A general framework for decentralized optimization with first-order methods

Ran Xin, Shi Pu, Angelia Nedić, and Usman A. Khan

Abstract—Decentralized optimization to minimize a finite sum of functions over a network of nodes has been a significant focus within control and signal processing research due to its natural relevance to optimal control and signal estimation problems. More recently, the emergence of sophisticated computing and large-scale data science needs has led to a resurgence of activity in this area. In this article, we discuss decentralized first-order gradient methods, which have found tremendous success in control, signal processing, and machine learning problems, where such methods, due to their simplicity, serve as the first method of choice for many complex inference and training tasks. In particular, we provide a general framework of decentralized first-order methods that is applicable to undirected and directed communication networks alike, and show that much of the existing work on optimization and consensus can be related explicitly to this framework. We further extend the discussion to decentralized stochastic first-order methods that rely on stochastic gradients at each node and describe how local variance reduction schemes, previously shown to have promise in the centralized settings, are able to improve the performance of decentralized methods when combined with what is known as gradient tracking. We motivate and demonstrate the effectiveness of the corresponding methods in the context of machine learning and signal processing problems that arise in decentralized environments.

Index Terms—Decentralized optimization, machine learning, stochastic methods, consensus, gradient descent

I. INTRODUCTION

Minimizing a cost function to select an optimal action or decision has been an important problem in science, engineering, and mathematics. The cost function, say $F: \mathbb{R}^p \rightarrow \mathbb{R}$, typically quantifies the loss in fitting data or measurements under a model parameterized by $x \in \mathbb{R}^p$. An optimal model or decision $x^*$ is often chosen as the one that minimizes the corresponding loss $F$. Optimization theory and algorithms $\mathcal{H}$ provide the fundamental tools to address such problems. Examples include the classical signal estimation and optimal control problems, where the goal in the former is to minimize the estimation error and in the latter is to minimize the cost of control actions. More recently, with the advent of modern computational machinery, complex nonlinear problems, such as image classification, natural language processing, and deep learning, have enabled a resurgence of interest in the domain of optimization theory and methods.

Many numerical optimization methods for minimizing a smooth function $F$ are built on a simple observation that moving along the negative gradient $-\nabla F$ decreases the function $F$. Thus, given $x$ and $\nabla F(x)$, both in $\mathbb{R}^p$, a protocol to decrease $F(x)$ is $x_{+} = x - \alpha \nabla F(x)$, if $\alpha$, which controls the size of the step taken in the descent direction, is small enough. With the descent in the negative gradient direction, we have that $F(x_{+}) \leq F(x)$. The algorithm that recursively applies the aforementioned protocol is well-known as gradient descent and it minimizes the corresponding cost function under certain conditions on the function $F$ and the step-size $\alpha$. Moreover, gradient descent is a first-order method as it only uses the first-derivative (gradient) of the cost, in contrast to, for example, second-order methods that typically compute the inverse of the Hessian $\nabla^2 F$ of the cost function at each iteration.

In this article, we focus on decentralized optimization problems where data samples are available across multiple nodes, such as machines, sensors, robots or mobile devices. The nodes communicate with each other according to a peer-to-peer network, without a central coordinator, and solve the underlying optimization problem in a cooperative manner. Such problems are prevalent in modern-day machine learning where, for example, a large collection of images are stored on multiple machines in a data center for the purpose of image classification. Moreover, classical applications like sensor networks and robotic swarms also fit this paradigm where the sensors and robots collect measurements in order to learn an underlying phenomenon, navigate an environment, or decide on an optimal control action. In such settings, the data samples available at the $i$th node lead to a local cost $f_i$, and the goal of the networked nodes is to agree on a minimizer of the global cost $F = \frac{1}{n} \sum_{i=1}^{n} f_i$ based on the data across all $n$ nodes.

In related applications of practical interest, raw data sharing among the nodes is often not permitted due to the private nature of data, such as text messages or medical images, or is inefficient due to limited communication resources.

Decentralized first-order methods thus rely on information exchange among the nodes and local gradient computation to build the solution of the global optimization problem. Each node $i$ iterates on a local state variable $x_i^k$ that is an estimate of a minimizer $x^*$ of the global cost function $F$ at iteration $k$, and recursively updates this estimate according to the estimates of neighboring nodes and local gradients. In other words, the nodes do not share their raw data (local gradients) directly and only communicate their estimates (state variables) and
perhaps a few other auxiliary variables. Classical solutions along these lines can be found in [5]–[7] that are built on average consensus and in [8] where a diffusion principle is used for agreement. More recent developments include gradient tracking [9]–[11], where the local descent direction $\nabla f_i$ at each node $i$ is replaced with a local iterative tracker of the gradient of the global cost function $\nabla F$. Thus, as these local trackers approach $\nabla F$, each local iterate $x^i_k$ descends in the global direction and converges exponentially (linearly on the log scale) to the unique minimizer for smooth and strongly convex problems [12]–[15], in a similar way as the centralized gradient descent. Our primary focus in this article is on the class of smooth and strongly convex problems for the ease of illustrating the key technical ideas. We emphasize however that the decentralized optimization algorithms described herein apply to smooth non-convex problems directly [16]–[21].

The first half of this article is devoted to decentralized first-order methods based on gradient tracking focusing on a recently introduced algorithm, AB [14] or Push–Pull [15], that utilizes a novel application of both row and column stochastic network weights to achieve linear convergence for smooth and strongly convex problems. Since doubly stochastic weights are not used, the corresponding methods are applicable to both undirected and directed networks. We further describe how AB/Push–Pull unifies much of the existing work on decentralized first-order methods that use gradient tracking and subsumes several non-trivial average consensus algorithms as special cases. Moreover, we emphasize the push and pull communication aspects enabled by the column and row stochastic weights in AB/Push–Pull and show how AB/Push–Pull unifies various communication architectures.

The second half of this article is devoted to decentralized stochastic first-order methods, describing the current state-of-the-art and open problems where some progress has been made only recently. In decentralized stochastic gradient methods, each node has access only to an imperfect gradient of its local cost, which results from either an incomplete knowledge of the true gradient or sampling a small subset from a large number of local data samples. In this context, we describe how gradient tracking, previously successful in non-stochastic cases, does not necessarily lead to the same performance improvement and show that exact linear convergence to the global minimum (for smooth and strongly convex problems) can be obtained when gradient tracking is further combined with variance reduction, well-known in centralized optimization. We emphasize that much of the existing work on decentralized stochastic gradient methods has focused on undirected networks and the results on directed networks are rather restrictive.

A. Literature Survey

Decentralized optimization has been a topic of significant research over the past decade, see e.g., [5]–[8], [14], [15], [22]–[26]. Methods applicable to undirected networks, based on average consensus protocols [27]–[29], include [6], [12], [30], [31] that require doubly stochastic network weight matrices. Relevant work that builds on diffusion principles can be found in [8], [19], [32]–[36]. For arbitrary directed networks, it may not be feasible to construct doubly stochastic weights and hence the corresponding decentralized methods build on consensus with row and/or column stochastic weights. For example, the methods in [13], [37]–[42] need the weights to be column stochastic and use the push-sum correction [43]–[45] that requires a division with a certain eigenvector estimate of the underlying weight matrix. In contrast, the methods in [46], [47] are based on surplus consensus [48] that employs both row and column stochastic weights simultaneously.

Decentralized stochastic optimization in general can be divided into two types: (i) online, where an imprecise (stochastic) gradient is drawn from an underlying probability distribution at each node; or (ii) offline/batch, where a finite collection of data samples is available locally at each node and a stochastic gradient is computed from samples drawn randomly from the local batch. Related work on decentralized online problems can be found, e.g., in [7], [18], [19], [32], [33], [49]–[54]. Stochastic optimization over finite data in the centralized settings have garnered a strong research activity where modern methods hinge on certain variance-reduction techniques that leverage the finite sum structure of the cost function to accelerate the standard stochastic gradient descent (SGD) [55], [56]; see, e.g., [57]–[63]. Existing variance-reduced decentralized stochastic methods can be found in [64]–[73]. Although not discussed in this article, second-order methods and algorithms based on the curvature of the cost functions can be found in [74]–[78]. Similarly, ADMM (alternating direction method of multipliers) and other primal-dual methods have also been used in decentralized optimization [36], [79]–[80]. See also related work in [87]–[89], which considers methods based on dual gradients. Methods that incorporate communication and computation imperfection and trade-offs, for example, time-varying and random graphs, asynchronous methods and quantization can be found in [90]–[97]. Another interesting line of work is to use tools from systems and control theory to analyze decentralized optimization algorithms [98]–[101].

B. Outline of the Article

We now describe the rest of this article. In Section II we provide the problem formulation, examples, and preliminaries on convex functions, communication graphs, and nonnegative matrices. Section III discusses early work on decentralized gradient descent and shows its applicability to directed networks with the help of row and column stochastic weights. We then describe gradient tracking and introduce the AB/Push–Pull algorithm in Section IV which further includes a sketch of the analysis, communication architectures, and accelerated methods. Section V describes how AB/Push–Pull provides a general framework to capture many first-order methods based on gradient tracking. We then begin the discussion on decentralized stochastic first-order methods over both undirected and directed graphs in Sections VI and VII. We show how gradient tracking alone is unable to ensure exact linear convergence in this setting (Section VI), but is subsequently achieved when gradient tracking is further combined with variance reduction (Section VII). Section VIII provides a detailed numerical study on the performance, including speed-up, and convergence of related algorithms. Finally, Section IX concludes the article.

The goal of this article is to provide an in-depth overview
of decentralized first-order methods and to further expose the reader with rigorous yet intuitive arguments to follow the technical analysis. In several remarks distributed throughout this article, we highlight the analysis techniques, practical aspects, and other salient features of the corresponding algorithms.

C. Notation

We use lowercase letters to denote scalars in $\mathbb{R}$, lowercase bold letters to denote vectors, and uppercase letters to denote matrices. For a vector $\mathbf{x} \in \mathbb{R}^p$, we denote its $i$th element by $[\mathbf{x}]_i$. For a set $\mathcal{S}$, we use $|\mathcal{S}|$ to denote its cardinality. The matrix $I_p$ is the $p \times p$ identity, and $1_p$ (resp. $0_p$) is the $p$-dimensional column vector of all ones (resp. zeros). For two matrices $X, Y$, $X \otimes Y$ denotes their Kronecker product. For two vectors $\mathbf{a}, \mathbf{b}$, we denote $\mathbf{a} \rightarrow \mathbf{b}$ or $\mathbf{b} \rightarrow \mathbf{a}$, as $\lim_{k \to \infty} \|\mathbf{a}_k - \mathbf{b}_k\|_2 = 0$; in particular, we denote $\mathbf{a}_k \rightarrow \mathbf{a}$ as $\lim_{k \to \infty} \mathbf{a}_k = \mathbf{a}$ for some $\mathbf{a} \in \mathbb{R}^p$. The spectral radius of a matrix $X$ is denoted by $\rho(X)$, while its spectral norm is denoted by $\|X\|_2$.

II. PROBLEM FORMULATION AND MOTIVATION

In this section, we introduce the canonical forms of the decentralized optimization problems and emphasize real-world scenarios where such problems are applicable and essential. Decentralized optimization is finite sum minimization formulated over a network of nodes. Formally, the goal of the nodes is to solve in a cooperative manner

**Problem P1:** $\min_{\mathbf{x} \in \mathbb{R}^p} F(\mathbf{x}), \quad F(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{x}),$

where each $f_i$ is only locally accessible and processed by node $i$ and is not shared with any other node. The cooperation (information exchange) among the nodes is peer-to-peer without the existence of a central coordinator and is typically modeled as a graph, see Fig. 1. In many cases of practical interest, each local cost $f_i$ can be further decomposed as a weighted sum over local data samples available at node $i$ and Problem P1 can be refined as a batch problem:

**Problem P2:** $\min_{\mathbf{x} \in \mathbb{R}^p} F(\mathbf{x}), \quad F(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{x}),$

where $f_i = \frac{1}{m_i} \sum_{j=1}^{m_i} f_{i,j}(\mathbf{x})$.

The paradigm of decentralized optimization preserves the privacy of local data and achieves data parallelism, thus enabling effective means for flexible parallel computation. We provide some illustrative examples of this mathematical formulation in the following Section II-A. For preliminaries on related technical concepts, see Section II-B.

