FAST ASYNCHRONOUS DECENTRALIZED OPTIMIZATION: ALLOWING MULTIPLE MASTERS

Meng Ma[†] *Jineng Ren*[†] *Georgios B. Giannakis*[†*] *Jarvis Haupt*[†*]

[†] Department of ECE, University of Minnesota
[*] Digital Technology Center, University of Minnesota

ABSTRACT

The present paper deals with asynchronous decentralized optimization over networks. Pertinent algorithms are either *centralized* relying on a specific topology, where a single master connects all workers, or *decentralized* devoid of any master by only exchanging information between single-hop neighbors. The present work bridges the gap of existing approaches with a novel *hybrid* framework that is capable of accommodating multiple masters. Moreover, it enables considerable acceleration of decentralized approaches without physically deploying masters, thus making it possible to achieve a desirable tradeoff between convergence and communication/computation complexity by tuning the configuration. Numerical tests showcase advantages over decentralized counterparts.

Index Terms— Asynchronous, decentralized optimization, distributed optimization, topology

1. INTRODUCTION

Consider the following distributed optimization problem over N networked computing nodes (henceforth called workers) with node-specific cost functions and privately available data

$$\min_{\mathbf{x}} \quad \sum_{i=1}^{N} f_i(\mathbf{x}) + h(\mathbf{x}) \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^p$ is the global decision variable, f_i denotes the local cost function at node i and h a (not necessarily smooth) regularizer. The goal is to find the optimal solution by cooperatively solving the per-worker subproblems. This setting arises frequently in estimation, learning and control tasks [1–4, 17]. Among various solvers, alternating direction method of multiplier (ADMM) has gained popularity for its decomposability and flexibility [2, 4]. Typically, ADMM-based solvers come in two formats: i) *centralized*, where a single master node is connected all workers [2]; and ii) *decentralized*, where no master is present and workers talk to single-hop neighbors only [1, 5]. Until recently [6], there has been no principled approaches to dealing with multiple masters. However, [6] dealt with *synchronous* algorithms that require global coordination, thus challenging their implementation.

Asynchronous algorithms may be more appealing in some settings, especially for decentralized optimization, since they do not need global coordination and consequently are more efficient when workers differ significantly in computing speed [11, 12, 16]. With prior art dealing with either *centralized* or *decentralized* operations, the main contribution of this work to develop an ADMM-based asynchronous decentralized solver of (1) using multiple masters is well motivated.

Related work. From the plethora of distributed optimization schemes, we will focus on ADMM-based ones, which can be split in two categories: synchronous and asynchronous.

Synchronous distributed optimization has been studied extensively for decades; see e.g., [4]. These methods may be *centralized* [2] or *decentralized* [1, 5], depending on whether a master (fusion center) is present or not. This setup of multiple masters remains a largely uncharted territory. Progress was made recently in [7] where a cluster of workers are handled together, but no explicit means to accommodate multiple masters. A novel synchronous approach that is capable of handling multiple masters was proposed in [6].

Similarly, asynchronous algorithms are either *centralized* or *decentralized*. Centralized ones are popular, and are easier to analyze and implement [8–11], but the single master operation faces single-point failure and bottleneck related challenges that limit the overall system performance. Consequently, decentralized alternatives have been developed [11, 12]. But no method is available to accommodate multiple masters except for the asynchronous version of [13].

Contributions. We classify our contributions as follows: C1) we develop AH-ADMM that is able to accommodate multiple masters; C2) we show that AH-ADMM enables topology-aware acceleration of decentralized algorithms without changing the underlying network; and C3) we establish convergence of AH-ADMM for nonconvex functions.

The rest of the paper is organized as follows. Sec. 2 presents detailed derivation of AH-ADMM, Sec 3 deals with topology-aware acceleration, Sec. 4 presents convergence analysis, Sec. 5 shows numerical tests and Sec. 6 concludes this paper.

Meng Ma and G. B. Giannakis were supported in part by NSF grants 1500713, 1508993, 1509040, 1711471, 1508993. Jineng Ren and J. Haupt were supported in part by DARPA YFA, Award N66001-14-1-4047.

2. ASYNCHRONOUS HYBRID ADMM

This section provides background about optimization with multiple masters, followed by development of AH-ADMM.

2.1. Accommodating multiple masters

The presence of multiple masters makes communication among workers much more complicated. Some workers may be connected to masters and also other worker, thus necessitating exchange of information to both. We refer to such communication constraints as *hybrid constraints*.

