BI-ALTERNATING DIRECTION METHOD OF MULTIPLIERS OVER GRAPHS

Guoqiang Zhang and Richard Heusdens

Group of Circuits and Systems (CAS)
Delft University of Technology
Delft, the Netherlands Email: {g.zhang-1,r.heusdens}@tudelft.nl

ABSTRACT

In this paper, we extend the bi-alternating direction method of multipliers (BiADMM) designed on a graph of two nodes to a graph of multiple nodes. In particular, we optimize a sum of convex functions defined over a general graph, where every edge carries a linear equality constraint. In designing the new algorithm, an augmented primal-dual Lagrangian function is carefully constructed which naturally captures the associated graph topology. We show that under both the synchronous and asynchronous updating schemes, the extended BiADMM has the convergence rate of $O(1/K)$ (where K denotes the iteration index) for general closed, proper and convex functions. As an example, we apply the new algorithm for distributed averaging. Experimental results show that the new algorithm remarkably outperforms the state-of-the-art methods.

Index Terms— Distributed optimization, alternating direction method of multipliers, bi-alternating direction of multipliers

1. INTRODUCTION

In this paper, we consider solving a decomposable optimization problem defined over a graphic model $G = (\mathcal{V}, \mathcal{E})$:

$$\min_{\boldsymbol{x}} \sum_{i\in\mathcal{V}} f_i(\boldsymbol{x}_i) \quad \text{s.t.} \quad \boldsymbol{A}_{i\to j}\boldsymbol{x}_i + \boldsymbol{A}_{j\to i}\boldsymbol{x}_j = \boldsymbol{c}_{ij}\ \forall (i,j)\in\mathcal{E}, \quad (1)$$

where $\boldsymbol{x} = [\boldsymbol{x}_1^T, \ldots, \boldsymbol{x}_{|\mathcal{V}|}]^T$, and for every $i \in \mathcal{V}$, $f_i : \mathbb{R}^{n_i} \to \mathbb{R}\cup\{\infty\}$ is a closed, proper and convex function. The above distributed optimization problem has drawn increasing attention due to the demand for big-data processing and easy access to ubiquitous computing units (e.g., a computer, a mobile phone or a senor equipped with CPUs). The basic idea is to have a set of computing units collaborate with each other in a distributed way to complete a complex task. Popular applications include telecommunication [1, 2], wireless sensor networks [3], cloud computing and machine learning [4]. The research challenge is on the design of efficient and robust distributed optimization algorithms for solving (1).

A majority of recent research have been focusing on a specialized form of the above problem (1), where the constraint on every edge $(i, j) \in \mathcal{E}$ reduces to $\boldsymbol{x}_i = \boldsymbol{x}_j$. The above special form is commonly known as the consensus problem in the literature. Classic methods include the dual-averaging algorithm [5], the subgradient algorithm [6], the diffusion adaptation algorithm [7].

In the literature, there is not much progress in solving the general problem (1). One algorithm that can be used for solving (1) is the alternating-direction method of multipliers (ADMM). In order to do so, (1) is firstly reformulated as

$$\min_{\boldsymbol{x},\boldsymbol{z}} f(\boldsymbol{x}) + g(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{c}, \quad (2)$$

where $f(\boldsymbol{x}) = \sum_{i\in\mathcal{V}} f_i(\boldsymbol{x}_i)$, $g(\boldsymbol{z}) = 0$ and $\boldsymbol{z}$ is an introduced auxiliary variable. The graphic structure is implicitly embedded in the two matrices $(\boldsymbol{A}, \boldsymbol{B})$ and the vector $\boldsymbol{c}$. The reformulation essentially converts the problem on a general graph with many nodes to a graph with only two nodes (2), allowing the application of ADMM.