A. Examples

The finite sum minimization Problems P1 and P2 are quite prevalent in signal processing, control, and machine learning. We provide some simple examples below.

1) Signal-plus-noise model

In classical signal processing, we are often interested in finding an unknown signal $\mathbf{x} \in \mathbb{R}^p$ based on the measurements $y_i = \mathbf{h}_i^\top \mathbf{x} + v_i$, obtained by a collection of sensors indexed by $i$, where $\mathbf{h}_i \in \mathbb{R}^p$ is the sensing matrix at sensor $i$ and $v_i \in \mathbb{R}$ is measurement noise. Finding $\mathbf{x} \in \mathbb{R}^p$ at each sensor $i$ may be formulated as a local minimization problem in terms of the squared error, i.e., $\min_{\mathbf{x} \in \mathbb{R}^p} (y_i - \mathbf{h}_i^\top \mathbf{x})^2$. However, since this problem may be ill-conditioned and the collected measurements have noise, collaboration among the sensors leads to a more robust estimate. The resulting formulation is

$$\min_{\mathbf{x} \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{x}), \quad f_i(\mathbf{x}) := (y_i - \mathbf{h}_i^\top \mathbf{x})^2,$$

which is also known as the least-squares problem.

2) Linear models for binary classification

The decentralized aspects of Problems P1 and P2 are more pronounced when we consider high-dimensional and potentially private or proprietary user data. For example, consider a set of $n$ users that are interested in learning a classifier to distinguish between male and female faces. Each user $i$ holds a collection of $m_i$ images, vectorized as $\mathbf{z}_{i,j} \in \mathbb{R}^p$, that are labeled $y_{i,j} = +1$ for a male face, or $y_{i,j} = -1$ for a female face. The classification may be performed through a linear classifier with parameters $\mathbf{x} \in \mathbb{R}^p$ and $b \in \mathbb{R}$, i.e., a hyperplane $y = \mathbf{x}^\top \mathbf{z} + b$ that separates the images from two classes. Clearly, a classifier trained on the collection of all images across all users will have superior performance than a locally trained classifier whose performance significantly depends on the size and quality of the local images. However, bringing all local images to a central server may be expensive depending on the size of the images and further requires sharing personal information. This discussion motivates the use of decentralized optimization to solve the underlying binary classification problem, for example, with the help of the logistic regression model:

$$\min_{\mathbf{x} \in \mathbb{R}^p, b \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{m_i} \sum_{j=1}^{m_i} f_{i,j}(\mathbf{x}) + \frac{\lambda}{2} \|\mathbf{x}\|_2^2,$$

where the logistic loss $f_{i,j}$ quantifies the error in the linear classifier and $\frac{\lambda}{2} \|\mathbf{x}\|_2^2$, for some $\lambda > 0$, is a regularization term to prevent overfitting of the data.

3) Empirical risk minimization

Problem P2 also arises as an approximation of expected risk minimization, see e.g., \cite{3} for additional details. In this context, the problem of interest is to find some model $\mathcal{H}$, parameterized by $\mathbf{x} \in \mathbb{R}^p$, that maps an input $\mathbf{z} \in \mathbb{R}^{d_i}$ to its corresponding output $\mathbf{y} \in \mathbb{R}^{d_y}$. The setup requires defining a loss function $\mathcal{L}(\mathcal{H}(\mathbf{z}; \mathbf{x}), \mathbf{y})$ that quantifies the mismatch between the model prediction $\mathcal{H}(\mathbf{z}; \mathbf{x})$, under the parameter $\mathbf{x}$, and the actual output data $\mathbf{y}$. Assuming that the data $(\mathbf{z}, \mathbf{y})$
belongs to an underlying distribution \( \mathcal{P} \), the goal here is to find the optimal parameter \( x^* \) that minimizes the expected loss over \( \mathcal{P} \), i.e., \( \min_{x \in \mathbb{R}^p} \mathbb{E}_{(x,y) \sim \mathcal{P}} [L(H(x; x), y)] \). However, the distribution \( \mathcal{P} \) is often intractable in practice and each node \( i \) is either able to draw random samples from this distribution in real-time (leading to an online stochastic formulation) or has access to a large set of data samples \( \{(z_{i,j}, y_{i,j})\}_{j=1}^m \) drawn from \( \mathcal{P} \). In the latter case with batch data, the average loss incurred by all data across all nodes serves as an appropriate surrogate for the expected risk and the corresponding problem is often referred to as empirical risk minimization, i.e.,

\[
\min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n \frac{1}{m_i} \sum_{j=1}^{m_i} f_{i,j}(x), \quad f_{i,j}(x) := \mathcal{L}(H(z_{i,j}; x), y_{i,j}).
\]

This formulation captures a wide range of machine learning models, including deep neural networks.

B. Preliminaries

We now briefly describe some mathematical concepts that aid the technical discussion in this article.

1) Convex functions

A convex function \( f : \mathbb{R}^p \to \mathbb{R} \) is such that for any \( 0 < \gamma < 1 \) and \( \forall x_1, x_2 \in \mathbb{R}^p \),

\[
f(\gamma x_1 + (1-\gamma)x_2) \leq \gamma f(x_1) + (1-\gamma)f(x_2).
\]

The above definition says that a convex function always stays below a line that connects any two points on the function. If \( f \) is differentiable, an equivalent definition of convexity is that it lies above all of its tangents, i.e., \( \forall x_1, x_2 \in \mathbb{R}^p \),

\[
f(x_2) \geq f(x_1) + \nabla f(x_1)^\top (x_2 - x_1),
\]

where \( \nabla f(x_1) \) denotes the gradient (derivative) of \( f \) at \( x_1 \). The convexity conditions above are general and do not guarantee the existence of a global minimum. The notion of strong convexity, as defined next, ensures that the global minimum of \( f \) exists and is unique. A function \( f : \mathbb{R}^p \to \mathbb{R} \) is \( \mu \)-strongly convex if \( \forall x_1, x_2 \in \mathbb{R}^p \),

\[
f(x_2) \geq f(x_1) + \nabla f(x_1)^\top (x_2 - x_1) + \frac{\mu}{2}\|x_2 - x_1\|^2,
\]

for some \( \mu > 0 \). It can be verified that strong convexity is stronger than \( \mathbb{R} \) in the sense that it further imposes a quadratic lower bound on \( f \). Finally, a function \( f : \mathbb{R}^p \to \mathbb{R} \), not necessarily convex, is \( \ell \)-smooth if for some \( \ell > 0 \) and \( \forall x_1, x_2 \in \mathbb{R}^p \),

\[
\|\nabla f(x_1) - \nabla f(x_2)\|_2 \leq \ell \|x_1 - x_2\|_2,
\]

which implies that

\[
f(x_2) \leq f(x_1) + \nabla f(x_1)^\top (x_2 - x_1) + \frac{\ell}{2}\|x_2 - x_1\|^2.
\]

Clearly, an \( \ell \)-smooth function has a quadratic upper bound. We denote the class of functions that are both \( \mu \)-strongly convex and \( \ell \)-smooth as \( S_{\mu,\ell} \). Note that each \( f \in S_{\mu,\ell} \) is subject to both the lower and upper quadratic bounds in (3) and (5); we thus always have \( \mu \leq \ell \). The ratio \( \kappa := \ell/\mu \) is called the condition number of \( f \) and the functions with large \( \kappa \) are said to be ill-conditioned. A rather simple example of a function in this class \( S_{\mu,\ell} \) is \( f(x) = x^\top Q x + x^\top b + c \), for \( x \in \mathbb{R}^p \) and a positive-definite matrix \( Q \in \mathbb{R}^{p \times p} \). The logistic loss \( f_{i,j} \) described before is convex and therefore the global cost function becomes strongly-convex with \( \lambda > 0 \).

See [2] for more details on convex functions. In this article, the algorithms in question are discussed under the assumption that each local function is smooth and strongly convex, i.e., \( f_i \in S_{\mu_i,\ell_i} \), \( i = 1, \ldots, n \), unless explicitly mentioned otherwise. Thus, the global cost \( F := \frac{1}{n} \sum_{i=1}^n f_i \) is such that \( F \in S_{\mu_F,\ell_F} \) and we denote the unique minimizer of \( F \) as \( x^* \). Where applicable, we refer to the appropriate literature where the convergence results of the related algorithms are generalized to the class of \( \ell \)-smooth but possibly non-convex functions.

2) Communication Graph

We now formally define the mathematical concept of graphs to characterize the communication (information exchange) among the nodes. Consider \( n \) nodes interacting over a potentially directed graph \( G = (\mathcal{V}, \mathcal{E}) \), where \( \mathcal{V} := \{1, \ldots, n\} \) is the set of node indices, and \( \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \) is a collection of ordered pairs \( (i, r) \) such that node \( r \) can send information to node \( i \), i.e., \( i \rightarrow r \). Note that \( G \) is not necessarily undirected, i.e., \( i \leftarrow r \rightarrow i \). We let \( N_i^{\text{in}} \) denote the set of incoming neighbors of node \( i \), i.e., nodes that can send information to node \( i \). Similarly, \( N_i^{\text{out}} \) denotes the set of outgoing neighbors, i.e., nodes that receive information from node \( i \). We assume throughout the paper that \( i \in N_i^{\text{in}} \cap N_i^{\text{out}}, \forall i \in \mathcal{V} \). For an undirected graph, we have that \( N_i = N_i^{\text{in}} = N_i^{\text{out}} \). A directed graph is said to be strongly connected if there exists a directed path between any two nodes. An undirected graph is said to be connected if it is strongly-connected. Given a matrix \( M = \{m_{ir}\} \in \mathbb{R}^{n \times n} \), the directed graph induced by the matrix \( M \) is denoted by \( \mathcal{G}_M = (\{1, 2, \ldots, n\}, \mathcal{E}_M) \), where \( (i, r) \in \mathcal{E}_M \) if and only if \( m_{ir} > 0 \). Conversely, given a directed graph \( G = (\mathcal{V}, \mathcal{E}) \), we say \( W = \{w_{ir} \} \in \mathbb{R}^{n \times n} \) is a weight matrix associated with \( G \) when \( w_{ir} > 0 \) if and only if \( (i, r) \in \mathcal{E} \) and \( w_{ir} = 0 \) otherwise.

3) Nonnegative matrices

A nonnegative (resp. positive) matrix is such that all of its elements are nonnegative (resp. positive). A matrix is row stochastic (resp. column stochastic) if it is nonnegative and all of its rows (resp. columns) sum to one and it is doubly stochastic if it is both row and column stochastic. The spectral radius of row, column, and doubly stochastic matrices is one [2] and one is also an eigenvalue of the corresponding matrix. A nonnegative matrix \( M \) is irreducible if its induced graph \( \mathcal{G}_M \) is strongly connected and \( M \) is further primitive if its trace is positive. For a primitive and row stochastic matrix \( A \), from Perron-Frobenius theorem [10], we denote \( \pi_A \) as its positive left eigenvector corresponding to the eigenvalue 1 such that \( \pi_A^\top 1_n = 1 \). Similarly, for a primitive and column stochastic matrix \( B \), we denote \( \pi_B \) as its positive right eigenvector corresponding to the eigenvalue 1 such that \( 1_n \pi_B = 1 \). Clearly,

\[
A 1_n = 1_n, \quad \pi_A^\top A = \pi_A^\top, \quad B \pi_B = \pi_B, \quad 1_n B = 1_n.
\]

and it can be shown that [102]

\[
A^\infty := \lim_{k \to \infty} A^k = 1_n \pi_A^\top, \quad B^\infty := \lim_{k \to \infty} B^k = \pi_B 1_n
\]

Additional details on these concepts can be found in [102].