Communication constraints can be effectively described by graphs. The centralized case corresponds to a star graph, with the master at the center and workers around. The decentralized case can be described by a connected graph, whose nodes corresponds to workers and edges represent communication between neighbors. Hybrid constraints are best depicted by *hypergraphs*. Each master is described by a hyperedge consisting of all its connected workers. Fig. 1 is an example of hybrid constraints with a master connected to workers 1, 2 and 3.

2.2. Problem formulation

A hypergraph is a tuple $\mathcal{H} := (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, N\}$ is the vertex set, and $\mathcal{E} := \{\mathcal{E}_i | \mathcal{E}_i \subset \mathcal{V}\}$ denotes the set of hyperedges. Let N be the number of nodes, and $M = |\mathcal{E}|$ the number of (hyper)edges. A vertex i and an edge $\mathcal{E}_j$ are said to be *incident* if $i \in \mathcal{E}_j$. A simple edge can be seen as an hyperedge consisting of two nodes.

By creating copies $\mathbf{x}_i$ at each node and assigning each hyperedge $\mathcal{E}_i$ an auxiliary variable $\mathbf{z}_i$, we can write hybrid constraints uniformly as $\mathbf{x}_i = \mathbf{z}_j,\ \forall i \in \mathcal{E}_j$. Let T be the number of constraints. Consider now vectors $\mathbf{x} \in \mathbb{R}^{Np}$, $\mathbf{z} \in \mathbb{R}^{Mp}$ concatenating $\{\mathbf{x}_i\}, \{\mathbf{z}_j\}$, and also matrices $\tilde{\mathbf{A}} \in \mathbb{R}^{T\times N}$, $\tilde{\mathbf{B}} \in \mathbb{R}^{T\times M}$ whose t-th row $\tilde{A}_{ti} = 1$, $\tilde{B}_{tj} = 1$ corresponds to t-th constraint $\mathbf{x}_i = \mathbf{z}_j,\ i \in \mathcal{E}_j$. Upon defining $\mathbf{A} = \tilde{\mathbf{A}} \otimes \mathbf{I}_p$, $\mathbf{B} = \tilde{\mathbf{B}} \otimes \mathbf{I}_p$, where $\otimes$ is the Kronecker product, problem (1) can be formulated as

$$\min_{\mathbf{x}_i} \sum_{i=1}^{N} f_i(\mathbf{x}_i) + \sum_{j=1}^{M} h_j(\mathbf{z}), \quad \text{s. to } \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{z} = \mathbf{0}. \tag{2}$$

where $h_j := h/M$. Let $\tilde{\mathbf{C}} \in \mathbb{R}^{N\times M}$ be the signless incidence matrix of the hypergraph, meaning $\tilde{C}_{ij} = 1$ if node i and edge j are incident, and $\tilde{C}_{ij} = 0$ otherwise. Let D_i denote the degree of node i (number of incident edges), E_j the degree of hyperedge j (number of incident nodes), and diagonal matrix $\tilde{\mathbf{D}} \in \mathbb{R}^{N\times N}$ and $\tilde{\mathbf{E}} \in \mathbb{R}^{M\times M}$ collecting $\{D_i\}_{i=1}^N$ and $\{E_j\}_{j=1}^M$, respectively. With $\mathbf{C} := \tilde{\mathbf{C}} \otimes \mathbf{I}_p$, $\mathbf{D} := \tilde{\mathbf{D}} \otimes \mathbf{I}_p$, one can show that $\mathbf{A}^\top\mathbf{A} = \mathbf{D}$, $\mathbf{B}^\top\mathbf{B} = \mathbf{E}$ and $\mathbf{A}^\top\mathbf{B} = \mathbf{C}$ (see [6] for the proof).

Example In Fig. 1, we have $N = 4$, $M = 2$, and $T = 5$. Specifically, the constraints are $x_i = z_1, i = 1, 2, 3$; $x_3 = z_2$, $x_4 = z_2$. These are expressible in compact form as $\mathbf{A}\mathbf{x} = \mathbf{B}\mathbf{z}$,

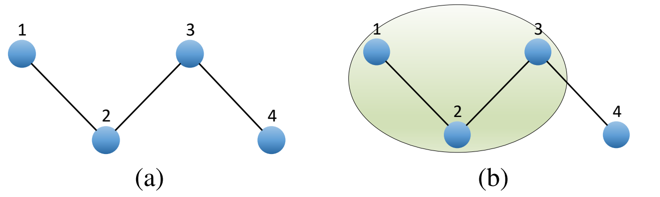

Fig. 1: An example of hybrid constraints described by a hypergraph. (a) is the underlying graph, and (b) is a hypergraph where the shaded ellipsoid denotes an hyperedge.

