AN EMPIRICAL COMPARISON OF MULTI-AGENT OPTIMIZATION ALGORITHMS

Mahmoud Assran and Michael Rabbat

Electrical and Computer Engineering
McGill University, Montréal, Canada
Email: mahmoud.assran@mail.mcgill.ca, michael.rabbat@mcgill.ca

ABSTRACT

In the past decade a large number of distributed algorithms for solving large-scale convex optimization problems have been proposed and analyzed in the literature, especially from the perspective of multi-agent systems. Although it is fairly well understood which algorithms have the most desirable theoretical properties, there has been very little work investigating and evaluating practical implementations of these algorithms, and there is a non-trivial gap between theory and practice. For example, many of the theoretical analyses ignore important practical issues such as asynchronism and communication delays. In this paper we perform an empirical evaluation of non-doubly stochastic multi-agent distributed optimization algorithms for large-scale convex optimization and open source the code. We find that a first order asynchronous subgradient optimization algorithm can actually out-perform state-of-the-art synchronous algorithms in a practical scenario for both small and large multi-agent networks running on a high performance cluster.

1. INTRODUCTION

The community has developed fast multi-agent optimization algorithms with strong theoretical performance guarantees. In this paper we focus on the empirical performance of the algorithms in solving the standard optimization problem:

$$\begin{aligned} \min_{x_i \in \mathbb{R}^d (i=1,\cdots,n)} \quad & F(\mathbf{x}) := \sum_{i=1}^{n} f_i(x_i) \\ \text{subject to} \quad & x_i = x_j \;\; (i,j = 1,\ldots,n). \end{aligned}$$

In particular, we perform simulations on a high performance cluster and show that algorithms with superior theoretical guarantees (e.g. faster known convergence rates) do not necessarily achieve faster convergence times in practice. We also provide a distributed Python implementation of these multi-agent optimization algorithms using the latest MPI standards, and make them available to the community.

Multi-agent optimization algorithms are inherently message-passing and can be decomposed into a generic two-step iterative process involving gossip and local minimization. The local minimization is used to steer the solution towards the minimizer, and the gossip is used to achieve consensus amongst the agents. Algorithms following this generic model can be subdivided into two classes: *push-based* where nodes can only send (push) messages to their neighbours, and *push-pull-based*, where nodes can send (push) and receive (pull) messages from their neighbours. The issue with *push-pull-based* (send-receive) algorithms is that they are susceptible to deadlocks; nodes can potentially initiate message exchanges with each other in a fashion that results in each node waiting on a response from another neighbour in the deadlock loop [1]. In this paper we only consider *push-based* algorithms, which are immune to deadlocking message exchanges [1]. Due to the nature of *push-based* methods, only directed communication graphs are required.

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada grant RGPIN-2017-06266.

Notation. Let $\mathbf{I}_n$ denote the $n \times n$ identity matrix, and $\mathbf{1}_n$ represent an n-dimensional column vector with each entry equal to 1. A generic local variable z held by agent i in the network at time t is denoted by the vector $z_i(t)$ where $z_i(t) \in \mathbb{R}^d$. Let $\mathbf{z}(t) \in \mathbb{R}^{n \times d}$ be a matrix which stores the copy of variable z for the entire n-agent network at time t, where $(z_i(t))^T$ is the i^{th} row of $\mathbf{z}(t)$. In general, all boldfaced symbols are matrices with the exception of $\mathbf{1}_n$ which is a vector.

2. ALGORITHMS

In this section we describe the algorithms being compared. Since the Push Sum Protocol for Consensus Averaging [2] lies at the foundation of the multi-agent methods analyzed in this paper we begin by briefly introducing it.

2.1. Push-Sum Consensus Averaging

Let each agent i in the n-agent network hold a local variable $x_i(0)$; the goal of distributed gossip averaging is to have all the nodes in the network agree (achieve consensus) on the average of their initial values by gossiping with their neighbours. i.e. $\mathbf{x}(t) \to \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T\mathbf{x}(0)$. Converging to the true average of the network through gossip is an iterative process; a typical consensus averaging protocol solves the problem by performing the iterations $\mathbf{x}(t+1) = \mathbf{A}(t)\mathbf{x}(t)$ starting at $\mathbf{x}(0)$ where $\mathbf{A}(t)$ is a doubly stochastic matrix. However doubly stochastic protocols are undesirable for many reasons, especially in peer-to-peer networks, which tend to lack a highly organized structure. This lack of organization, as well as the fact that node and link failures have become the norm rather than the exception [1, 2, 3], present new types of constraints on the consensus algorithms being used, thereby eliminating the applicability of many doubly-stochastic protocols, which in general tend to have more strict network constraints [1, 2, 4, 5, 6, 7]. The push-sum protocol achieves consensus averaging using only column stochastic matrices, and is directly used in the state-of-the-art optimization algorithms explored in this paper. The formulation of the Push Sum Averaging Protocol is:

$$\begin{aligned} &\mathbf{y}(t+1) = \mathbf{A}(t)\,\mathbf{y}(t) \\ &\mathbf{w}(t+1) = \mathbf{A}(t)\,\mathbf{w}(t);\; \mathbf{W}(t+1) = \text{diag}(\mathbf{w}(t+1)) \\ &\mathbf{x}(t+1) = (\mathbf{W}(t+1))^{-1}\mathbf{y}(t+1). \end{aligned}$$

The initializations are $\mathbf{y}(0) = \mathbf{x}(0)$, $\mathbf{w}(0) = \mathbf{1}_n$ where $\mathbf{y}(t) \in \mathbb{R}^{n\times d}$ and $\mathbf{w}(t) \in \mathbb{R}^{n\times 1}$ in keeping with our notation. In the sequel, $w_i(t)$ are referred to as the push-sum weights, $y_i(t)$ the push-sum numerators, $\mathbf{A}(t)$ the mixing matrices, and $x_i(t)$ the consensus estimates.

Since the mixing matrices are only column-stochastic, sending nodes simply control their own respective columns of the mixing matrices independently without coordination with the other nodes in the network. At each averaging iteration t the nodes gossip (or *push*) their scaled push-sum numerators, $A_{ij}(t)y_i(t)$, and their scaled push-sum weights, $A_{ij}(t)w_i(t)$, to their neighbours. After pushing their values to peers, nodes subsequently perform a local update by *summing* the push-sum messages that they have received. Any bias built up in the push-sum numerator $y_i(t)$ is also built up in the scalar push-sum weight $w_i(t)$, and so a division of the push-sum numerator by the scalar yields the unbiased consensus estimate of the network average. After several such iterations, the nodes will converge to the true average of the network.

2.1.1. Synchronous vs. asynchronous

The push-sum protocol can be implemented synchronously — where all the nodes gossip and perform updates at the same time and wait for the slowest node before moving on to the next gossip-update round, or asynchronously — where nodes gossip once a local update has been completed and do not wait to receive messages from any of their neighbours before proceeding to the next iteration [8]. The main distinction here between synchronous and asynchronous distributed protocols refers to the presence or absence of a common global clock used to synchronize the operation of the different processors [9]. In the synchronous case, the iteration index t is the same at each node, while in the asynchronous case, the iteration index t can differ from node to node at any time instance; i.e. nodes can perform a drastically different number of updates in the same period of time and furthermore may be working with outdated values.

An issue with synchronous gossip is that the slowest node in the network causes a bottleneck on the convergence time. This can be understood by recalling that all nodes must wait on each other before proceeding to the next gossip-update round. Such behaviour can lead to an accumulating delay in the convergence time of the algorithm if one node in the cluster runs at a slightly slower pace than the other nodes in the cluster. Asynchronous gossip on the other hand allows all the nodes to work at their own rate without slowing each other down. Generally speaking, asynchronous modifications present several improvements to synchronous algorithms, but also come with their own unique host of practical difficulties [1].

2.2. Subgradient Push Optimization

The Subgradient Push optimization algorithm [7] is used as a benchmark and corresponds to a *push-based* analogue of the classical gradient descent algorithm:

$$\begin{aligned}\mathbf{y}(t+1) &= \mathbf{A}(t)\,[\mathbf{y}(t) - \alpha\nabla\mathbf{F}(\mathbf{x}(t))]\\ \mathbf{w}(t+1) &= \mathbf{A}(t)\,\mathbf{w}(t);\ \mathbf{W}(t+1) = \text{diag}(\mathbf{w}(t+1))\\ \mathbf{x}(t+1) &= (\mathbf{W}(t+1))^{-1}\mathbf{y}(t+1),\end{aligned}$$