1This can be shown by, for example, Gershgorin circle theorem.
2Formally, a nonnegative matrix is said to be primitive if it is irreducible and only has one non-zero eigenvalue of maximum modulus.
III. Decentralized Gradient Descent

One of the classical methods to minimize a differentiable function $F: \mathbb{R}^p \rightarrow \mathbb{R}$ is the gradient descent algorithm $[1]:$

$$x_{k+1} = x_k - \alpha \cdot \nabla F(x_k), \quad \forall k \geq 0,$$

(6)

where $\{x_k\}_{k \geq 0}$ is a sequence of estimates of a minimizer of $F$ and $\alpha > 0$ is a constant step-size. When $F$ is $\ell$-smooth (but not necessarily convex) and $\alpha \in (0, \frac{1}{\ell})$, it can be shown that $\|\nabla F(x_k)\|_2 \rightarrow 0$, i.e., gradient descent finds a critical point of $F$ asymptotically. When $F$ is further $\mu$-strongly convex, i.e., $F \in S_{\mu, \ell}$, as considered in this article, then a critical point $x^*$ is further the unique global minimum of $F$. In this case, we have that $\|x_k - x^*\|_2 \leq \left(1 - \mu \alpha\right)^k \|x_0 - x^*\|_2$ for any $\alpha \in (0, \frac{1}{\ell})$, i.e., gradient descent converges to $x^*$ (on the log-scale rate) $[1]$. In the rest of this article, we discuss various decentralized methods of the gradient descent type that are suitable for many practical problems.

When the gradient descent algorithm is implemented locally at node $i$ without any cooperation to solve the decentralized optimization Problem P1 and P2, we note that (6) only finds the minimizer of $f_i$ at each node $i$, but not the minimizer of the global cost $F = \frac{1}{n} \sum_{i=1}^{n} f_i$ in general. A decentralized method thus must have a means of fusing information over the nodes in the network such that the estimate $x_i^k$, at node $i$, is steered towards the global minimum $x^*$ of $F$. In other words, a decentralized optimization algorithm requires two key ingredients: (i) Agreement: all nodes must agree on the same estimate; and, (ii) Optimality: the agreement must be on the global minimum. Agreement, required by any decentralized optimization algorithm, is typically achieved with the help of average consensus $[27] - [29]$ when the underlying communication graph is undirected, or by push-sum $[43] - [45]$ or surplus consensus $[48]$ when the underlying graph is directed. We next discuss decentralized optimization methods that build directly on top of these consensus algorithms.

A. Decentralized gradient descent: Undirected graphs

We start with the case of undirected and connected graphs where the construction of doubly stochastic (network) weight matrices in a decentralized manner is straightforward; popular methods include Metropolis and Laplacian weights $[25]$. The most well-known and perhaps the simplest optimization algorithm for solving Problem P1 and P2 is Decentralized Gradient Descent (DGD) $[6] - [8]$ described as follows. Let $x_i^k \in \mathbb{R}^p$ denote the estimate of $x^*$ at node $i$ and iteration $k$. DGD recursively runs the following iterations at each node $i$:

$$x_{i,k+1} = \sum_{r \in N_i} w_{ir} x_{r,k} - \alpha \cdot \nabla f_i(x_{i,k}^k), \quad \forall k \geq 0,$$

(7)

where $w_{ir}$ is a weight that node $i$ assigns to each of its neighboring nodes and the weight matrix $W = \{w_{ir}\} \in \mathbb{R}^{n \times n}$ is doubly stochastic. Note that when $\alpha_{k} = 0, \forall k$, DGD reduces to the classical average consensus, where $x_i^k \rightarrow \frac{1}{n} \sum_{i=1}^{n} x_i^0, \forall i$, i.e., all nodes agree on the average of their initial states.

To understand DGD, let us consider its vector-matrix form:

$$x_{k+1} = Wx_k - \alpha \cdot \nabla f(x_k), \quad \forall k \geq 0,$$

(8)

A vector $x^* \in \mathbb{R}^p$ is called a critical point of $F$ if $\nabla F(x^*) = 0_p$.

where $W = W \otimes I_p \in \mathbb{R}^{pn \times pn}$, $x_k \in \mathbb{R}^{pn}$ concatenates the local $x_i^k$’s, and $\nabla f(x_k) \in \mathbb{R}^{pn}$ concatenates the local gradients $\nabla f_i(x_i^k)$’s. Assuming that $\alpha_k \rightarrow 0$, it can be shown under certain conditions that $x_i^k \rightarrow x_k$, where $x_k := \frac{1}{n} \sum_{i=1}^{n} x_i^k$ is the mean iterate over the network. Moreover, multiplying both sides of (8) by $\frac{1}{n} (I_{pn} \otimes I_p)$, we obtain that

$$x_{k+1} = x_k - \alpha \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x_i^k), \quad \forall k \geq 0.$$

Intuitively we have $\frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x_i^k) \rightarrow \nabla F(x_k)$ as $x_i^k \rightarrow x_k$, at each node $i$. Therefore, when $\alpha_k \rightarrow 0$ and the weight matrix $W$ is doubly stochastic, we have that each $x_i^k \rightarrow x_k$ and $x_k$ converges to the minimum of $F = \frac{1}{n} \sum_{i=1}^{n} f_i$ following (6), which guarantees simultaneously the agreement and optimality requirement of decentralized optimization. The rigorous analysis of DGD can be found in, e.g., $[8]$, $[25]$, $[30]$.

We make a few remarks in the following.

Remark 1 (Performance of DGD–Rate/accuracy tradeoffs): DGD converges sublinearly to the exact global minimum $x^*$ of Problem P1 for decaying step-sizes such that $\alpha_k \rightarrow 0$. Under a constant step-size $\alpha$, DGD converges linearly, however, to an inexact solution with accuracy $O(\alpha)$.

It can be shown that a larger constant step-size leads to a faster convergence albeit with worse accuracy; see $[30]$, $[32]$, $[33]$ for details. We will revisit this rate/accuracy tradeoff in Section IV and show that this issue can be fixed by a technique called gradient tracking.

Remark 2 (Consensus + innovation and diffusion learning): DGD-type algorithms of the form (7) are also known as “consensus + innovation” and “diffusion learning” in the context of signal estimation problems especially when the cost functions are quadratic; see $[7]$, $[8]$, $[103] - [105]$ and references therein.

B. Decentralized gradient descent: Directed graphs

We now consider strongly connected, directed graphs where the network weight matrices are either row stochastic or column stochastic but not doubly stochastic, in general. As a consequence, when the weight matrix in (7) is column stochastic but not row stochastic, the nodes do not agree since the right eigenvector corresponding to the eigenvalue of 1 is not $1_n$, that is essential for agreement or consensus $[46]$.

Similarly, when the weights are row stochastic but not column stochastic, the nodes agree however on a sub-optimal solution that is the minimum of a weighted average of local functions (and not the mean). We formally discuss these issues next.

1) DGD with column stochastic weights

Let $B = \{b_{ir}\} \in \mathbb{R}^{n \times n}$ be a primitive, column stochastic network weight matrix such that $1_n^\top B = 1_n$ and $B \pi_B = \pi_B$. Consider DGD (8) with $B$, i.e.,

$$x_{i,k+1} = B x_i - \alpha \cdot \nabla f(x_i), \quad \forall k \geq 0,$$

(10)

where $B = B \otimes I_p$. Recall that $B^k \rightarrow B^\infty := \pi_B 1_n^\top$ and consider the convergence of (10) when $\alpha_{k} = 0, \forall k \geq 0$, i.e.,

$$x_i \rightarrow x_0 = (\pi_B 1_n^\top \otimes I_p) x_0 = (\pi_B \otimes I_p) (1_n^\top \otimes I_p) x_0,$$

which shows that $x_i^k \rightarrow \pi_B (\sum_{r=1}^{p} x_r^0, \forall i$, in other words, when the weights are only column stochastic, the nodes do not agree because of the non-identical elements in $\pi_B$. We observe however that the scaled iterates $x_i^k / \pi_B (\sum_{r=1}^{p} x_r^0, \forall i, converge to the
and average of the initial states of the nodes, i.e., \( x_i^n/t(n) = 1/n \sum_{r=1}^{n} x_i^r, \forall i \). Since no node in the network has the knowledge of the eigenvector \( \pi_B \), another set of iterations is required to asymptotically estimate \( x_i^n/t(n) \) at each node \( i \). The resulting algorithm is known as Gradient-Push \([37], [38]\), which runs the following iterations at each node \( i \):

\[
\begin{align*}
   z_k^{i+1} &= \sum_{r \in N_i^n} b_{ir}z_r^k, \\
   x_k^{i+1} &= \sum_{r \in N_i^n} b_{ir}x_r^k - \alpha_k \cdot \nabla f_i(x_k^r), \\
   w_k^{i+1} &= x_k^r + 1 \sum_{i=1}^n \pi_A^r, (11a)
\end{align*}
\]

which runs the following iterations at each node \( i \):

\[
\begin{align*}
   z_k^{i+1} &= \sum_{r \in N_i^n} b_{ir}z_r^k, \\
   x_k^{i+1} &= \sum_{r \in N_i^n} b_{ir}x_r^k - \alpha_k \cdot \nabla f_i(w_k^r), \\
   w_k^{i+1} &= x_k^r + 1 \sum_{i=1}^n \pi_A^r. (11b)
\end{align*}
\]

where \( x_i^0 = w_i^0 \in \mathbb{R}^p \) is arbitrarily chosen and \( z_i^0 = 1, \forall i \). It is straightforward to verify that \( z_k^i \to n[\pi_B], \forall i \), and thus if \( \alpha_k \to 0 \), intuitively \( w_i^k \) achieves agreement. On the other hand, the column stochasticity of \( B \) guarantees the optimality of the Gradient-Push as discussed earlier for \( N \).

2) DGD with row stochastic weights

Consider DGD now with row stochastic weights \( A = \{a_{ir}\} \):

\[
x_k^{i+1} = Ax_k - \alpha_k \cdot \nabla f(x_k), (12)
\]

where \( A = A \otimes I_p \). We have that \( A1_n = 1_n \) and \( \pi_A^r A = \pi_A^r \), where \( \pi_A > 0 \) and \( A \to A^\infty = 1_n \pi_A^r \). In other words, the nodes are able to achieve an agreement under a row stochastic network weight matrix. In particular, without gradient corrections, it is straightforward to show that \( x_i^k \to \sum_{r=1}^{n} [\pi_A]^r x_i^r \), at each node \( i \). In \( (12) \), suppose that \( \alpha_k \to 0 \), then we have that \( x_i^k \to \sum_{r=1}^{n} [\pi_A]^r x_i^r := \bar{x}_k, \forall i \). Multiplying both sides of \( (12) \) by \( (\pi_A^r \otimes I_p) \), we obtain that

\[
\bar{x}_{k+1} = \bar{x}_k - \alpha_k \cdot \sum_{r=1}^{n} \pi_A^r \nabla f_i(x_i^r). (13)
\]

Based on the above discussion, we conclude that each node approaches \( \bar{x}_k \), which converges to the minimum of a weighted average of the local costs \( f_i \). The decentralized algorithm to find the minimum of the global cost \( F \) may be constructed by dividing \( \nabla f_i \) by \( [\pi_A]^r \). Similar to Gradient-Push, separate iterations however are required to estimate the eigenvector \( \pi_A \) since it is not locally known at any node. The resulting algorithm \([106]\), termed as DGD-RS, is given by

\[
\begin{align*}
   x_k^{i+1} &= \sum_{r \in N_i^n} a_{ir}x_r^k - \alpha_k \cdot \nabla f_i(x_k^r), \\
   \varepsilon_k^{i+1} &= \sum_{r \in N_i^n} a_{ir} \varepsilon_r^k. (14a)
\end{align*}
\]

Remainder (Column and row stochastic weights over directed graphs): A column stochastic weight matrix \( B = \{b_{ir}\} \) is often constructed as \( b_{ir} = 1/|N_i^{out}|, \forall r \in N_i^{out} \). This formulation requires each node to know its out degree, i.e., \( |N_i^{out}| \). A row stochastic weight matrix \( A = \{a_{ir}\} \), on the other hand, can be easily constructed as \( a_{ir} = 1/|N_i^{out}|, \forall r \in N_i^{out}, \forall i \), since each node can locally assign weights to the information it receives.

Remark 4 (Average consensus over directed graphs): It can be easily verified that \( (11a) - (11c) \) recover the average of the initial conditions \( x_i^0 \)’s when \( \alpha_k = 0, \forall k \). This algorithm is well-known as push-sum that implements average consensus with the help of column stochastic weights \([43], [44]\).

Remark 5 (Eigenvector estimation): Based on the current discussion, we note that over directed graphs there is a certain imbalance that is caused by not having \( 1_n \) as either the left or the right eigenvector corresponding to the eigenvalue 1 of the corresponding weight matrices. When the weights are column stochastic, this imbalance manifests itself in disagreement of the estimates, whereas in the row stochastic case, this imbalance causes convergence to the minimum of a weighted sum of local cost functions, in stead of \( F \). To overcome this imbalance, a division by the appropriate eigenvector elements is required that leads to separate iterations for eigenvector estimation. The eigenvector estimation, an iterative procedure in itself, may slow down the convergence of the corresponding algorithms especially when the graphs are not well-connected.