where $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ are given by

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \tilde{\mathbf{B}} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}. \tag{3}$$

2.3. Asynchronous Hybrid ADMM

The setup of AH-ADMM is similar to that of synchronous hybrid ADMM [6], except that the former has to cope with asynchrony. The high-level description of AH-ADMM is as follows. i) Each worker solves its subproblem individually and communicates the solution to *incident* master(s), then starts waiting until responses from its masters arrive; ii) each master updates its solution whenever updates from a preselected number of workers have been received. Received values can be outdated; and iii) after updating, each master sends results to all *received* workers it received updates from, in order for them to update their associated dual variables.

To describe the process mathematically, we first introduce some definitions. Let k denote a *virtual* iteration counter that is increased by 1 when *any* master finishes its update. By definition, at iteration k, there is exactly *one* master updating, denoted by j_k. Let $\mathcal{A}^k$ be the active set at iteration k, containing received workers of master j_k.

The updates of AH-ADMM are obtained by optimizing the augmented Lagrangian with respect to corresponding variables (see [6] for details). However, due to asynchrony, $\mathbf{x}_i^{k+1}$ may depend on outdated values $\hat{\mathbf{z}}_j$ as incident masters could have updated again while worker i is computing. On the other hand, the update of $\mathbf{z}_{j_k}^{k+1}$ always has access to latest values of $\{\mathbf{x}_i^{k+1}, i \in \mathcal{A}^k\}$, thanks to the updating order, and likewise for $\boldsymbol{\lambda}_t^{k+1}$. All other values not involved remain the same. Equivalently, updates of active workers $i \in \mathcal{A}^k$ can be seen as occuring right before the update of master j_k. Specifically, the updates are

$$\mathbf{x}_i^{k+1} = \arg\min_{\mathbf{x}} f_i(\mathbf{x}_i) + \mathbf{x}_i^\top \mathbf{u}_i^k + \frac{\rho}{2}\Big(D_i\|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^\top \sum_{i\sim j} \hat{\mathbf{z}}_j\Big), i \in \mathcal{A}^k \tag{4a}$$

Algorithm 1: AH-ADMM: worker side

```
Input: $\rho$, $\boldsymbol{\lambda}^0, \mathbf{z}^0$
while stop criterion not satisfied do
    for worker i = 1,2,..., N do
        update $\mathbf{x}_i$ by (4a)
        while not enough masters are received do
            wait
        update $\boldsymbol{\lambda}_t$ by (4c)
        send $\{\mathbf{x}_i, \boldsymbol{\lambda}_t\}$ to incident masters and
          neighboring workers
```

Algorithm 2: AH-ADMM: master side

```
Input: $\rho$, $\boldsymbol{\lambda}^0, \mathbf{x}^0, \mathbf{z}^0$
while stop criterion not satisfied do
    for master j = 1,2,..., M do
        while not enough workers are received do
            wait
        update $\mathbf{z}_j$ by (4b)
        send $\mathbf{z}_j$ to all received workers
```

$$\mathbf{z}_{j_k}^{k+1} = \arg\min_{\mathbf{z}_{j_k}} h_{j_k}(\mathbf{z}_{j_k}) - \mathbf{z}_{j_k}^\top \mathbf{v}_{j_k}^k + \frac{\gamma}{2}\|\mathbf{z}_{j_k} - \mathbf{z}_{j_k}^k\|^2 + \frac{\rho}{2}\Big(E_{j_k}\|\mathbf{z}_{j_k}\|^2 - 2\mathbf{z}_{j_k}^\top \sum_{i\in\mathcal{A}^k} \mathbf{x}_i^{k+1}\Big) \tag{4b}$$

$$\boldsymbol{\lambda}_t^{k+1} = \boldsymbol{\lambda}_t^k + \rho(\mathbf{A}\mathbf{x}_i^{k+1} - \mathbf{B}\mathbf{z}_j^{k+1}),\ t = \mathcal{T}(i,j) \tag{4c}$$

where $\mathcal{T}(i,j) = t$ describes the mapping of worker i and master j to the associated multiplier $\boldsymbol{\lambda}_t$, while $\mathbf{u} := [\mathbf{u}_1^\top, \ldots, \mathbf{u}_N^\top]^\top = \mathbf{A}^\top\boldsymbol{\lambda}$ and $\mathbf{v} := [\mathbf{v}_1^\top, \ldots, \mathbf{v}_M^\top]^\top = \mathbf{B}^\top\boldsymbol{\lambda}$ are change of variables. We include a proximal term in (4b) to guarantee the convergence of AH-ADMM, see Theorem 1. The AH-ADMM algorithm is better understood by considering masters and workers separately, see Algorithms 1 and 2.

