_k - \mathbf{1}\bar{x}_k\|_R$ and $\|\mathbf{y}_k - v\bar{y}_k\|_C$ all converge to 0 linearly at rate $\mathcal{O}(\rho_A^k)$ if the spectral radius of A satisfies $\rho_A < 1$. The next lemma provides some sufficient conditions for the relation $\rho_A < 1$ to hold.

Lemma 9: Given a nonnegative, irreducible matrix $M = [m_{ij}] \in \mathbb{R}^{3\times 3}$ with $m_{11}, m_{22}, m_{33} < \lambda^*$ for some $\lambda^* > 0$. A necessary and sufficient condition for $\rho_M < \lambda^*$ is $\det(\lambda^* I - M) > 0$.

Proof: See [34] Appendix VI-F. ■

Now, we are ready to deliver our main convergence result for the Push-Pull algorithm in (3).

Theorem 1: Suppose Assumptions 1-4 hold and

$$\alpha \le \min\left\{\frac{2c_3}{c_2+\sqrt{c_2^2+4c_1c_3}}, \frac{(1-\sigma_C)}{2\sigma_C\delta_{C,2}\|R\|_2 L}\right\}, \quad (17)$$

where c_1, c_2 and c_3 are constants (see [34]). Then, the quantities $\|\bar{x}_k - x^*\|_2$, $\|\mathbf{x}_k - \mathbf{1}\bar{x}_k\|_R$ and $\|\mathbf{y}_k - v\bar{y}_k\|_C$ all converge to 0 at the linear rate $\mathcal{O}(\rho_A^k)$ with $\rho_A < 1$, where ρ_A denotes the spectral radius of A.

Proof: See [34]. ■

Remark 2: When α is sufficiently small, it can be shown that $\rho_A \simeq 1 - \alpha'\mu$, in which case the Push-Pull algorithm is comparable to its centralized counterpart with step-size α'.

IV. SIMULATIONS

In this section, we provide numerical comparisons of a few different algorithms under various network settings. Our settings for objective functions are the same as that described in [22]. Each node in the network holds a Huber-typed objective function $f_i(x)$ and the goal is to optimize the total Huber loss $f(x) = \sum_{i=1}^n f_i(x)$. The objective functions f_i's are randomly generated but are manipulated such that the global optimizer x^* is located at the ℓ_2^2 zone of $f(x)$ while the origin (which is set to be the initial state of $\mathbf{x}_k$ for all involved algorithms) is located outside of that zone.

We first conduct an experiment over time-invariant directed graphs. The network is generated randomly with 12 nodes and 24 unidirectional links (at most $12 \times 11 = 131$ possible links in this case) and is guaranteed to be strongly connected. We test our proposed algorithm, Push-Pull, against Push-DIGing [22] and Xi-Row [24]. Among these algorithms, Push-DIGing is a push-sum based algorithm which only needs push operations for information dissemination in the network; Xi-row is an algorithm that only uses row stochastic mixing matrices and thus only needs pull operations to fetch information in the network; in comparison, our algorithm needs the network to support both push operations and pull operations. The per-node storage complexity of Push-Pull (or Push-DIGing) is $O(p)$ while that of Xi-row is $O(n+p)$. Note that at each iteration, the amount of data transmitted over each link also scales at such orders for these algorithms, respectively. For large-scale networks ($n \gg p$), Xi-row may suffer from high needs in storage/bandwidth and/or become under limited transmission rates. The evolution of the (normalized) residual $\frac{\|\mathbf{x}_k - \mathbf{x}^*\|_2^2}{\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2}$ is illustrated in Fig. 2. The step-sizes are hand-tuned for all the algorithms to optimize the convergence speed.

Although our algorithm is designed and analyzed over time-invariant directed graphs, its extension to time-varying directed graphs is straightforward. Let us use the above generated directed graph as a base graph. To test our theory for a leader-follower architecture, we randomly select multiple nodes as leaders and randomly add enough links between the leaders (in this example, number of leaders is 2) so that they form a strongly connected subnet. Then at each iteration,

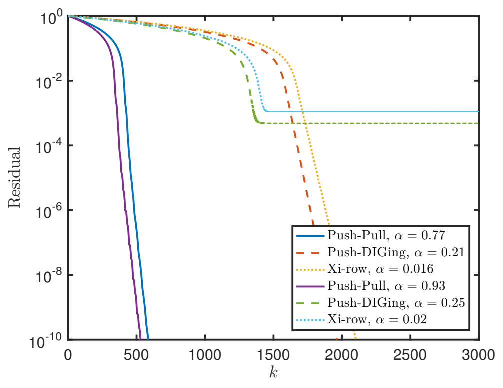

Fig. 2. Plots of (normalized) residuals against number of iterations over a time-invariant directed graph.

only 50% randomly chosen links will be activated. In Fig. 3, we plot the performance of Push-Pull-half (a variant of Push-Pull where the ATC strategy is not employed in the $\mathbf{y}$-update; it needs only one round of communication at each iteration) and Push-DIGing without considering the leader-follower structure. That is, for Push-Pull-half and Push-DIGing, a time-varying directed graph sequence based on random link activation is used where the underlying graph is strongly connected.