IV. THE AB/PUSH-PULL FRAMEWORK

All three decentralized algorithms discussed in the previous section, with doubly stochastic, column stochastic, and row stochastic weights respectively, converge sub-linearly to the minimizer \( x^* \) with decaying step-sizes, i.e., with \( \alpha_k \to 0 \). With a constant step-size, these methods converge linearly albeit to a sub-optimal solution. The reason for this sub-optimality is that \( x^* \) is not a fixed point when the step-size is a constant, as we explain in the following. Consider \([7]\) with a constant step-size \( \alpha \) and with \( x_k^i = x^*, \forall i \) and for some \( k \), then

\[
x_k^{i+1} = \sum_{r \in N_i^n} a_{ir}x^r - \alpha \cdot \nabla f_i(x^*), (14b)
\]

because \( \nabla f_i(x^*) \neq 0, \forall i \) in general, i.e., the minimizer \( x^* \) of the global cost function \( F \) does not necessarily minimize the local functions \( f_i^r \)'s. Clearly, this issue disappears if \( \nabla f_i \) at each node is replaced by \( \nabla F \) but that is not possible since \( f_i^r \)'s are distributed and private.

A recently introduced scheme that is referred to as gradient tracking overcomes the aforementioned steady-state error, while keeping a constant step-size, by replacing \( \nabla f_i \) with an iterative tracker of the global gradient \( \nabla F \) at each node \( i \) \([10], [12], [13], [16], [40], [107]\). Formally, a new variable \( y_i^k \in \mathbb{R}^p \) is added locally at each node \( i \) to track \( 1/n \sum_{r=1}^{n} \nabla f_i(x_k^r) \), and the estimate \( x_k^i \) of \( x^* \) descends in the direction of \( -y_i^k \) in stead of \( \nabla f_i \) as in the DGD framework. The task is thus to design an algorithm that tracks a time-varying function \( 1/n \sum_{r=1}^{n} \nabla f_i(x_k^r) \) whose components are distributed over the network. To this aim, Dynamic Average Consensus \([9]\) is employed that is able to track the sum \( \sum_{i=1}^{n} y_i^k \) of time-varying functions \( y_i^k \), assuming that each \( r_i^k \) approaches a constant.
The resulting algorithm, Decentralized Gradient Descent with Gradient Tracking (GT-DGD), linearly converges to the global minimum $x^\ast$ with a constant step-size $\alpha$ for smooth and strongly convex problems, i.e., when each $f_i \in \mathcal{S}_{\mu,i}$. The update of GT-DGD is given by the following iterations: at each node $i$ and $k\geq 0$,

$$x_{k+1} = \sum_{r \in N_i} w_{ir} x^i_k - \alpha \cdot y^i_k, \quad (15a)$$

$$y_{k+1} = \sum_{r \in N_i} w_{ir} y^i_k + \nabla f_i(x^i_{k+1}) - \nabla f_i(x^i_k), \quad (15b)$$

where $x^i_k$'s in $\mathbb{R}^p$ are arbitrary and $y^i_k = \nabla f_i(x^i_k)$, $\forall i$. Here, similar to DGD, the network weights $\{w_{ir}\}$ are chosen such that $W = \{w_{ir}\}$ is primitive and doubly stochastic. Intuitively, if $x^i_k \rightarrow \hat{x}_i$ at each node $i$, then $\sum_{i=1}^n \nabla f_i(x^i_k)$ approaches the global gradient $\nabla F(x_k)$ and we have that $y^i_k \rightarrow \nabla F(x_k)$ at each node $i$. As a consequence, the iterates obtained via equation (15a) asymptotically descends in the global direction $\nabla F$. However, GT-DGD is restricted to undirected (or balanced) graphs because the weights $W$ are required to be doubly stochastic. Recall that the row stochasticity of $W$ leads to agreement, while the column stochasticity leads to optimality. Although row stochasticity and column stochasticity are both required, we may ask whether they must hold simultaneously for the weight matrices in (15a) and (15b).

In particular, (15a) may be implemented with row stochastic weights (that are not necessarily column stochastic) leading to an agreement among the nodes, while (15b) may be implemented with column stochastic weights (that are not necessarily row stochastic) ensuring optimality. This observation leads to the AB/Push-Pull algorithm described formally in Algorithm 1 with the help of row and column stochastic weights, $A = \{a_{ir}\}$ and $B = \{b_{ir}\}$, respectively. Since the implementation of AB/Push-Pull does not require doubly stochastic weights, the algorithm is applicable to both directed and undirected graphs. When each $f_i \in \mathcal{S}_{\mu,i}$, AB/Push-Pull converges linearly to the global minimum $x^\ast$ of $F$. [4], [5].

**Algorithm 1** AB/Push-Pull: At each node $i$

| Require: $x^i_0 \in \mathbb{R}^p$, $y^i_0 = \nabla f_i(x^i_0)$, $\alpha > 0$. |
|---|
| 1: for $k = 0, 1, 2, \ldots$ do |
| 2: State update: $x^i_{k+1} = \sum_{r \in N_i} a_{ir} x^i_r - \alpha \cdot y^i_k$ |
| 3: Gradient tracking update: $y^i_{k+1} = \sum_{r \in N_i} b_{ir} y^i_r + \nabla f_i(x^i_{k+1}) - \nabla f_i(x^i_k)$ |
| 4: end for |

**Remark 6 (Uncoordinated Step-sizes):** The AB/Push-Pull algorithm is applicable to the case when each node $i$ chooses a constant but distinct step-size $\alpha_i$. The linear convergence of AB/Push-Pull can be established when the maximum step-size $\max \alpha_i$ across all nodes is positive and sufficiently small, while all other step-sizes can in fact be chosen as 0; see [42], [108], [109] for detailed discussion. For the sake of the argument, let us assume that node 1 in a network of $n$ nodes chooses an appropriate step-size $\alpha_1 > 0$, while $\alpha_i = 0$.

4Algorithm [4] was proposed parallelly as AB in [4] and as push-pull in [5]; see Section IV-B for push and pull aspects of the underlying communication.
(i) the agreement error \( \|x_k - A^\infty x_k\|_{\pi_A \otimes 1_p} \); (ii) the optimality gap \( A^\infty x_k - 1_n \otimes x^* \|_2 \); and, (iii) the gradient tracking error \( \|y_k - B^\infty y_k\|_{\pi_B \otimes 1_p} \).

These errors can be written in a vector

\[
t_k = \begin{bmatrix} \|x_k - A^\infty x_k\|_{\pi_A \otimes 1_p} \\ \|A^\infty x_k - 1_n \otimes x^*\|_2 \\ \|y_k - B^\infty y_k\|_{\pi_B \otimes 1_p} \end{bmatrix}
\]

that follows:

\[
t_{k+1} \leq J_\alpha t_k,
\]

where

\[
J_\alpha = \begin{bmatrix} \sigma_A & 0 & 0 \\ 0 & 0 & 0 \\ a_0 & 0 & \sigma_B \end{bmatrix} + \alpha \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & -n\mu(\pi_A^\top \pi_B) & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix},
\]

for some nonnegative constants \( a_i \)'s; see \([14], [15] \) for details.

**Step 3–Linear convergence:** Clearly, if \( t_k \to 0_3 \), then \( x_i \)'s eventually reach agreement and the agreement is on the global minimum \( x^* \) of \( F \), while the rate at which \( t_k \to 0_3 \) provides an upper bound on the convergence rate of \textbf{AB/Push-Pull}.

Given that \( t_k \) follows the LTI system in \([15] \), linear convergence of \textbf{AB/Push-Pull} to \( x^* \) can be established by showing that the spectral radius \( \rho(J_\alpha) \) of \( J_\alpha \) is less than one. There are several ways to obtain \( \rho(J_\alpha) < 1 \) for the matrix \( J_\alpha \) given above and we present the argument from \([14] \) for simplicity; see also \([70], [108] \). First note that when \( \alpha = 0 \), \( J_\alpha \) has three eigenvalues: \( \sigma_A, \sigma_B, 1 \), two of which are strictly less than 1 as given by Step 1. Next, since the eigenvalues of a matrix are continuous functions of its elements, it can be shown that as the step-size \( \alpha \) slightly increases from 0, the eigenvalue of 1 strictly decreases. In particular, denote by \( q(\alpha) \) the eigenvalues of \( J_\alpha \) as a function of \( \alpha \), we have that

\[
\left. \frac{d q(\alpha)}{d \alpha} \right|_{\alpha=0,q=1} = -n\mu(\pi_A^\top \pi_B),
\]

which is strictly negative because \( \pi_A \) and \( \pi_B \) are both strictly positive, where \( n \) is the number of nodes and \( \mu > 0 \) is the strong convexity constant of \( F \). In other words, we have that \( \rho(J_\alpha) < 1 \) for a sufficiently small \( \alpha > 0 \).

**B. Architectures**

The **AB/Push-Pull** algorithm unifies different types of communication architectures, including decentralized, centralized, and semi-centralized architecture \([15], [108] \).

To illustrate, let graphs \( G_A \) and \( G_B \) be induced by the two matrices \( A \) and \( B \), respectively, i.e., \( a_{ir} > 0 \) (resp. \( b_{ir} > 0 \)) if and only if there exists a link from node \( r \) to node \( i \) in graph \( G_A \) (repos. \( G_B \)). Note that when implementing the **AB/Push-Pull** algorithm, we have the flexibility to design two different graphs \( G_A \) and \( G_B \); rather than restricting to using one single graph that is commonly considered in the decentralized optimization literature (see e.g. \([13] \)). Such flexibility is key to the unifying property of the **AB/Push-Pull** algorithm. Formally, we proceed the discussion with the following condition on the graphs \( G_A \) and \( G_B^\top \) (the graph \( G_B^\top \) is the graph \( G_B \) with all its edges reversed); the graphs \( G_A \) and \( G_B^\top \) each contain at least one spanning tree; moreover, there exists at least one node that is a root of spanning trees for both \( G_A \) and \( G_B^\top \).

In light of the above conditions on the graphs, we explain how **AB/Push-Pull** unifies different types of communication architectures using examples from \([15] \). First, for the fully decentralized architecture, suppose we have a directed and strongly connected graph \( G \). We can let \( G_A = G_B = G \) and design the weights for \( A \) and \( B \) accordingly. Then we have a typical peer-to-peer network structure which is fully decentralized. For the case of (semi)-centralized architectures, which is less straightforward, we consider a simple star network \( G \) of four nodes, as shown in Fig. 2. Clearly \( G_A \) and \( G_B^\top \) are identical spanning tree with node 1 being their common root. Under this setting, the weight matrices \( A \) and \( B \) in **AB/Push-Pull** can be chosen as

\[
A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0.5 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0.5 & 0.5 & 0.5 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix}.
\]

It can be seen from Fig. 2 that the central node 1’s information regarding \( x_k \) is pulled by the entire network through \( G_A^\top \); the other nodes only passively receive and use node 1’s information. Meanwhile, node 1 has been pushed information regarding \( y_k \) (\( i = 2, 3, 4 \)) from nodes 2, 3, and 4 through \( G_B^\top \); the other nodes only actively comply with the request from node 1. This motivates the algorithm’s name **Push-Pull** in \([15], [108] \). Although nodes 2, 3, and 4 are updating \( y_k \)'s accordingly, they do not have to contribute to the optimization process: due to the weights in the last three rows of \( B \), the values of \( y_k \)'s for nodes 2, 3, and 4 will decrease to 0 geometrically fast. Without loss of generality, suppose \( f_1(x) = 0, \forall x \in \mathbb{R}^5 \), and the local stepsize \( \alpha \) for nodes 2, 3, and 4 are set to 0. Then the **AB/Push-Pull** algorithm represents a typical centralized gradient method 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 fashion.