3. TOPOLOGY-AWARE ACCELERATION

Hybrid ADMM generally converges faster than its decentralized counterparts [6], which is not surprising due to the presence of multiple masters. When no masters are available, AH-ADMM reduces to AD-ADMM, no longer providing any performance gain. Therefore, we are more interested in the question "*Can we benefit from AH-ADMM even if no master exists?*" The answer is *yes*. The idea is to create *virtual masters* inside workers and employ AH-ADMM afterwards. This technique transforms a decentralized optimization problem to one that can be tackled using hybrid ADMM without physically adding nodes or edges.

The first step to apply this technique is to select workers as hosts that serves as *virtual masters* inside. Subsequently,

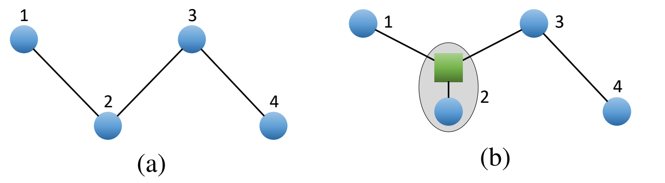

Fig. 2: Illustration of topology-aware acceleration. The underlying graph is (a), and node 2 is selected as host to serve as virtual master, depicted by the square. The shaded ellipsoid in (b) plays the same role as node 2 in (a).

we connect each virtual master to all neighbors of its host and the host itself, which can be done using all edges of the host. Repeating this procedure creates multiple masters, with the help of whom it becomes possible to employ AH-ADMM subsequently; see also Fig. 2 for an example. Notice that in the process, no *physical* nodes or edges have been deployed since virtual masters are just *logical* entities. However, host nodal updates increase complexity. AH-ADMM is also flexible to allow the deployment of any number of virtual masters, a balanced means of boosting performance.

4. CONVERGENCE ANALYSIS

In this section, we analyze the convergence of AH-ADMM. Let τ be the maximum delay, which means that every worker performs update at least once during τ iterations; and $F^\star$ the optimal objective value of (2). Inspired by [9], the following theorem establishes the convergence of AH-ADMM. The analysis in [9] assumes a single master. It is nontrivial to extend the reasoning in [9] to multiple masters, because the dual variables in $\boldsymbol{\lambda}$ are coupled.

Theorem 1. *Suppose that the maximum delay is finite $\tau < \infty$, f_i is twice differentiable and its gradient ∇f_i is Lipschitz with constant L, while h is proper and convex. Then sequences $\{\mathbf{x}_i^k\}_{i=1}^N$, $\{\mathbf{z}_j^k\}_{j=1}^M$ and $\{\boldsymbol{\lambda}_t^k\}_{t=1}^T$ converge to some limit points satisfying the KKT condition of problem* (2)*, provided that*

$$0 \le L_\rho(\mathbf{x}^0, \mathbf{z}^0, \boldsymbol{\lambda}^0) - F^\star < \infty \tag{5}$$

$$\rho > \frac{D_{min} + L + \sqrt{(D_{min} + L)^2 + 8L^2 D_{min}}}{2D_{min}} \tag{6}$$

$$\gamma \ge \frac{S_m(\tau-1)^2\rho^2 - \rho E_{min}}{2} \tag{7}$$

where $S_m = \max_k |\mathcal{A}^k|$, $D_{min} = \min_i D_i$, $E_{min} = \min_j E_j$.

Theorem 1 shows that the solution given by AH-ADMM is guaranteed to converge to some KKT points of (2), as long as ρ and γ are large enough. Note that f_i does not need to be convex. Different from [11, 12], Theorem 1 does not need assumptions of random activation of workers, thus being able to cope also with deterministic settings.