where $\nabla\mathbf{F}(\mathbf{x}(t))$ is the Jacobian of the global objective $F(\mathbf{x})$ at time t with respect to the x_i in the differentiable objective setting (in the non-differentiable but convex objective setting, the rows of $\nabla\mathbf{F}(\mathbf{x}(t))$ are members of the subgradient set of the global objective $F(\mathbf{x})$ at time t). The initializations are $\mathbf{y}(0) = \mathbf{x}(0)$, $\mathbf{w}(0) = \mathbf{1}_n$, and α is just a positive scalar step-size. The push-sum modification simply interleaves a push-sum averaging step with a gradient-descent step. The push-sum averaging step is used to steer the nodes' estimates of the optimal parameter setting towards each other (consensus), and the gradient-descent step is used to steer the nodes' arguments towards the minimizer (greedy minimization). The Subgradient Push algorithm converges sublinearly at a rate of $O(\ln\, t/\sqrt{t})$ for general convex functions [7], and at a rate of $O(\ln\, t/t)$ for strongly convex functions [10].

2.2.1. Asynchronous Subgradient Push Optimization

Asynchrony is introduced in the following sense: Messages are sent in a non-blocking fashion between agents with some arbitrary, but bounded, transmission delays. Agents do not wait to receive messages from their neighbours before proceeding with their computation, and therefore may perform local gradient steps with outdated information. Agents may take some arbitrary, but bounded, amount of time to perform a gradient step at each local iteration, and therefore may perform a drastically different number of gradient steps over any time interval.

2.3. Extra Push Optimization

The first order state-of-the-art Extra Push optimization algorithm [6] is an extension of the original EXTRA (exact first order algorithm) [11]. The Extra Push update is:

$$\begin{aligned}\mathbf{y}(t+1) &= [\mathbf{A}+\mathbf{I}_n]\mathbf{y}(t) - \frac{1}{2}[\mathbf{A}+\mathbf{I}_n]\mathbf{y}(t-1)\\ &\quad - \alpha[\nabla\mathbf{F}(\mathbf{x}(t)) - \nabla\mathbf{F}(\mathbf{x}(t-1))]\\ \mathbf{w}(t+1) &= \mathbf{A}\,\mathbf{w}(t);\ \mathbf{W}(t+1) = \text{diag}(\mathbf{w}(t+1))\\ \mathbf{x}(t+1) &= (\mathbf{W}(t+1))^{-1}\mathbf{y}(t+1).\end{aligned}$$

The initializations are $\mathbf{y}(0) = \mathbf{x}(0)$, $\mathbf{w}(0) = \mathbf{1}_n$, $\mathbf{w}(1) = \mathbf{A}\mathbf{w}(0)$, $\mathbf{y}(1) = \mathbf{A}\mathbf{y}(0) - \alpha\nabla\mathbf{F}(\mathbf{x}(0))$, and α is just a positive (scalar) step-size. The Extra Push update can be directly derived from the Subgradient Push update by taking the difference between two successive iterations using the mixing matrices $\mathbf{A}$ and $\widetilde{\mathbf{A}}$ repsectively where $\widetilde{\mathbf{A}} := \frac{1}{2}(\mathbf{I}_n + \mathbf{A})$. By using a gradient difference Extra Push is able to achieve exact convergence for general convex functions using a constant step-size. However, due to the nature of the push-based updates, the literature on the convergence theory for the Extra Push algorithm is restricted to synchronous and static (time-invariant) directed graphs, hence the omission of the iteration index for the mixing matrices in the above formulation. The Extra Push algorithm converges Q-linearly when the objective is strongly convex [5, 6].

2.4. Push DIGing Optimization

The Push DIGing optimization algorithm [5] is another first order state-of-the-art method. The Push DIGing update is:

$$\begin{aligned}\mathbf{y}(t+1) &= \mathbf{A}(t)\,[\mathbf{y}(t) - \alpha\mathbf{z}(t)]\\ \mathbf{w}(t+1) &= \mathbf{A}(t)\,\mathbf{w}(t);\ \mathbf{W}(t+1) = \text{diag}(\mathbf{w}(t+1))\\ \mathbf{x}(t+1) &= (\mathbf{W}(t+1))^{-1}\mathbf{y}(t+1)\\ \mathbf{z}(t+1) &= \mathbf{A}(t)\,\mathbf{z}(t) + [\nabla\mathbf{F}(\mathbf{x}(t+1) - \nabla\mathbf{F}(\mathbf{x}(t)].\end{aligned}$$

The initializations are $\mathbf{y}(0) = \mathbf{x}(0)$, $\mathbf{w}(0) = \mathbf{1}_n$, $\mathbf{z}(0) = \nabla\mathbf{F}(\mathbf{x}(0))$, and once more α is just a positive (scalar) step-size. The algorithm shares many similarities to Extra Push, however one

of the most salient differences is that the Push DIGing method performs dynamic gradient tracking through the variable z, and is therefore capable of converging over time varying directed graphs. The synchronous Push Diging algorithm converges R-linearly when the objective is strongly convex [5].