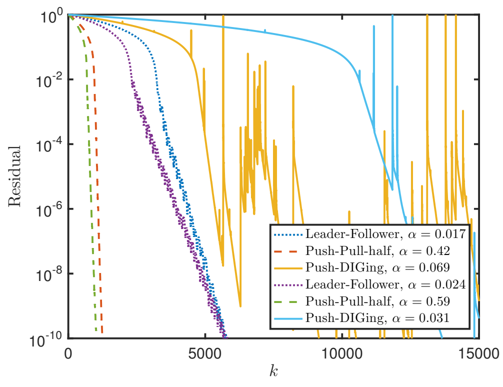

Fig. 3. Plots of (normalized) residuals against number of iterations over a time-varying directed graph sequence.

Then in Fig. 3 we further show the performance of Push-Pull under the leader-follower architecture. The major difference on the graph sequence is that, in the leader-follower architecture, all the outbound information links of the leader subnet are not used when performing the $\mathbf{y}$-update; all the inbound information links of the leader subnet are not used when performing the $\mathbf{x}$-update. Note that in such a way, the union of all directed graphs corresponding to R_k (or C_k) is not strongly connected. The numerical results show, as expected, that the convergence of Push-Pull under the leader-follower architecture is slower than that of Push-Pull-half with strongly connected underlying graphs.

In the experiment, we observe that there are many spikes on the residual curve of Push-DIGing. Push-DIGing for time-varying graphs can be numerically unstable due to the use of division operations in the algorithm and the divisors can scale badly at the order of $\Omega(n^{-Bn})$ where n is the number of nodes and B a bounded constant that describes the connectivity of time-varying graphs (the smaller B is, the better the network is connected; see [22] for the definition of B). A simple network with number of nodes $n = 15$ and time-varying constant $B = 10$ will easily give a number that is recognized by common computers (using double-precision floating-point format) as 0. As a contrast, in Push-Pull, there is no divisors that scale at such level. In addition, the theoretical upper bound on the step-size of Push-DIGing also scales at the order of $O(n^{-Bn})$. This implies that Push-DIGing will not work well for either large scale networks or time-varying networks with large variations.

V. Conclusions

In this paper, we have studied the problem of distributed optimization over a network. In particular, we proposed a new distributed gradient-based method (Push-Pull) where each node maintains estimates of the optimal decision variable and the average gradient of the agents' objective functions. From the viewpoint of an agent, the information about the decision variable is pushed to its neighbors, while the information about the gradients is pulled from its neighbors. This method works for different types of distributed architecture, including decentralized, centralized, and semi-centralized architecture. We have showed that the algorithm converges linearly for strongly convex and smooth objective functions over a directed static network. In the simulations, we have demonstrated the effectiveness of the proposed algorithm for both static and time-varying directed networks.

References

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[2] Z. Peng, Y. Xu, M. Yan, and W. Yin, "Arock: an algorithmic framework for asynchronous parallel coordinate updates," *SIAM Journal on Scientific Computing*, vol. 38, no. 5, pp. A2851–A2879, 2016.

[3] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the Linear Convergence of the ADMM in Decentralized Consensus Optimization," *IEEE Transactions on Signal Processing*, vol. 62, no. 7, pp. 1750–1761, 2014.

[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.

[5] A. Mokhtari, W. Shi, Q. Ling, and A. Ribeiro, "A Decentralized Second-Order Method with Exact Linear Convergence Rate for Consensus Optimization," *arXiv preprint arXiv:1602.00596*, 2016.

[6] D. Varagnolo, F. Zanella, A. Cenedese, G. Pillonetto, and L. Schenato, "Newton-raphson consensus for distributed convex optimization," *IEEE Transactions on Automatic Control*, vol. 61, no. 4, pp. 994–1009, 2016.

[7] A. Olshevsky, "Linear time average consensus and distributed optimization on fixed graphs," *SIAM Journal on Control and Optimization*, vol. 55, no. 6, pp. 3990–4014, 2017.