Theorem 1 implies that ρ and γ should be sufficiently large to guarantee the convergence of AH-ADMM. Speicifically, (7)

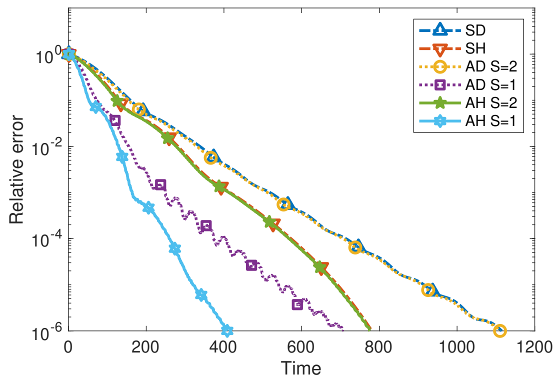

Fig. 3: Relative error of SD-ADMM (SD), SH-ADMM (SH), AD-ADMM (AD) and AH-ADMM (AH) with different threshold (S) vs. wall clock time.

suggests that a large γ is needed when the maximum delay is large. Also, (6) implies that ρ can be smaller when the cost functions are smoother.

5. NUMERICAL EXPERIMENTS

In this section, we carry out numerical experiments to test the acceleration merits of AH-ADMM, and compare with asynchronous decentralized ADMM (AD-ADMM) [11, 12], synchronous decentralized ADMM (SD-ADMM) [1], and synchronous hybrid ADMM [6].

Different from existing works [11, 12, 16], our method does not assume every worker can activate each iteration. For this reason, it can deal with more general settings. For example, each worker has a positive activation probability in [11, 16] at each iteration, while one edge is randomly selected each time in [12]. Both assumptions exclude the case of deterministic activation patterns, as verified in the following experiment.

In our experiment, the speed of each worker is deterministic and initialized randomly by drawing from a uniform distribution $U[1, 10]$, so that the fastest worker can be 10 times faster than the slowest one, which also ensures bounded delays. We evaluate the performance by plotting its relative error $\frac{\|\mathbf{x}^k-\mathbf{x}^\star\|}{\|\mathbf{x}^0-\mathbf{x}^\star\|}$ against wall clock time. The optimal solution $\mathbf{x}^\star$ can either be computed directly if it admits a closed-form solution, or be obtained using CVX [14, 15]. We also show average working and waiting time of workers to demonstrate the effects of the threshold of masters.

The decentralized *sparse compressed sensing* we tested aims at reconstructing a sparse unknown vector $\mathbf{x} \in \mathbb{R}^p$ through nodal measurements $\mathbf{b}_i = \mathbf{H}_i\mathbf{x} + \mathbf{e}_i,\ i = 1, \ldots, N$, where $\mathbf{H}_i \in \mathbb{R}^{n_i \times p}$ is the sensing matrix of node i and $\mathbf{e}_i$ represents a vector of i.i.d. Gaussian noise. When $p > \sum_{i=1}^N n_i$, there are more unknowns than measurements. The sparsity of $\mathbf{x}$ suggests solving a L_1-regularized least-squares problem

$$\min_{\mathbf{x}} \quad \sum_{i=1}^{N} \frac{1}{2}\|\mathbf{H}_i\mathbf{x} - \mathbf{b}_i\|^2 + \mu\|\mathbf{x}\|_1 \tag{8}$$

where $\mu > 0$ is the regularization parameter.

We consider a ring graph of 10 nodes, and set $n_i = 3$ and $p = 40$ such that $p > \sum_{i=1}^N n_i$. The entries of $\mathbf{H}_i$ are generated from the standard Gaussian distribution $N(0, 1)$, and then normalized so that $\|\mathbf{H}_i\|_2 = 1$. The unknown vector $\mathbf{x}$ is drawn from $N(0, 1)$ with 10% nonzero entries, and subsequently $\mathbf{b}_i$ is generated. Since (8) admits no closed-form solution, we solve it using CVX to obtain the optimal solution $\mathbf{x}^\star$. We set $\gamma = 0$ and tune ρ to be nearly optimal.

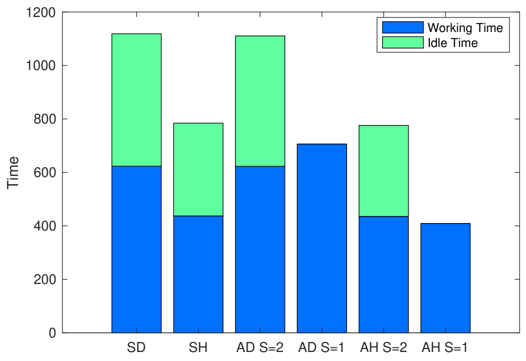

Fig. 4: Average working and waiting time of workers of SD-ADMM, SH-ADMM, AD-ADMM and AH-ADMM.