Taking the above as a toy example for illustrating the centralized architecture, note that node 1 can be replaced by a strongly connected subnet in \( G_A \) and \( G_B^\top \), respectively. Similarly, all the other nodes can be replaced by subnets as long as the information from the master layer in these subnets can be diffused to all the slave layer nodes in \( G_A \) and the information from all the slave layer nodes can be diffused to the master layer in \( G_B^\top \). The concept of rooted trees can be used to understand the specific requirements on connectivities of slave subnets. In general, the nodes are referred to as leaders

\footnote{This condition is weaker than assuming both \( G_A \) and \( G_B \) are strongly connected, while still imposing certain restrictions on the graph topology. Note that the union of \( G_A \) and \( G_B \) is still guaranteed to be strongly connected, which ensures the essential information exchange among the nodes.}
if their roles in the network are similar to the role of node 1 in Fig. 2 and the other nodes are referred to as followers. After replacing the individual nodes by subnets, all the subnets have decentralized network structures, while the leader subnet and the follower subnets form a master-slave relationship. This is why such an architecture is called semi-centralized.

C. Acceleration

It is natural to consider adding momentum to accelerate the convergence of AB/Push-Pull. One immediate extension, based on the Polyak’s heavy-ball method [1], is ABm [109] that is given by the following equations: 
\[ \forall k \geq 1, \]
\[ x_{k+1} = \sum_{r \in N_i^a} a_{ir} x_r^k - \alpha \cdot y_r^k + \beta \cdot (x_i^k - x_{i-1}^k), \] (19a)
\[ y_{k+1} = \sum_{r \in N_i^a} b_{ir} y_r^k + \nabla f_i(x_{k+1}^i) - \nabla f_i(x_i^k), \] (19b)
where \( x_i^k - x_{i-1}^k \) is the heavy-ball momentum added at each node, \( \alpha \) and \( \beta \) are the step-size and momentum parameters, respectively. It is shown in [109] that ABm converges linearly for sufficiently small step-size and momentum parameters, however, acceleration over AB is only shown numerically and a detailed theoretical analysis remains an open problem. Alternatively, the use of Nesterov’s momentum [2] is also studied in the context of AB/Push-Pull. The corresponding algorithm ABN [112] is given by the following equations:
\[ s_{k+1}^i = \sum_{r \in N_i^a} a_{ir} x_r^k - \alpha \cdot y_r^k, \] (20a)
\[ x_{k+1}^i = s_{k+1}^i + \beta_k \cdot (s_{k+1}^i - s_k^i), \] (20b)
\[ y_{k+1}^i = \sum_{r \in N_i^a} b_{ir} y_r^k + \nabla f_i(x_{k+1}^i) - \nabla f_i(x_i^k), \] (20c)
where \( \beta_k \) is the momentum parameter. When the set of weights above are both doubly stochastic, a closely related form of ABN is studied in [113] where acceleration is analytically shown. The analysis of ABN however remains an open problem and acceleration is only shown numerically in [112].

V. AB/Push-Pull: A GENERAL FRAMEWORK

We now cast AB/Push-Pull as a general framework that unifies much of the existing work on decentralized first-order methods with gradient tracking. Clearly, GT-DGD [15a–15b] is a special case of AB/Push-Pull when both sets of weights are further doubly stochastic. We note that DGD also follows from AB/Push-Pull [16–17], when \( A \) is replaced by a doubly stochastic matrix \( W \) and \( B \) is chosen as identity \( I_{np} \). Below, we relate AB/Push-Pull to well-known algorithms in the literature; see [109] for additional discussion.

A. GT-DGD with column stochastic weights

We recall AB/Push-Pull written compactly in [16–17], where \( A = \Pi_A \otimes I_p \) is row stochastic, \( B = \tilde{B} \otimes I_p \) is column stochastic, while both \( \Pi_A \) and \( \tilde{B} \) are in addition primitive. Define \( \Pi_A \) to be a diagonal matrix with \( \Pi_A \otimes I_p \) on its main diagonal, where \( \Pi_A \otimes I_p = \Pi_A \) and \( I_p \) are in addition primitive. Define \( \Pi_A \) to be a diagonal matrix with \( \Pi_A \otimes I_p \) on its main diagonal, where \( \Pi_A \otimes I_p = \Pi_A \) and \( I_p \) are in addition primitive. Clearly, we have \( \Pi_A = \text{diag}(\Pi_A) \otimes I_p \) and note that \( \Pi_A \Pi_A^{-1} = \text{column stochastic}, i.e., \)
\[ \Pi_{np}^\top \omega \Pi_A \Pi_A^{-1} = \Pi_{np}^\top \text{diag}(\Pi_A) \otimes I_p \Pi_A \Pi_A^{-1} = \Pi_{np}^\top. \]
Similarly, we have \( \Pi_A \Pi_A^{-1} = (\Pi_A \otimes I_p) \Pi_A^{-1} = \Pi_A \otimes I_p. \) With the help of the invertible matrix \( \Pi_A \), we now define a state transformation \( \tilde{x}_k = \Pi_A x_k \) and modify (16)–(17) to obtain
\[ \tilde{x}_{k+1} = \tilde{B} \tilde{x}_k - \alpha \Pi_A \cdot y_k, \] (21)
\[ y_{k+1} = B y_k + \nabla f(\Pi_A^{-1} \tilde{x}_{k+1} - \tilde{f}(\Pi_A^{-1} \tilde{x}_k), \] (22)
where \( \tilde{B} := \Pi_A \Pi_A^{-1} \) is column stochastic as we noted earlier. This state transformation shows that a decentralized optimization algorithm with only column stochastic weights (\( B \) and \( \tilde{B} \)) is naturally embedded in AB/Push-Pull. However, such an algorithm additionally requires the eigenvector \( \Pi_A \) of the weights \( B \) in [21] to implement (22). Since \( \Pi_A \) is not locally known to any node, implementing (21) and (22) require estimating this non-I-r eigenvector of the column stochastic \( \tilde{B} \); not required in AB/Push-Pull because of the row stochasticity of \( A \). The algorithm that follows is given by, at each node \( i \):
\[ \tilde{x}_{k+1}^i = \sum_{r \in N_i^a} \tilde{b}_{ir} \tilde{x}_r^k - \alpha \cdot y_r^k, \] (23a)
\[ z_{k+1}^i = \sum_{r \in N_i^a} \tilde{b}_{ir} z_r^k, \] (23b)
\[ \tilde{y}_{k+1}^i = \sum_{r \in N_i^a} b_{ir} \tilde{y}_r^k + \nabla f_i(\tilde{x}_{k+1}^i) - \nabla f_i(\tilde{x}_i^k), \] (23c)
where \( \{\tilde{b}_{ir}\} \) and \( \{b_{ir}\} \) are (possibly different) column stochastic weights. Recall from Section III-B that the iterate \( z_k^i \) estimates the right eigenvector \( \Pi_A \) of \( \tilde{B} = \{\tilde{b}_{ir}\} \). The above algorithm essentially adds gradient tracking to gradient-push described earlier in [11a–11c].

Remark 7: The algorithm in (23a–23c) is well-known as ADDDOPT [40] and Push-Diging [13], both of which add push-sum consensus to GT-DGD implemented over directed graphs with column stochastic weights. We note that the general idea of push-sum consensus explored over the GT-DGD framework as in ADDDOPT/Push-Diging [13, 40], essentially results from a state transformation in AB/Push-Pull.

B. GT-DGD with row stochastic weights

Following the previous discussion, we now perform a state transformation on the \( y_k \)-update such that the column stochastic weight matrix \( B \) in AB/Push-Pull transforms into a row stochastic matrix. Let \( \Pi_B := \text{diag}(\Pi_B) \otimes I_p \), it is easy to verify that \( \tilde{A} := \Pi_B^\top B \Pi_B = \{\bar{a}_{ir}\} \) is row stochastic, i.e., \( \Pi_B^\top B \Pi_B 1_{np} = 1_{np}, \) \( (\Pi_B^\top \otimes I_p) \Pi_B^\top B \Pi_B = \Pi_B^\top \otimes I_p. \) The applicable state transformation \( \tilde{y}_k := \Pi_B^\top y_k \) leads to
\[ \tilde{x}_{k+1}^i = \sum_{r \in N_i^a} a_{ir} \tilde{x}_r^k - \alpha \cdot \tilde{y}_r^k, \] (24a)
\[ \tilde{e}_{k+1}^i = \sum_{r \in N_i^a} \bar{a}_{ir} \tilde{e}_r^k, \] (24b)
\[ \tilde{y}_{k+1}^i = \sum_{r \in N_i^a} \bar{a}_{ir} \tilde{y}_r^k + \nabla f_i(\tilde{x}_{k+1}^i) - \nabla f_i(\tilde{x}_i^k), \] (24c)

\[ 1_{np}^\top \omega \Pi_A \Pi_A^{-1} = 1_{np}^\top (\text{diag}(\Pi_A) \otimes I_p) (\Pi_A \otimes I_p) \Pi_A^{-1} = 1_{np}^\top. \]
after adding eigenvector estimation to estimate the non-1, eigenvector vector of $B$ corresponding to the eigenvalue 1, where $[e^i_k]_i$ is the $i$th element of $e^i_k \in \mathbb{R}^n$ at node $i$. The algorithm is initialized with an arbitrary $x^0_i \in \mathbb{R}^p$ and $y^0_i = \nabla f_i(x^0_i)$, while $e^i_0 \in \mathbb{R}^n$ is a vector of zeros with a one at the $i$th location. The resulting algorithm is formally studied in [41], [42] as FROST and is a gradient tracking extension of DGD with row stochastic weights [106], described earlier in (14a)–(14b).

C. Average consensus

We now interpret AB/Push-Pull only for consensus problems and show that it subsumes average consensus over strongly connected graphs as a special case. To show this, we choose the cost function at each node $i$ as

$$f_i(x) = \frac{1}{2}\|x - v_i\|^2_2,$$  \hspace{1cm} (25)

for some $v_i \in \mathbb{R}^p$. Clearly, the minimum of $F = \sum_{i=1}^n f_i$ is achieved at $x^* = \frac{1}{n}\sum_{i=1}^n v_i$. Thus, AB/Push-Pull naturally leads to the following average consensus algorithm by noting that $\nabla F(x_{k+1}) = \nabla F(x_k) = x_{k+1} - x_k$ for the local functions described in (25).

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} A & -\alpha I_{np} \\ A - I_{np} & B - \alpha I_{np} \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix},$$ \hspace{1cm} (26)

where $x^i_0 = v_i$ and $y^i_0 = 0$, $\forall i$. From the linear convergence of AB/Push-Pull, as shown in Section V-A it follows that the above equations converge linearly to the average of $v_i$'s. What is surprising is that the above algorithm is closely related to a well-known average consensus over directed graphs known as surplus consensus [48]. In particular, surplus consensus is obtained after a state transformation of (25) with diag $[I_{np}, -I_{np}]$; see [109] for more details.

Remark 8 (Consensus over arbitrary graphs): Following this discussion, choosing the local functions $f_i$'s as given by (25) in GT-DGD [10], [12], or in ADDOPT/Push-DIGing [13], [40], or in FROST [41], [42], leads to average consensus, that utilizes gradient tracking, with only doubly stochastic, column stochastic, or row stochastic weights. The protocol that results directly from AB/Push-Pull is surplus consensus, while the one resulting from FROST has been considered in [45]. Following the state transformations of Sections V-A and V-B it can be verified that the algorithm in [45] is in fact related to surplus consensus after a state transformation.

Remark 9: Recall that DGD [7] adds a gradient correction to consensus with doubly stochastic weights and is applicable to undirected graphs. Its extension, GP [11a], [11c], to directed graphs adds gradient correction to push-sum consensus [38], [114]. Similarly, we may use (26), or equivalently surplus consensus [48], as the consensus layer and add a gradient correction to enable decentralized optimization over directed graphs. This idea was explored in [46], [47], however, the convergence rate restrictions are similar to that of DGD.

VI. DECENTRALIZED STOCHASTIC OPTIMIZATION

We now discuss stochastic first-order methods in Section VI and VII for Problem P2. The discussion in Section VI in particular, can be extended to the more general case of online stochastic optimization where a noisy gradient is sampled in real-time; see [115] for smooth and strongly convex problems and [20] for non-convex problems.