[8] K. Scaman, F. Bach, S. Bubeck, Y. T. Lee, and L. Massoulié, "Optimal algorithms for smooth and strongly convex distributed optimization in networks," *arXiv preprint arXiv:1702.08704*, 2017.

[9] C. A. Uribe, S. Lee, A. Gasnikov, and A. Nedić, "Optimal algorithms for distributed optimization," *arXiv preprint arXiv:1712.00232*, 2017.

[10] J. Xu, S. Zhu, Y. Soh, and L. Xie, "Augmented Distributed Gradient Methods for Multi-Agent Optimization Under Uncoordinated Constant Stepsizes," in *Proceedings of the 54th IEEE Conference on Decision and Control (CDC)*, 2015, pp. 2055–2060.

[11] A. Nedić, A. Olshevsky, W. Shi, and C. A. Uribe, "Geometrically convergent distributed optimization with uncoordinated step-sizes," in *American Control Conference (ACC), 2017*. IEEE, 2017, pp. 3950–3955.

[12] P. Di Lorenzo and G. Scutari, "Next: In-network nonconvex optimization," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 2, pp. 120–136, 2016.

[13] W. Shi, Q. Ling, G. Wu, and W. Yin, "A Proximal Gradient Algorithm for Decentralized Composite Optimization," *IEEE Transactions on Signal Processing*, vol. 63, no. 22, pp. 6013–6023, 2015.

[14] Z. Li, W. Shi, and M. Yan, "A decentralized proximal-gradient method with network independent step-sizes and separated convergence rates," *arXiv preprint arXiv:1704.07807*, 2017.

[15] A. Nedić and A. Olshevsky, "Distributed optimization over time-varying directed graphs," *IEEE Transactions on Automatic Control*, vol. 60, no. 3, pp. 601–615, 2015.

[16] C. Xi and U. A. Khan, "On the linear convergence of distributed optimization over directed graphs," *arXiv preprint arXiv:1510.02149*, 2015.

[17] J. Zeng and W. Yin, "ExtraPush for Convex Smooth Decentralized Optimization over Directed Networks," *arXiv preprint arXiv:1511.02942*, 2015.

[18] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "Convergence of asynchronous distributed gradient methods over stochastic networks," *IEEE Transactions on Automatic Control*, 2017.

[19] T. Wu, K. Yuan, Q. Ling, W. Yin, and A. H. Sayed, "Decentralized consensus optimization with asynchrony and delays," in *Signals, Systems and Computers, 2016 50th Asilomar Conference on*. IEEE, 2016, pp. 992–996.

[20] S. Pu and A. Garcia, "A flocking-based approach for distributed stochastic optimization," *Operations Research*, vol. 1, pp. 267–281, 2018.

[21] S. Pu and A. Nedić, "Distributed stochastic gradient tracking methods," *arXiv preprint arXiv:1805.11454*, 2018.

[22] 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.

[23] G. Qu and N. Li, "Harnessing smoothness to accelerate distributed optimization," *IEEE Transactions on Control of Network Systems*, 2017.

[24] C. Xi, V. S. Mai, R. Xin, E. H. Abed, and U. A. Khan, "Linear convergence in optimization over directed graphs with row-stochastic matrices," *IEEE Transactions on Automatic Control*, 2018.

[25] E. Wei, A. Ozdaglar, and A. Jadbabaie, "A distributed newton method for network utility maximization–i: Algorithm," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2162–2175, 2013.

[26] A. Mokhtari, Q. Ling, and A. Ribeiro, "Network newton distributed optimization methods," *IEEE Transactions on Signal Processing*, vol. 65, no. 1, pp. 146–161, 2017.

[27] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-Based Computation of Aggregate Information," in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003, pp. 482–491.

[28] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Push-sum distributed dual averaging for convex optimization," in *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. IEEE, 2012, pp. 5453–5458.

[29] R. Xin and U. A. Khan, "A linear algorithm for optimization over directed graphs with geometric convergence," *arXiv preprint arXiv:1803.02503*, 2018.

[30] C. Godsil and G. F. Royle, *Algebraic graph theory*. Springer Science & Business Media, 2013, vol. 207.

[31] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 1990.

[32] K. Cai and H. Ishii, "Average consensus on general strongly connected digraphs," *Automatica*, vol. 48, no. 11, pp. 2750–2761, 2012.

[33] A. Sayed, "Diffusion Adaptation over Networks," *Academic Press Library in Signal Processing*, vol. 3, pp. 323–454, 2013.

[34] S. Pu, W. Shi, J. Xu, and A. Nedich, "A push-pull gradient method for distributed optimization in networks," *arXiv preprint arXiv:1803.07588*, 2018.