We note that the above approach of reformulating (1) into (2) does not make full use of the graphic structure $G = (\mathcal{V}, \mathcal{E})$ carried in (1). It is of great interest to design an optimization algorithm for the problem (1) directly. By doing so, we might have a deep understanding of distributed optimization over graphs.

Recently, in [8], we have proposed a new algorithm for (2) defined over a graph of two nodes, which is referred to as the *bi-alternating direction method of multipliers* (BiADMM). One advantage of BiADMM is that the algorithm converges by following either the synchronous or asynchronous updating scheme, as opposed to ADMM which has to follow the Gaussian-Seidel procedure at each iteration.

In this paper, we tackle the general problem (1) by extending BiADMM designed on a graph of two nodes to a graph of multiple nodes. Inspired by [8], we construct an augmented primal-dual Lagrangian function for (1) without introducing the auxiliary variable $\boldsymbol{z}$ as in (2). The extended BiADMM solves (1) by iteratively approaching a saddle point of the constructed function. We show that for both the synchronous and asynchronous updating schemes, the new algorithm converges with the rate of $\mathcal{O}(1/K)$ regardless of the graph topology. As an example, we apply the extended BiADMM to distributed averaging. The new algorithm significantly outperforms the state-of-the-art methods in terms of convergence speed.

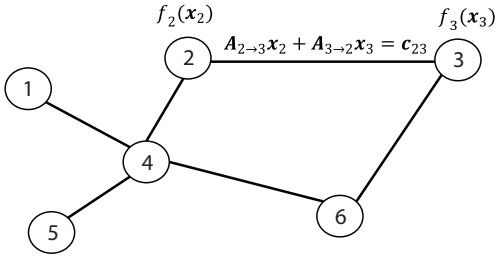

Fig. 1. Demonstration of problem (1), where every edge carries an equality constraint.

This work was supported by the COMMIT program, The Netherlands.

2. PROBLEM REFORMULATION

2.1. Problem assumption

Considering the problem (1), we let $(\boldsymbol{c}_{ij}, \boldsymbol{A}_{i\to j}, \boldsymbol{A}_{j\to i}) \in (\mathbb{R}^{n_{ij}}, \mathbb{R}^{n_{ij}\times n_i}, \mathbb{R}^{n_{ij}\times n_j})$ for every edge $(i,j) \in \mathcal{E}$. We use $\mathcal{N}(i)$ to denote the neighboring set of node i. Also we let the set $\mathcal{V} = \{1, 2, \ldots, m\}$. As a result, $|\mathcal{V}| = m$. Finally, we denote the set of all the directed edges in the graph as $\vec{\mathcal{E}}$. The Lagrangian function associated with (1) can be constructed as

$$L(\boldsymbol{x}, \boldsymbol{\delta}) = \sum_{(i,j)\in\mathcal{E}} \boldsymbol{\delta}_{ij}^T(\boldsymbol{c}_{ij} - \boldsymbol{A}_{i\to j}\boldsymbol{x}_i - \boldsymbol{A}_{j\to i}\boldsymbol{x}_j) + \sum_{i\in\mathcal{V}} f_i(\boldsymbol{x}_i), \quad (3)$$

where $\boldsymbol{\delta}_{ij}$ is the Lagrangian multiplier (or the dual variable) for each constraint $\boldsymbol{A}_{i\to j}\boldsymbol{x}_i + \boldsymbol{A}_{j\to i}\boldsymbol{x}_j = \boldsymbol{c}_{ij}$ in (1). The vector $\boldsymbol{\delta}$ is obtained by stacking the individual variables $\boldsymbol{\delta}_{ij}$, $(i,j) \in \mathcal{E}$. Therefore, $\boldsymbol{x} \in \mathbb{R}^{\sum_i n_i}$ and $\boldsymbol{\delta} \in \mathbb{R}^{\sum_{(i,j)} n_{ij}}$. The Lagrangian function is convex in $\boldsymbol{x}$ for fixed $\boldsymbol{\delta}$, and concave in $\boldsymbol{\delta}$ for fixed $\boldsymbol{x}$. Throughout the rest of the paper, we will make the following (common) assumption:

Assumption 1. *There exists a saddle point $(\boldsymbol{x}^*, \boldsymbol{\delta}^*)$ to the Lagrangian function $L(\boldsymbol{x}, \boldsymbol{\delta})$ such that for all $\boldsymbol{x} \in \mathbb{R}^{\sum_i n_i}$ and $\boldsymbol{\delta} \in \mathbb{R}^{\sum_{(i,j)} n_{ij}}$ we have*

$$L(\boldsymbol{x}^*, \boldsymbol{\delta}) \le L(\boldsymbol{x}^*, \boldsymbol{\delta}^*) \le L(\boldsymbol{x}, \boldsymbol{\delta}^*).$$

2.2. Augmented Primal-Dual Lagrangian Function

In this subsection, we build an augmented primal-dual Lagrangian function by using $\{f_i | i \in \mathcal{V}\}$ and their conjugate functions $\{f_i^* | i \in \mathcal{V}\}$ (see [9] for the definition). We show that the problem (1) can be alternatively solved by finding the saddle point of the newly constructed function.

Before presenting the new function, we firstly introduce a set of variables and notations. In particular, we introduce two variables $\boldsymbol{\lambda}_{i|j} \in \mathbb{R}^{n_{ij}}$ and $\boldsymbol{\lambda}_{j|i} \in \mathbb{R}^{n_{ij}}$ for every edge $(i,j) \in \mathcal{E}$. The subscript $i|j$ indicates that the variable $\boldsymbol{\lambda}_{i|j}$ is owned by node i and related to neighboring node j. We use $\boldsymbol{\lambda}_i$ to denote the vector obtained by vertically concatenating all $\boldsymbol{\lambda}_{i|j}$, $j \in \mathcal{N}(i)$. Therefore, every node i owns two variables $\boldsymbol{x}_i$ and $\boldsymbol{\lambda}_i$. We let $\boldsymbol{A}_i^T$ denote the matrix obtained by horizontally concatenating all $\boldsymbol{A}_{i\to j}^T$, $j \in \mathcal{N}(i)$. The notation $\boldsymbol{M} \succeq \boldsymbol{0}$ (or $\boldsymbol{M} \succ \boldsymbol{0}$) indicates that $\boldsymbol{M}$ is a positive semi-definite matrix (or a positive definite matrix). Suppose $\boldsymbol{M} \succeq \boldsymbol{0}$, we let $\|\boldsymbol{y}\|_{\boldsymbol{M}}^2 = \boldsymbol{y}^T\boldsymbol{M}\boldsymbol{y}$, which generalizes the l_2 norm.

Inspired by [8], we construct an augmented primal-dual Lagrangian function as

$$L_{\mathcal{P}}(\boldsymbol{x}, \boldsymbol{\lambda}) = \sum_{i\in\mathcal{V}}\Big[f_i(\boldsymbol{x}_i) - \sum_{j\in\mathcal{N}(i)} \boldsymbol{\lambda}_{j|i}^T(\boldsymbol{A}_{i\to j}\boldsymbol{x}_i - \boldsymbol{c}_{ij}) - f_i^*(\boldsymbol{A}_i^T\boldsymbol{\lambda}_i)\Big] + h_{\mathcal{P}}(\boldsymbol{x}, \boldsymbol{\lambda}), \quad (4)$$

where $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_1^T, \boldsymbol{\lambda}_2^T, \ldots, \boldsymbol{\lambda}_m^T]^T$ and

$$\boldsymbol{A}_i^T\boldsymbol{\lambda}_i = \sum_{j\in\mathcal{N}(i)} \boldsymbol{A}_{i\to j}^T\boldsymbol{\lambda}_{i|j} \quad i \in \mathcal{V}$$