Recall from Problem P2 that each node $i$ possesses $m_i$ data samples leading to $m_i$ component costs $\{f_{i,j}\}_{j=1}^{m_i}$ that constitute the local cost $f_i$. For such problems, the methods described so far in this article are batch operations, i.e., they compute the gradient $\nabla f_i$ of the full local cost at each node and thus the descent direction $\sum_{j=1}^{m_i} \nabla f_{i,j}$ is to be computed over all local data samples $f_{i,j}$'s. Computing $m_i$ gradients at each node $i$ can be taxing particularly when the data is high-dimensional and the cost functions are non-trivial. Computationally-efficient schemes thus rely on stochastic policies to randomly select a small subset of data from the local batch, over which the descent direction is computed. The centralized algorithm based on this idea is the well-known Stochastic Gradient Descent (SGD); see [3] for a comprehensive review. Assuming that the data is not distributed ($n = 1$), and the goal is to minimize $F = \frac{1}{m} \sum_{j=1}^{m} f_j$, the centralized SGD, in its simplest form, is given by

$$x_{k+1} = x_k - \alpha_k \cdot \nabla f_{\tau_k}(x_k), \hspace{1cm} \forall k \geq 0,$$ \hspace{1cm} (27)

where $\tau_k$ is an index chosen uniformly at random from the set $\{1, \ldots, m\}$ at each $k$.

It can be verified that $\nabla f_{\tau_k}(x_k)$ is an unbiased estimate of the full gradient, i.e., $E[\nabla f_{\tau_k}(x_k)|x_k] = \nabla F(x_k)$. Moreover, it is often assumed that each stochastic gradient $\nabla f_{\tau_k}(x_k)$ has uniformly bounded variance, i.e.,

$$E [\|\nabla f_{\tau_k}(x_k) - \nabla F(x_k)\|^2_2 | x_k] \leq \sigma^2, \hspace{1cm} \forall k \geq 0,$$

where $\sigma > 0$ is some universal constant. Under an additional assumption that $F \in \mathcal{S}_{\ell,\tau}$, it can be shown that with a constant step-size $\alpha \in (0, \frac{1}{\ell}]$, the mean squared error $E [\|x_k - x^*\|^2_2]$ decays linearly, at the rate of $(1 - \mu\alpha)^k$, to a neighborhood of $x^*$. Formally, we have [3].

$$E [\|x_k - x^*\|^2_2] \leq (1 - \mu\alpha)^k + \frac{\sigma^2}{\mu}, \hspace{1cm} \forall k \geq 0.$$ \hspace{1cm} (28)

The constant steady-state error given by $\frac{\sigma^2}{\mu}$ implies the “inexact convergence” of SGD due to the persistent gradient noise, i.e., the variance $\sigma^2$ does not vanish. Diminishing step-sizes overcome this issue and leads to exact convergence albeit at a slower rate. For example, with $\alpha_k = \frac{1}{\mu(k+1)}$, we have $\forall k \geq 0,$

$$E [\|x_k - x^*\|^2_2] \leq \frac{1}{k+1} \max \left\{ \frac{2\sigma^2}{\mu^2}, \|x_0 - x^*\|^2_2 \right\}.$$ \hspace{1cm} (29)

In other words, to reach an $\epsilon$-accurate solution of $x^*$, i.e., $E [\|x_k - x^*\|^2_2] \leq \epsilon$, the SGD method with decaying step-sizes requires $O(\kappa\epsilon^{-1})$ component gradient evaluations, where $\kappa = \ell/\mu$. Note that this rate is independent of the sample size since SGD only computes one component gradient (or a subset of component gradients) per iteration. In the following, we discuss the decentralized versions of SGD.

Remark 10 (GD vs. SGD): We note that gradient descent (GD) [6] computes a full gradient, i.e., $m$ component gradients per iteration, while SGD [27] computes only one component gradient (or a subset of component gradients). Consequently,
SGD requires $O(kc^{-1})$ component gradient evaluations to reach an $\epsilon$-accurate solution of the global minimum $x^*$, independent of the size $m$ of the data, versus $O(mk \ln c^{-1})$ required by GD that uses the entire batch. It can be argued that SGD is often more preferable in the big data regimes where $m$ is very large and low-precision solutions may suffice [3]. Moreover, it is worth noting that SGD is applicable in the more general setting of online, streaming data; see Remark [14].

A. Decentralized Stochastic Gradient Descent (DSGD): Undirected Graphs

We now go back to Problem P1 where the data is distributed over a network of $n > 1$ nodes and each node $i$ has access to $m_i$ component functions indexed by $j$. Similar to SGD, each node $i$ samples from its local full batch and implements the following protocol known as DSGD [7], [8], [9]:

$$x_{k+1}^i = \sum_{r \in N_i^m} w_{ir} x_k^r - \alpha_k \cdot \nabla f_{i,r}(x_k^i), \quad \forall k \geq 0, \quad (30)$$

where $\tau_k^r$ is an index chosen uniformly at random from the local index set $\{1, \ldots, m_i\}$, and the weight matrix $W = \{w_{ir}\}$ is doubly stochastic. Since the weight matrix is required to be doubly stochastic, the underlying graphs are restricted to be undirected or weight-balanced directed in general.

As in the centralized case, it is straightforward to verify that $E[\nabla f_{i,r}(x_k^r) \cdot \nabla f_i(x_k^i)] = 0$, $\forall i$, and we assume that the variance of each stochastic gradient is uniformly bounded, i.e.,

$$E\left[\left\|\nabla f_{i,r}(x_k^r) - \nabla f_i(x_k^i)\right\|^2 x_k^i\right] \leq \sigma^2, \quad \forall i, k \geq 0.$$

Assuming that each local function is smooth and strongly convex, i.e., $f_i \in S_{\mu,i}, \forall i$, and that $\lambda$ is the second largest singular value of $W$, it is shown in [54] that under a constant step-size, $\alpha_k = \alpha \in (0, O(\frac{1}{\sqrt{\lambda}}))$, $\forall k$, the mean squared error $E[\|x_k^i - x^i\|^2]$ decays linearly, at the rate of $(1 - O(\mu\alpha))^k$ up to a steady state error such that

$$\lim_{k \to \infty} \sup_{n} \frac{1}{n} \sum_{i=1}^{n} E\left[\left\|x_k^i - x^i\right\|^2\right] = O \left( \frac{\alpha \sigma^2}{\mu} + \frac{\alpha^2 \kappa^2 \sigma^2}{1 - \lambda} + \frac{\alpha^2 \kappa b}{(1 - \lambda)^2} \right), \quad (31)$$

where $b := \frac{1}{n} \sum_{i=1}^{n} \left\|\nabla f_i (x^i)\right\|^2$. With a decaying step-size such that $\alpha_k = O(1/k)$, DSGD achieves an exact convergence [93], [116]–[119], such that

$$\sup_{n \geq 1} \frac{1}{n} \sum_{i=1}^{n} E\left[\left\|x_k^i - x^i\right\|^2\right] = O \left( \frac{1}{k} \right), \quad \forall k \geq 0. \quad (32)$$

See [118] for explicit non-asymptotic characterization of the mean-squared rate of DSGD under decaying step-sizes and [18] for smooth non-convex problems.

Remark 11 (SGD vs. DSGD): With decaying step-sizes $\alpha_k = O(1/k)$, DSGD achieves a network-independent convergence rate asymptotically that matches the centralized minibatch SGD (that samples $n$ stochastic gradients per iteration); the underlying network topology, in fact, only affects the number of transient iterations required to reach this asymptotic rate (see [118], [119] for precise statements). With an appropriate constant step-size, DSGD achieves linear convergence but to a steady-state error [31]. This steady-state error depends on two additional bias terms when compared with that of centralized SGD [28]. Both terms arise from the decentralized nature of DSGD and depend on the factor $(1 - \lambda)$ that encodes the connectivity of the underlying graph. However, the last bias term further includes the constant $b$ that captures the average deviation between the local and global solutions. In particular, assume $x_k^i$ to be the locally optimal solution such that $\nabla f_i(x_k^i) = 0_i, \forall i$. We have $b \leq \frac{1}{n} \sum_{i=1}^{n} \|x^i - x_k^i\|^2$. Clearly, $b$ is large when data distributions across the nodes are heterogeneous and the local optimal is far from the global optimal. This discussion motivates the use of gradient tracking technique in DSGD as we discuss next.

B. DSGD with Gradient Tracking (GT–DSGD): Undirected Graphs

As we describe in Section IV the performance of DSGD [7] improves with the addition of gradient tracking. Similarly, we can add gradient tracking in DSGD to obtain GT–DSGD, introduced in [115], [120], described as follows:

$$x_{k+1}^i = \sum_{r \in N_i^m} w_{ir} x_k^r - \alpha_k \cdot y_k^i, \quad (33a)$$

$$y_k^i = \sum_{r \in N_i^m} w_{ir} y_k^r + \nabla f_{i,r}(x_k^r) - \nabla f_i(x_k^i), \quad (33b)$$

Similar to GT–SGD, the descent direction $y_k^i$ at each node $i$ in GT–DSGD tracks the average of local stochastic gradients. In the following, we assume that each $f_{i,j}$ is $\ell$-smooth and the global cost $F$ is $\mu$-strongly-convex. With a constant step-size $\alpha \in (0, O(\min \{1 - \lambda, \kappa^{-2}/\ell \}) \frac{1}{\mu})$, the mean-squared error $E[[x_k^i - x^i]]$ decays linearly, at each node $i$, at the rate of $(1 - O(\mu\alpha))^k$ up to a steady-state error such that

$$\lim_{k \to \infty} \sup_{n} \frac{1}{n} \sum_{i=1}^{n} E\left[\left\|x_k^i - x^i\right\|^2\right] = O \left( \frac{\alpha \sigma^2}{\mu} + \frac{\alpha^2 \kappa^2 \sigma^2}{1 - \lambda} \right). \quad (34)$$

With a decaying step-size such that $\alpha_k = O(1/k)$, we have

$$\sup_{n \geq 1} \frac{1}{n} \sum_{i=1}^{n} E\left[\left\|x_k^i - x^i\right\|^2\right] = O \left( \frac{1}{k} \right), \quad \forall k \geq 0. \quad (35)$$

See [20] for explicit non-asymptotic characterizations and the performance of GT–DSGD for non-convex functions.

Remark 12 (DSGD vs. GT–DSGD): With decaying step-sizes $\alpha_k = O(1/k)$, GT–DSGD matches the convergence rate of centralized minibatch SGD after a finite number of transient iterations; see [20] for a precise transient time comparison between GT–DSGD and DSGD. With a constant step-size, the performances of DSGD and GT–DSGD have distinct features. On the one hand, the dependence on $b$ in the steady-state error of DSGD [31] does not exist in that of GT–DSGD [34]. This is intuitive because each local descent direction $y_k^i$ in GT–DSGD is towards an estimate of the average of local stochastic gradients with the help of gradient tracking; see also Remark [11]. On the other hand, the steady-state error in DSGD has a better network dependence $O((1 - \lambda)^{-2})$ when compared with $O((1 - \lambda)^{-3})$ in GT–DSGD. In short, when the data distributions across nodes are substantially heterogeneous, GT–DSGD may be preferable over DSGD.
C. Decentralized Stochastic Optimization: Directed Graphs

When the underlying network is directed, we generally are unable to implement the doubly stochastic weights in DSGD and GT-D SGD. An immediate extension is to use push-sum with DSGD (that uses column stochastic weights) to obtain Stochastic Gradient Push (SGP) \([11, 52]\) that replaces the local full batch \(\sum_j \nabla f_{ij,j}\) in the algorithm described in \([11a-11c]\) with a stochastic gradient \([52]\). A similar technique can be implemented with only row stochastic weights by writing the algorithm described in \([14a-14b]\) with stochastic gradients. For the sake of brevity, we only describe the SAB algorithm here that is a stochastic gradient variant of AB/Push-Pull. The basic algorithm is given by

\[
\begin{align*}
x_{k+1} &= \sum_{r \in \mathcal{N}^p} a_{ir} x^r_k - \alpha \cdot y^r_k, \\
y_{k+1} &= \sum_{r \in \mathcal{N}^p} b_{ir} y^r_k + \nabla f_{i,\tau_k}(x_{k+1}^r) - \nabla f_{i,\tau_k}(x^r_k),
\end{align*}
\]

where \(A = \{a_{ir}\}\) is row stochastic and \(B = \{b_{ir}\}\) is column stochastic. SAB has been recently introduced in \([111]\), where it is shown that SAB converges linearly with a constant step-size up to a steady-state error. However, a precise characterization of the rate and the steady-state error remains an open problem.