3. PRACTICAL IMPLEMENTATION ISSUES

In this section we discuss some practical implementation issues that arise when implementing multi-agent optimization algorithms; our implementations are in Python and use the MPI standards for message passing.

3.1. Blocking vs non-blocking communiation

In theory, *push-based* methods are immune to deadlock. However, if one uses that standard `Send()` and `Recv()` interfaces specified by MPI, deadlock will likely occur to due the behaviour of the underlying communication protocol and we have certainly observed this in our experiments. To avoid deadlock in our implementations all nodes perform non-blocking send operations followed by either a blocking receive (synchronous case), or a non-blocking receive (asynchronous case). The non-blocking receive corresponds to simply emptying the receive buffer, while the blocking receive corresponds to waiting to receive a pre-specified number of messages.

3.2. Numerical instability in the asynchronous setting

In the context of the distributed optimization methods analyzed in this paper, numerical instability manifests itself as the growth of round-off errors that cause the optimization process to fail. In particular, node i scales its push-sum weight $w_i(t)$ at time t by the mixing matrix entry A_{ii} which is strictly less than one. If a node does not receive any messages from its peers at time t, the new push-sum weight at time $t+1$ is $w_i(t+1) = A_{ii}w_i(t)$. In the asynchronous setting, if a node is able to compute much faster than it can communicate, it can go several iterations, say k iterations, without receiving any messages, in which case, the new push-sum weight at time $t+k$ is $w_i(t+k) = (A_{ii})^k w_i(t)$ and the push-sum weight decays geometrically. It is possible that the push-sum weight becomes so small (close to machine precision) that an incorrect result is produced if we try to compute the consensus estimate $x_i(t+k) = \frac{y_i(t+k)}{w_i(t+k)}$. Such numerical instability issues for asynchronous algorithms are encountered in our experiments. To get around this issue, we adopt the procedure suggested in [1] and prohibit a node from sending messages to its neighbours if its push-sum weight drops below a certain threshold value.

4. EXPERIMENTAL SETUP

4.1. Graph generation

The graphs used in the distributed optimization algorithms are randomly generated. Amongst other parameters, our implementation allows the user to specify the size of desired graph, as well as the average out-degree of each node. The generator produces a user-specified number of graphs, confirms that they are strongly connected by performing a breadth-first search through the graphs, takes the adjacency matrices that are strongly connected and scores them in terms of information diffusion speed by creating an equivalent doubly stochastic uniform edge weighted matrix using a procedure similar to that in [12], computes the second largest eigenvalue of the doubly stochastic matrices (the largest is equal to one), and saves the graph corresponding to the matrix with the smallest second largest eigenvalue [13].

4.2. High performance computing cluster

The experiments are conducted using high performance computing (HPC) facilities. The cluster has 21000 processing cores running on the latest Intel Sandy Bridge and Westmere processesors. The cluster makes use of a QDR InfiniBand network capable of 40 Gbps to each node and hosts large scale storage systems operating with an optimized parallel file system. The MVAPICH2 MPI distribution is used with Python bindings for message passing.

4.3. Optimization problem

A multinomial logistic regression classifier (softmax predictor) is trained on the *Covertype* dataset [14] (available from the *UCI repository*) using the negative log-likelihood loss function. The dataset contains 581012 data samples, and 54 raw predictive features for each of the seven class. Consequently the optimizer solves for 378 (=54×7) parametrizing weights. Parallelization is performed by giving each node in the n-agent network a subset of the data samples ($\sim 581012/n$) from which to construct local negative log-likelihood loss functions, $f_i(w_i)$, where $w_i \in \mathbb{R}^{378}$ are the parameterizing weight vectors held locally by agent i.