Fig. 3 depicts the relative error against wall clock time. When $S = 2$, AH-ADMM and AD-ADMM are almost equivalent to their synchronous counterparts, while AD-ADMM with $S = 1$ corresponds to the case that each node updates as soon as it receives information from any neighbor, thus eliminating waiting time. One can immediately make two observations: a) asynchronous approaches converge at least as fast as, if not faster, than their decentralized counterparts; and b) hybrid approaches always outperform their decentralized counterparts, showcasing their promising potential.

Fig. 4 compares average working and waiting time of different approaches. Again, we notice that a) asynchronous approaches reduce or even eliminate waiting time, thus improving efficiency; and b) hybrid approaches consume significantly less working and waiting.

6. CONCLUSIONS

This paper presents an asynchronous distributed optimization algorithm, capable of handling multiple masters, AH-ADMM, which not only broadens the applicability of ADMM, but also yields a technique that significantly accelerates the convergence of decentralized ADMM without changing the underlying topology. A convergence result is provided and numerical experiments are performed to compare AH-ADMM against decentralized approaches.

7. REFERENCES

[1] G. B. Giannakis, Q. Ling, G. Mateos, I. D. Schizas, and H. Zhu, "Decentralized Learning for Wireless Communications and Networking," in *Splitting Methods in Communication, Imaging, Science, and Engineering*, R. Glowinski, S. J. Osher, and W. Yin, Eds. Springer, Cham, 2016, pp. 461–497.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Mirection Method of Multipliers," *Found. Trends® Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[3] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and Cooperation in Networked Multi-Agent Systems," *Proc. IEEE*, vol. 95, no. 1, pp. 215–233, Jan. 2007.

[4] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1989, vol. 23.

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

[6] M. Ma, A. N. Nikolakopoulos, and G. B. Giannakis, "Fast Decentralized Optimization over Networks," *arXiv:1804.02425*, Apr. 2018.

[7] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Explicit Convergence Rate of a Distributed Alternating Direction Method of Multipliers," *IEEE Trans. Autom. Control*, vol. 61, no. 4, pp. 892–904, Apr. 2016.

[8] R. Zhang and J. Kwok, "Asynchronous Distributed ADMM for Consensus Optimization," in *Proceedings of International Conference on Machine Learning*, Beijing, China, Jun. 2014, pp. 1701–1709.

[9] T. H. Chang, M. Hong, W. C. Liao, and X. Wang, "Asynchronous Distributed ADMM for Large-Scale Optimization—Part I: Algorithm and Convergence Analysis," *IEEE Trans. Signal Process.*, vol. 64, no. 12, pp. 3118–3130, Jun. 2016.

[10] M. Hong, "A Distributed, Asynchronous and Incremental Algorithm for Nonconvex Optimization: An ADMM Approach," *IEEE Trans. Control Netw. Syst.*, to be published.

[11] Z. Peng, Y. Xu, M. Yan, and W. Yin, "ARock: An Algorithmic Framework for Asynchronous Parallel Coordinate Updates," *SIAM J. Sci. Comput.*, vol. 38, no. 5, pp. A2851–A2879, Jan. 2016.

[12] E. Wei and A. Ozdaglar, "On the O(1/k) Convergence of Asynchronous Distributed Alternating Direction Method of Multipliers," in *Proceedings of Global Conference on Signal and Information Processing*, Austin, TX, Dec. 2013, pp. 551–554.

[13] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Asynchronous Distributed Optimization using a Randomized Alternating Direction Method of Multipliers," in *Proceedings of Conference on Decision and Control*, Florence, Italy, Dec. 2013, pp. 3671–3676.

[14] M. Grant and S. Boyd, "CVX: Matlab Software for Disciplined Convex Programming, version 2.1," http://cvxr.com/cvx, Mar. 2014.

[15] M. C. Grant and S. P. Boyd, "Graph Implementations for Nonsmooth Convex Programs," in *Recent Advances in Learning and Control*, ser. Lecture Notes in Control and Information Sciences. Springer, London, 2008, pp. 95–110.

[16] T. Wu, K. Yuan, Q. Ling, W. Yin, and A. H. Sayed, "Decentralized Consensus Optimization with Asynchrony and Delays," *IEEE Trans. Signal Inf. Process. Netw.*, vol. PP, no. 99, pp. 1–1, 2017.

[17] J. Ren, X. Li, and J. Haupt, "Communication-Efficient Distributed Optimization for Sparse Learning via Two-way Truncation," in *Proceedings of International Workshop on Computational Advances in Multi-Sensor Adaptive Processing*, Curacao, Dec. 2017, pp. 1–5.