Remark 13 (Generalization of SAB): Clearly, following the discussion in Section VI, one can develop stochastic extensions of ADDOPT/Push-DIGing \([13, 40]\) and FROST \([42]\), where the former only uses column stochastic weights (see recent work in \([121]\) on S-ADDOPT that is a stochastic extension of ADDOPT for online problems) while the latter only uses row stochastic weights. These extensions rely on eigenvector estimation as described before. Over directed graphs, these variants have the promise of improved performance when compared to the SGP \([52]\) because of the additional gradient tracking component. However, a formal analysis of SAB and the extensions based on row and column stochastic weights remain open problems.

Remark 14 (Stochastic first-order oracle): All stochastic gradient methods discussed in this section are applicable to more general online stochastic problems, which do not necessarily have the finite sum structure of Problem P2 on the local costs \(f_i\)'s. In particular, each local stochastic gradient in the online case is a noisy sample of the corresponding true gradient \(\nabla f_i(x_k^r)\), instead of a randomly chosen component function \(\nabla f_{i,\tau_k}(x_k^r)\). Formally, we assume that each node is able to call a Stochastic First-order Oracle (SFO), i.e., at each iteration \(k\) and node \(i\), given \(x^r_k \in \mathbb{R}^p\) as the input, the SFO returns a stochastic gradient \(\xi^r_i(x_k^r, \xi_k^r) \in \mathbb{R}^p\), where \(\xi_k^r's\) are random vectors. The stochastic gradients satisfy the following assumptions: The set of random vectors \(\{\xi_k^r\}_{k \geq 0, i \in V}\) are statistically independent of each other, and

\[
\begin{align*}
(i) & \quad \mathbb{E}_{\xi_k^r} \left[ g^r_i(x_k^r, \xi_k^r) \right] = \nabla f_i(x_k^r), \\
(ii) & \quad \mathbb{E}_{\xi_k^r} \left[ \|g^r_i(x_k^r, \xi_k^r) - \nabla f_i(x_k^r)\|_2^2 | x_k^r \right] \leq \sigma^2.
\end{align*}
\]

See \([3, 55, 56]\) for additional details and discussion.\(^6\)

\(^6\)The bounded variance condition can be relaxed to (see e.g., \([54]\))

\[
\mathbb{E}_{\xi_k^r} \left[ \|g^r_i(x_k^r, \xi_k^r) - \nabla f_i(x_k^r)\|_2^2 | x_k^r \right] \leq c\|x_k^r\|_2^2 + \sigma^2, \text{ for some } c > 0.
\]

VII. DECENTRALIZED STOCHASTIC FIRST-ORDER METHODS WITH VARIANCE REDUCTION

The discussion in this section is only applicable to Problem P2, where the local costs \(f_i\)'s have a finite sum structure. In the centralized settings, the corresponding finite sum or batch problem \(\min F(x) = \frac{1}{n} \sum_{i=1}^n f_i(x)\) i.e., Problem P2 with one node \((n = 1)\), has been a topic of significant research recently. As noted before, SGD \([27]\) that has served as a promising solution for such problems linearly converges up to a steady-state error \([28]\) with a constant step-size. This steady-state error is explicitly given by \(\frac{\mu}{2\sigma^2}\) and is a consequence of the variance \(\sigma^2\) of the stochastic gradient. Since the variance is persistent, a plausible way to avoid it is to replace the stochastic gradient in \([27]\) with an estimator of the full batch gradient. Various variance-reduced methods are developed for the corresponding finite sum batch problems and share a key property that the variance of the gradient estimator goes to zero asymptotically.

A well-known variance reduction method is SAGA \([58]\) that replaces the randomly sampled component gradient \(\nabla f_{g_i}\) at each iteration of SGD \([27]\) with an estimator \(g_k\) of the batch gradient. The SAGA method is given by

\[
g_k = \nabla f_{g_k}(x_k) = \nabla f_{g_k} (\bar{x}_k) + \frac{1}{m} \sum_{j=1}^m \nabla f_j(x_k),
\]

where \(\bar{x}_k\) is the most recent iterate where the \(j\)th component gradient \(\nabla f_j\) was last evaluated and \(\tau_k\) is chosen uniformly at random from the index set \(\{1, \ldots, m\}\), at each iteration \(k\). The implementation of the gradient estimator \(g_k\) can be explained as follows. For an arbitrary \(x_0 \in \mathbb{R}^p\), compute the component gradients, \(\nabla f_{01}(x_0), \nabla f_{02}(x_0), \ldots, \nabla f_{0m}(x_0)\), and store them in a table. At each \(k \geq 0\), draw an index \(\tau_k\), compute \(g_k\) and replace the \(\tau_k\)-th element of the gradient table with \(\nabla f_{g_k}(x_k)\) while keeping all other entries in the table unchanged. It can be shown that the SAGA estimator is unbiased \([58]\), i.e., \(\mathbb{E}[g_k | x_k] = \nabla F(x_k)\), and has the property that

\[
\mathbb{E} \left[ \|g_k - \nabla F(x_k)\|_2^2 \right] \to 0,
\]

as \(x_k \to x^*\). Under the assumption that \(f_j \in \mathcal{S}_{\mu, \ell}, \forall j\), it can be shown that with \(\alpha = \frac{1}{2\sigma^2}\), we have \([58]\),

\[
\mathbb{E} \left[ \|x_k - x^*\|_2^2 \right] \leq C \left( 1 - \min \left\{ \frac{1}{4m}, \frac{1}{5\mu} \right\} \right)^k, \quad \forall k \geq 0,
\]

for some constant \(C > 0\). In other words, SAGA achieves an \(\epsilon\)-accurate solution of \(x^*\) in \(O \left( \max \{m, \kappa \} \ln^{-1} \right)\) component gradient evaluations, where recall that \(\kappa = \frac{L}{\mu}\) is the condition number of the global cost \(F\).

Remark 15 (SGD vs SAGA): SAGA, particularly because of the variance reduction \([18]\), achieves fast linear convergence to the optimal solution that is independent of the variance of the randomly sampled gradient at each iteration. It is sometimes argued \([3]\) that when low-precision solutions suffice, SAGA, due to its overhead in terms of gradient storage and relatively slow convergence in the beginning, may not be advantageous enough in comparison with SGD that achieves a fairly decent solution very fast with simpler iterations and no storage needs.

Remark 16 (Other variance-reduction methods): Other well-known and popular variance reduction techniques in-
include SAG [57], SVRG [59], S2GD [63], SARAH [60] and SPIDER [61]. These methods mostly differ in terms of the gradient estimator and lead to different computation and storage tradeoffs; see [3] for further discussion.

A. Decentralized Gradient Tracking and Variance Reduction: the GT-VR framework

Following the discussion so far, it is natural to improve the performance of decentralized stochastic methods with the help of both gradient tracking and variance reduction. Recall that although GT-D SGD is able to eliminate the dependence on $b$ in the steady state error of D SGD, its linear convergence with a constant step-size is still inexact due to the persistent variance of local stochastic gradients. Hence, the addition of local variance reduction to GT-D SGD potentially eliminates this steady-state error. The resulting framework, called GT-VR [70], at each node $i$ is summarized as:

1. **Local variance reduction:** Draw a local stochastic gradient and update the estimate $g^i_k$ of the local batch gradient $\nabla f_i(x^i_0)$ using a variance-reduction procedure;
2. **Global gradient tracking:** Fuse the local batch gradient estimates $g^i_k$’s over nearby nodes to obtain $y^i_{k+1}$ using the gradient tracking protocol;
3. **Local descent:** Update the state $x^i_{k+1}$ as a linear combination of the neighboring states followed by a descent in the direction of $-y^i_{k+1}$.

We can interpret the local variance reduction step as intra-node fusion where each node estimates its own local batch gradient, while the global gradient tracking step as the inter-node fusion where the nodes fuse (the estimate of) their local batch gradients over the network. In essence, the variability in the gradient selection process is removed by performing these two fusions and the overall descent is in the direction of the global negative gradient $-\nabla F$, asymptotically.

We next describe an instance of the GT-VR framework, called GT-SAGA [70] that uses SAGA-type local variance reduction.

Each node $i$ starts with an arbitrary $x^i_0 \in \mathbb{R}^p$, maintains a local gradient table $\{\nabla f_i, (x^i_0), \ldots, \nabla f_i, (x^i_m)\}$, and computes $y^i_0 = \nabla f_i(x^i_0)$. Node $i$ then iteratively performs the following steps at each $k \geq 0$:

1. **Perform the $x_k$-update:**
   $$x^i_{k+1} = \sum_{r \in N_i} w_{ir} x^r_k - \alpha y^i_k; \tag{39}$$
2. **Draw $\tau^i_k$ uniformly at random from $\{1, \ldots, m_i\}$;
3. **Local variance reduction:**
   $$g^i_{k+1} = \nabla f_i, \tau^i_k (x^i_{k+1}) - \nabla \hat{f}_{i, \tau^i_k} + \frac{1}{m_i} \sum_{j=1}^{m_i} \nabla \hat{f}_{i,j}; \tag{40}$$
   where $\hat{f}_{i, \tau^i_k}$ is the $\tau^i_k$-th element in the gradient table;
4. **Global gradient tracking:**
   $$y^i_{k+1} = \sum_{r \in N_i} b_{ir} y^i_k + g^i_{k+1} - g^i_k; \tag{41}$$
5. **Replace $\hat{f}_{i, \tau^i_k}$ with $\nabla f_i, \tau^i_k (x^i_{k+1})$ in the gradient table.**

Similar to the centralized SAGA [58], GT-SAGA converges linearly to $x^*$ with a constant step-size. Formally, when each $f_{i,j}$ is $\ell$-smooth and the global cost $F$ is $\mu$-strongly-convex, and the step-size is chosen such that $\alpha = O \left( \min \left\{ \frac{1}{\mu M}, \frac{m}{\mu (1-\lambda)^2} \right\} \right)$, we have that $\forall k \geq 0$,

$$\frac{1}{n} \sum_{i=1}^{n} \mathbb{E} \left[ \|x^i_k - x^*\|_2^2 \right] \leq R \left( 1 - \min \left\{ O \left( \frac{1}{M} \right), O \left( \frac{m}{M (1-\lambda)^2} \right) \right\} \right)^k, \tag{42}$$

where $m = \min_i \{m_i\}, M = \max_i \{m_i\}$, and $R > 0$ is some constant [70]. In other words, GT-SAGA achieves an $\epsilon$-accurate solution of $x^*$ in

$$O \left( \max \left\{ M, \frac{m}{M (1-\lambda)^2} \right\} \ln \frac{1}{\epsilon} \right) \tag{43}$$

component gradient computations at each node.

Remark 17 (Non-asymptotic linear speedup of GT-SAGA): When each node has a large data set such that $M \approx m \geq \frac{\epsilon}{\lambda^2}$, the complexity [43] of GT-SAGA becomes $O(M \ln \epsilon^{-1})$, independent of the network, and is further $n$ times faster than that of the centralized SAGA algorithm where it is $O(nM \ln \epsilon^{-1})$ [58]. Clearly, in this “big-data” regime, GT-SAGA acts effectively as a means for parallel computation and achieves linear speed-up compared with centralized variance-reduced methods. We also numerically show the linear speedups of decentralized algorithms in Section VIII.

We finally note that other instances of the GT-VR framework also achieve non-asymptotic linear speedup in certain regimes; see [21], [69], [70] for details.

Remark 18 (DSGD, GT-D SGD, and GT-SAGA): Recapping the performances of the decentralized stochastic first-order methods, we note that GT-SAGA achieves linear convergence to $x^*$ under a constant step-size without a steady-state error due to the local variance-reduction scheme employed at each node. However, the gradient tables result in an additional storage cost of $O(pm_i)$ at each node $i$. Given a desired accuracy, applicable trade-offs between the convergence benefits of the GT-VR framework versus the simplicity of other methods can be formulated; see also [69] for a detailed review.

Remark 19 (Other instances of the GT-VR framework): The discussion in this section enables a suite of methods that combine various gradient tracking and variance reduction techniques leading to different performance and implementation trade-offs towards different problem classes [21], [69]. For example, the storage of GT-SAGA can be avoided in GT-SVRG, which computes batch gradients periodically, however, at the expense of additional network synchronization [69], [70]. For non-convex finite sum problems, it is recently shown that GT-SARAH [21], that uses variance reduction of SARAH-type [60], [61], achieves an optimal and network-independent performance in a certain regime of practical interest. See also [71] that uses SARAH-type variance reduction for non-convex expected risk minimization problems. We also note that a recent work [73] proposes Push-SAGA that extends GT-SAGA to arbitrary directed graphs with the help of push-sum consensus protocol.