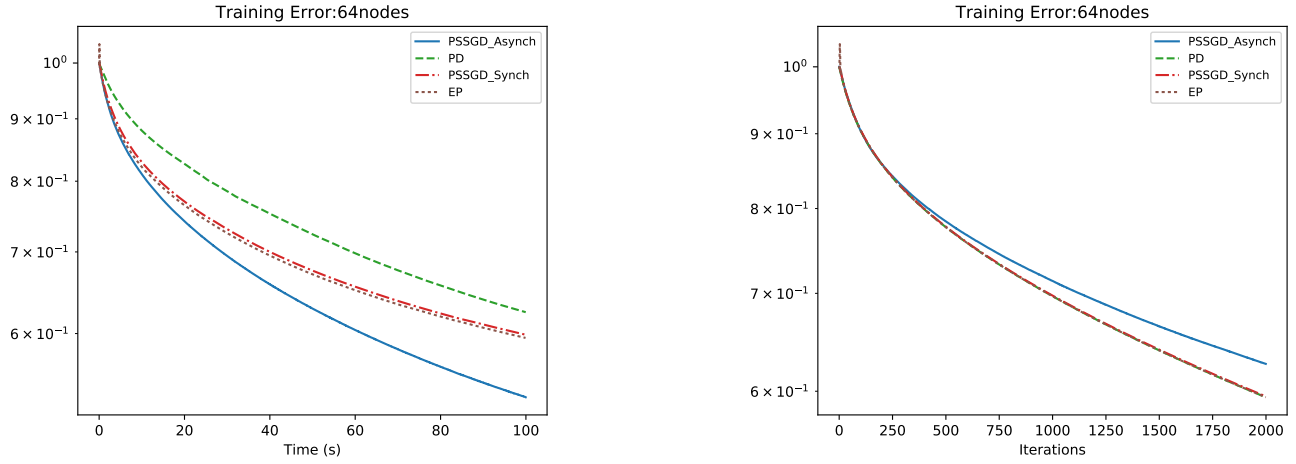
5. RESULTS

5.1. Observations

The Push DIGing (PD), Extra Push (EP), Synchronous Subgradient Push (PSSGD_Synch), and Asynchronous Subgradient Push (PSSGD_Asynch) algorithms are used to minimize the negative log-likelihood of the softmax function constructed form the *Covertype* dataset. Our experiments show that there is a discrepancy between the classical analysis of algorithms (in iterations) and their practical convergence rates (in time). Figure 1 shows one such example, where the asynchronous subgradient algorithm achieves the slowest error-minimization in iterations, but the fastest in time. Furthermore, Figure 2 shows that the asynchronous subgradient algorithm actually decreases the residual error for both small and large network sizes faster than the state-of-the art methods and its synchronous counterpart. The asynchronous algorithm also appears to be more robust than the synchronous algorithms to failing or stalling nodes. Figure 3 shows the residual error of the algorithms for different network sizes, with an artificial 500 millisecond delay induced at agent 1 at each iteration. The synchronous algorithms experience a significant slowdown relative to the results in Figure 2, whereas the asynchronous algorithm is largely unaffected.

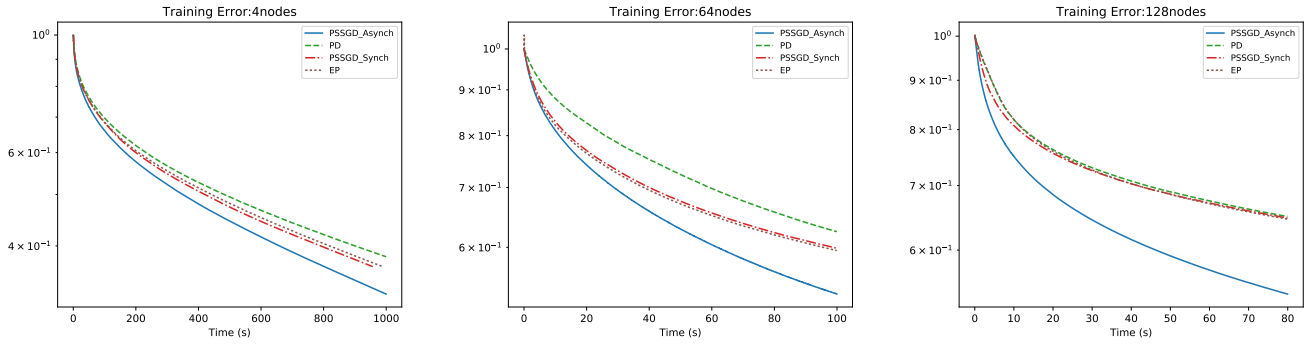
6. CONCLUSION

In this paper, we provided an empirical exposition of the non-trivial gap between theory and practice in distributed optimization algorithms from the perspective of multi-agent systems. We solved large-scale optimization problems using a high performance cluster, and observed that a first-order asynchronous subgradient algorithm actually outperforms the current state-of-the-art synchronous algorithms in a practical setting from both a robustness and speed perspective.