$$h_{\mathcal{P}}(\boldsymbol{x}, \boldsymbol{\lambda}) = \sum_{(i,j)\in\mathcal{E}}\Big(\frac{1}{2}\|\boldsymbol{A}_{i\to j}\boldsymbol{x}_i + \boldsymbol{A}_{j\to i}\boldsymbol{x}_j + \boldsymbol{c}_{ij}\|_{\boldsymbol{P}_{p,ij}}^2 - \frac{1}{2}\|\boldsymbol{\lambda}_{i|j} - \boldsymbol{\lambda}_{j|i}\|_{\boldsymbol{P}_{d,ij}}^2\Big), \quad (5)$$

where $\mathcal{P}$ is a set of positive definite matrices defined as

$$\mathcal{P} = \{\boldsymbol{P}_{p,ij} \succ 0, \boldsymbol{P}_{d,ij} \succ 0 | (i,j) \in \mathcal{E}\}.$$

For every edge $(i,j) \in \mathcal{E}$, $\boldsymbol{\lambda}_{i|j}$ and $\boldsymbol{\lambda}_{j|i}$ essentially substitute the role of $\boldsymbol{\delta}_{ij}$ as in (3). It is not difficult to show that $L_{\mathcal{P}}$ is convex in $\boldsymbol{x}$ for fixed $\boldsymbol{\lambda}$ and concave in $\boldsymbol{\lambda}$ for fixed $\boldsymbol{x}$.

Next we establish the relation between the original problem (1) and the constructed function (4):

Theorem 1 (Saddle point theorem)**.** *If $\boldsymbol{x}^*$ solves the original problem (1), there exists $\boldsymbol{\lambda}^*$ such that $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ is a saddle point of $L_{\mathcal{P}}(\boldsymbol{x}, \boldsymbol{\lambda})$. Conversely, if $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ is a saddle point of $L_{\mathcal{P}}(\boldsymbol{x}, \boldsymbol{\lambda})$, then $\boldsymbol{x}^*$ solves the original problem.*

The proof for the above theorem is similar to the one in [8] for a graph of two nodes. In the rest of the paper, we consider solving the following min-max problem

$$(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \arg\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} L_{\mathcal{P}}(\boldsymbol{x}, \boldsymbol{\lambda}). \quad (6)$$

We will explain in next section how to iteratively approach the saddle point $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ in a distributed manner.

3. BI-ALTERNATING DIRECTION METHOD OF MULTIPLIERS

In this section, we present BiADMM to iteratively find a saddle point of $L_{\mathcal{P}}$. We propose both the synchronous and asynchronous updating schemes for BiADMM. Finally, we consider simplifying the computation of BiADMM per iteration.

3.1. Synchronous updating scheme

The synchronous scheme refers to the operation that at each iteration, all the variables receives their new estimates by using the most recent information from their neighbors. Suppose $(\hat{\boldsymbol{x}}^k, \hat{\boldsymbol{\lambda}}^k)$ is the estimate obtained from the kth iteration, $k \ge 0$. The new estimate $(\hat{\boldsymbol{x}}_i^{k+1}, \hat{\boldsymbol{\lambda}}_i^{k+1})$ for every $i \in \mathcal{V}$ is computed as

$$\Big(\hat{\boldsymbol{x}}_i^{k+1}, \hat{\boldsymbol{\lambda}}_i^{k+1}\Big) = \arg\min_{\boldsymbol{x}_i}\max_{\boldsymbol{\lambda}_i} L_{\mathcal{P}}\Big(\Big[\ldots, \hat{\boldsymbol{x}}_{i-1}^{k,T}, \boldsymbol{x}_i^T, \hat{\boldsymbol{x}}_{i+1}^{k,T}, \ldots\Big]^T, \Big[\ldots, \hat{\boldsymbol{\lambda}}_{i-1}^{k,T}, \boldsymbol{\lambda}_i^T, \hat{\boldsymbol{\lambda}}_{i+1}^{k,T}, \ldots\Big]^T\Big) \quad i \in \mathcal{V}. \quad (7)$$