Remark 20 (Other decentralized variance reduction methods): Decentralized variance-reduced stochastic gradient methods include DSA [64] that combines EXTRA [31] and SAGA [58], diffusion-AVRG [65] that combines exact diffusion [34] and AVR G [122], DSBA [66] that adds proximal
mapping [23] to DSA, ADFS [68] that applies an accelerated randomized proximal coordinate gradient method [124] to a dual formulation, and Network-SVRG/SARAH [72] that implements variance reduction in the DANE framework [125]. We note that when \( M \approx m \) is very large, GT-SAGA improves upon the convergence rate of these methods in terms of the joint dependence on \( \kappa \) and \( M \), with the exception of DSBA and ADFS, both of which achieves better iteration complexity, however, at the expense of computing proximal mappings.

VIII. NUMERICAL EXPERIMENTS

In this section, we compare the performance of the decentralized first-order methods discussed in this article with the help of linear and logistic regression problems.

1) Decentralized state estimation in sensor networks

Consider a network of sensors deployed to estimate a state vector \( \mathbf{x} \in \mathbb{R}^p \), e.g., the location of a target or a physical quantity of interest. Each sensor \( i \) obtains its local measurements \( \mathbf{y}_i \in \mathbb{R}^p \) as \( \mathbf{y}_i = \mathbf{H}_i \mathbf{x} + \mathbf{n}_i \), where \( \mathbf{H}_i \in \mathbb{R}^{p \times p} \) is the local sensing matrix and \( \mathbf{n}_i \) is some random noise. To recover the state vector, the nodes cooperate to solve the following optimization problem: \( \min_{\mathbf{x} \in \mathbb{R}^p} \sum_{i=1}^n \| \mathbf{y}_i - \mathbf{H}_i \mathbf{x} \|^2 \). We randomly generate an undirected graph of 50 nodes using the nearest neighbor rule, i.e., two nodes are connected if they lie within a communication radius. A directed graph is obtained by making half of the links in the undirected geometric graph one-directional. Two such sample graphs are shown in Fig. 3.

We set the dimension of the state \( \mathbf{x} \in \mathbb{R}^p \) to be \( p = 100 \) and generate the sensing matrices and the state vector from Gaussian distribution with zero-mean and a standard deviation of 10. The maximum rank of the sensing matrices does not exceed 20, i.e., no node is able to recover the state on its own.

Fig. 3. An undirected geometric graph (top) and a directed geometric graph (bottom) generated by nearest neighbor rule.

In Fig. 4 we compare DGD, GT-DGD, and AB/Push-Pull over an undirected graph. Note that doubly stochastic weights are used for DGD and GT-DGD while row and column stochastic weights are used in AB/Push-Pull. The same constant step-size \( \alpha = 10^{-5} \) is used for all of the algorithms. As discussed before, DGD with a constant step-size has an inexact linear convergence while with a decaying step-size has an exact but sub-linear convergence. Both GT-DGD and AB/Push-Pull converge linearly to the exact solution due to gradient tracking. In Fig. 7 we compare Gradient-Push (GP), ADDOPT/Push-DIGing (with only column stochastic weights), FROST (with only row stochastic weights) and AB/Push-Pull over a directed graph. GP being an extension of DGD with push-sum consensus has an inexact linear convergence with a constant step-size while all of the other algorithms are based on gradient tracking and converge to the exact optimal solution linearly.

2) Decentralized training over multiple machines

We next consider decentralized logistic regression over the directed exponential graphs [52], shown in Fig. 6 where the goal is to classify hand-written digits 3 and 8 from the MNIST dataset [126]. The class of directed exponential graphs is weight-balanced (therefore admits doubly stochastic weights), sparsely-connected (low-communication cost per node), and possesses a strong algebraic connectivity. The nodes cooperatively solve the following smooth and strongly convex problem: \( \min_{\mathbf{w}, \mathbf{b}} F(\mathbf{w}, \mathbf{b}) = \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m f_{i,j}(\mathbf{w}, \mathbf{b}) \), with \( f_{i,j}(\mathbf{w}, \mathbf{b}) = \ln \left( 1 + \exp \left\{ - (\mathbf{w}^\top \mathbf{z}_{i,j} + \mathbf{b})y_{i,j} \right\} \right) + \frac{\lambda}{2} \| \mathbf{w} \|_2^2 \).
where $z_{i,j} \in \mathbb{R}^{784}$ is the feature vector of the $j$th (digit) image at node $i$, and $y_{i,j}$ is the corresponding binary label. In our setup, a total number of 11968 training images are evenly distributed among the nodes. We set the regularization parameter $\lambda = \frac{1}{nm}$ [57]. The global minimum is found by centralized Nesterov gradient descent. We plot the average residual $\frac{1}{n} \sum_{i=1}^{n} (F(x_k^i) - F^*)$ across all nodes for comparison. The hyper-parameters for all algorithms are tuned manually for best performances.

We first show performance comparison over the directed exponential graph with $n = 8$ nodes (thus there are $nm = 1496$ images per node). Fig. 7 compares DGD, GT-DGD, GT-DSGD, GT-DSDG, and GT-SAGA, all with constant step-sizes, where one unit on the horizontal axis (or epoch) represents $m = 1496$ component gradient evaluations at each node, i.e., one effective pass of the local data. It can be observed that in the first few epochs, stochastic gradient methods, DSGD and GT-DSGD, make very fast progress and significantly outperform their deterministic counterparts, DGD and GT-DGD. Over time, however, stochastic methods drastically slow down and full gradient methods that make steady progress across iterations typically achieve a better performance. Clearly, GT-SAGA that uses both gradient tracking and variance-reduction exhibits consistent fast linear convergence to the global minimum and outperforms all other methods.

Finally, we study the speedup achieved by the decentralized stochastic methods over their centralized counterparts in Fig. 8, where we compare DSGD and GT-DSGD with centralized SGD, and GT-SAGA with centralized SAGA. Each iteration of centralized SGD and SAGA evaluates one component gradient while each iteration of DSGD, GT-DSDG and GT-SAGA evaluates $n$ component gradients in parallel, i.e., one at each node. The speedup can be described as the the ratio of the number of iterations taken by the centralized method over the corresponding decentralized method to achieve the same accuracy ($10^{-3}$ for comparisons with SGD and $10^{-15}$ for comparison with SAGA). We conduct the experiment over the directed exponential graphs with 4, 8, 6, and 32 nodes shown in Fig. 6. It can be observed that a linear speedup is achieved for all three decentralized approaches. In other words, the decentralized methods are approximately $n$ times faster than their centralized counterparts and thus are particularly favorable when data can be processed in parallel.

![Fig. 6. Directed exponential graphs with 4, 8, 16, and 32 nodes. These graphs are balanced and thus admit doubly stochastic weights.](image)

![Fig. 7. Decentralized logistic regression over the directed exponential graph with $n = 8$ nodes.](image)

![Fig. 8. Linear speedup: DSGD and GT-DSGD vs. SGD to achieve an accuracy of $10^{-3}$; GT-SAGA vs. SAGA to achieve an accuracy of $10^{-15}$.](image)

**IX. Conclusions**

In this article, we study decentralized first-order methods with full and stochastic gradients over undirected and directed graphs. We show that most of the existing work on decentralized methods based on gradient tracking can be cast in a unifying framework provided by the AB/Push-Pull algorithm. On the stochastic front, we discuss how gradient tracking is unable to achieve exact linear convergence, which can be recovered with the addition of variance-reduction. We provide numerical results to illustrate the convergence behavior and performance of the corresponding methods.

**References**

[1] B. Polyak, *Introduction to optimization*, Optimization Software, 1987.

[2] Y. Nesterov, *Lectures on convex optimization*, vol. 137, Springer, 2018.

[3] L. Bottou, F. E Curtis, and J. Nocedal, “Optimization methods for large-scale machine learning,” *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[4] G. Lan, *First-order and Stochastic Optimization Methods for Machine Learning*, Springer, 2020.
D. Jakovetic, D. Bajovic, A. K. Sahu, and S. Kar, “Convergence rates R. A. Horn and C. R. Johnson, S. Han, “Systematic design of decentralized algorithms for consensus R. A. Horn and C. R. Johnson, S. Han, “Systematic design of decentralized algorithms for consensus

S. Kar and J. M. F. Moura, “Consensus+ innovations distributed inference S. Kar and J. M. F. Moura, “Consensus+ innovations distributed inference

V. S. Mai and E. H. Abed, “Distributed optimization over weighted directed graphs using row stochastic matrix,” in IEEE American Control Conference, July 2016, pp. 7165–7170.

G. Scutari and Y. Sun, “Distributed nonconvex constrained optimization over time-varying digraphs,” Mathematical Programming, vol. 176, no. 1-2, pp. 497–544, 2019.

S. Pu, W. Shi, J. Xu, and A. Nedić, “Push-pull gradient methods for distributed optimization in networks,” IEEE Trans. on Autom. Control, 2020.

R. Xin and U. A. Khan, “Distributed heavy-ball: A generalization and acceleration of first-order methods with gradient tracking,” IEEE Trans. on Autom. Control, 2019.

S. Safavi and U. A. Khan, “Leader-follower consensus in mobile sensor networks,” IEEE Signal Processing Letters, vol. 22, no. 12, pp. 2249–2253, Dec. 2015.

R. Xin, A. K. Sahu, U. A. Khan, and S. Kar, “Distributed stochastic optimization with gradient tracking over strongly connected networks,” in 58th IEEE Conference of Decision and Control, Nice, France, 2019.

R. Xin, D. Jakovetić, and U. A. Khan, “Distributed Nesterov gradient methods over arbitrary graphs,” IEEE Signal Processing Letters, vol. 26, no. 18, pp. 1247–1251, Jun. 2019.

G. Qu and N. Li, “Accelerated distributed nesterov gradient descent,” IEEE Trans. on Autom. Control, 2019.

A. Nedić and A. Olshevsky, “Distributed optimization over time-varying directed graphs,” in 52nd IEEE Annual Conference on Decision and Control, Dec. 2013, pp. 6855–6860.

S. Pu and A. Nedić, “A distributed stochastic gradient tracking method,” in Proc. IEEE Conf. Decis. Control, 2018, pp. 963–968.

D. Jakovetic, D. Bajovic, A. K. Sahu, and S. Kar, “Convergence rates for distributed stochastic optimization over random networks,” in IEEE Conference on Decision and Control, 2018, pp. 4238–4245.

A. Koloskova, S. U Stich, and M. Jaggi, “Decentralized stochastic optimization and gossip algorithms with compressed communication,” arXiv:1902.00340, 2019.

S. Pu, A. Olshevsky, and I. C. Paschalidis, “A sharp estimate on the transient time of distributed stochastic gradient descent,” arXiv:1906.02702, 2019.

S. Pu, A. Olshevsky, and I. C. Paschalidis, “Asymptotic network independence in distributed stochastic optimization for machine learning: Examining distributed and centralized stochastic gradient descent,” IEEE Signal Processing Magazine, vol. 37, no. 3, pp. 114–122, 2020.

S. Pu and A. Nedić, “Distributed stochastic gradient tracking methods,” Mathematical Programming, pp. 1–49, 2020.

M. I. Qureshi, R. Xin, S. Kar, and U. A. Khan, “S-ADDOPT: Decentralized stochastic first-order optimization over directed graphs,” IEEE Control Systems Letters, vol. 5, no. 3, pp. 953–958, Jul. 2021.

B. Ying, K. Yuan, and A. H. Sayed, “Variance-reduced stochastic learning under random reshuffling,” arXiv:1708.01383, 2017.

A. Defazio, “A simple practical accelerated method for finite sums,” in Advances in neural information processing systems, 2016, pp. 676–684.

Q. Lin, Z. Lu, and L. Xiao, “An accelerated randomized proximal coordinate gradient method and its application to regularized empirical risk minimization,” SIAM J. Optim., vol. 25, no. 4, pp. 2244–2273, 2015.

O. Shamir, N. Srebro, and T. Zhang, “Communication-efficient distributed optimization using an approximate newton-type method,” in International conference on machine learning, 2014, pp. 1000–1008.

Y. LeCun, “The MNIST database of handwritten digits,” http://yann.lecun.com/exdb/mnist/, 1998.