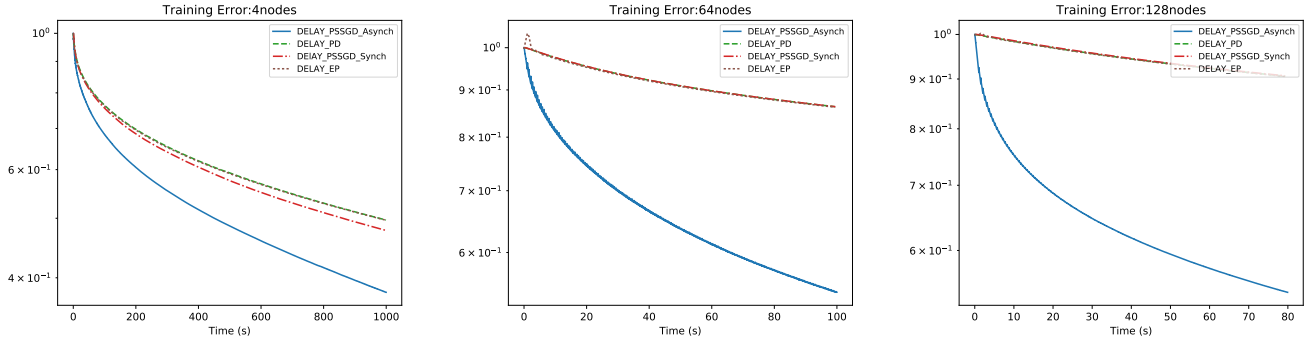
(a) (b)

Fig. 1. $\min_i(||w_i(t) - w^*||_2 \;/\; ||w_i(0) - w^*||_2)$ for a 64-agent network with no artificial delays. The asynchronous algorithm decreases the residual error slower than the synchronous algorithms when compared on a classical per-iteration basis, but faster in time.



(a) (b) (c)

Fig. 2. $\min_i(||w_i(t) - w^*||_2 \;/\; ||w_i(0) - w^*||_2)$ versus time, with no artificial delays. In all networks, the asynchronous subgradient algorithm is able to decrease the residual error faster than the state-of-the art methods, and its synchronous counterpart.



(a) (b) (c)

Fig. 3. $\min_i(||w_i(t) - w^*||_2 \;/\; ||w_i(0) - w^*||_2)$ versus time, with an artificial 500 millisecond delay induced at agent 1 at each iteration.

7. REFERENCES

[1] Konstantinos I Tsianos, Sean Lawlor, and Michael G Rabbat, "Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning," in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*. IEEE, 2012, pp. 1543–1550.

[2] David Kempe, Alin Dobra, and Johannes Gehrke, "Gossip-based computation of aggregate information," in *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*. IEEE, 2003, pp. 482–491.

[3] Jeffrey Dean and Luiz André Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

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

[5] Angelia Nedich, Alex Olshevsky, and Wei Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *arXiv preprint arXiv:1607.03218*, 2016.

[6] J. Zeng and W. Yin, "Extra push for convex smooth decentralized optimization over directed networks," *UCLA CAM Report*, vol. 15-61, 2015, http://arxiv.org/abs/1511.02942.

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

[8] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE transactions on automatic control*, vol. 31, no. 9, pp. 803–812, 1986.

[9] Dimitri P Bertsekas and John N Tsitsiklis, *Parallel and distributed computation: numerical methods*, vol. 23, Prentice hall Englewood Cliffs, NJ, 1989.

[10] Angelia Nedić and Alex Olshevsky, "Stochastic gradient-push for strongly convex functions on time-varying directed graphs," *IEEE Transactions on Automatic Control*, vol. 61, no. 12, pp. 3936–3947, 2016.

[11] Wei Shi, Qing Ling, Gang Wu, and Wotao Yin, "Extra: An exact first-order algorithm for decentralized consensus optimization," *SIAM Journal on Optimization*, vol. 25, no. 2, pp. 994–966, 2015.

[12] Valerio Cappellini, Hans-Jürgen Sommers, Wojciech Bruzda, and Karol Życzkowski, "Random bistochastic matrices," *Journal of Physics A: Mathematical and Theoretical*, vol. 42, no. 36, pp. 365209, 2009.

[13] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah, "Randomized gossip algorithms," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2508–2530, 2006.

[14] M. Lichman, "UCI machine learning repository," 2013.