By inserting the expression (4) for $L_{\mathcal{P}}(\boldsymbol{x}, \boldsymbol{\lambda})$ into (7), the updating expression can be simplified as

$$\hat{\boldsymbol{x}}_i^{k+1} = \arg\min_{\boldsymbol{x}_i}\Bigg[\sum_{j\in\mathcal{N}(i)} \frac{1}{2}\Big\|\boldsymbol{A}_{i\to j}\boldsymbol{x}_i + \boldsymbol{A}_{j\to i}\hat{\boldsymbol{x}}_j^k - \boldsymbol{c}_{ij}\Big\|_{\boldsymbol{P}_{p,ij}}^2 - \boldsymbol{x}_i^T\Bigg(\sum_{j\in N(i)} \boldsymbol{A}_{i\to j}^T\hat{\boldsymbol{\lambda}}_{j|i}^k\Bigg) + f_i(\boldsymbol{x}_i)\Bigg] \quad i \in \mathcal{V} \quad (8)$$

$$\hat{\boldsymbol{\lambda}}_i^{k+1} = \arg\min_{\boldsymbol{\lambda}_i}\Bigg[\sum_{j\in\mathcal{N}(i)}\Bigg(\frac{1}{2}\Big\|\boldsymbol{\lambda}_{i|j} - \hat{\boldsymbol{\lambda}}_{j|i}^k\Big\|_{\boldsymbol{P}_{d,ij}}^2 + \boldsymbol{\lambda}_{i|j}^T\boldsymbol{A}_{j\to i}\hat{\boldsymbol{x}}_j^k - \boldsymbol{\lambda}_{i|j}^T\boldsymbol{c}_{ij}\Bigg) + f_i^*(\boldsymbol{A}_i^T\boldsymbol{\lambda}_i)\Bigg] \quad i \in \mathcal{V}. \quad (9)$$

(8)-(9) suggests that the computation of $\hat{\boldsymbol{x}}_i^{k+1}$ and $\hat{\boldsymbol{\lambda}}_i^{k+1}$ can be carried out on node i separately. This property is due to the fact that $\boldsymbol{x}_i$ and $\boldsymbol{\lambda}_i$ are not directly related in $L_{\mathcal{P}}(\boldsymbol{x}, \boldsymbol{\lambda})$, thus simplifying the computational procedure.

3.2. Asynchronous updating scheme

The asynchronous scheme refers to the operation that at each iteration, only the variables associated with one node update their estimates while all other variables keep their estimates fixed. Suppose

at iteration k, node i is activated in the graph. $(\hat{\boldsymbol{x}}^{k+1}, \hat{\boldsymbol{\lambda}}^{k+1})$ can be obtained as

$$\Big(\hat{\boldsymbol{x}}_i^{k+1}, \hat{\boldsymbol{\lambda}}_i^{k+1}\Big) = \arg\min_{\boldsymbol{x}_i} \max_{\boldsymbol{\lambda}_i} L_{\mathcal{P}}\Big(\Big[\ldots, \hat{\boldsymbol{x}}_{i-1}^{k,T}, \boldsymbol{x}_i^T, \hat{\boldsymbol{x}}_{i+1}^{k,T}, \ldots\Big]^T, \Big[\ldots, \hat{\boldsymbol{\lambda}}_{i-1}^{k,T}, \boldsymbol{\lambda}_i^T, \hat{\boldsymbol{\lambda}}_{i+1}^{k,T}, \ldots\Big]^T\Big) \quad (10)$$

$$\Big(\hat{\boldsymbol{x}}_j^{k+1}, \hat{\boldsymbol{\lambda}}_j^{k+1}\Big) = \Big(\hat{\boldsymbol{x}}_j^{k}, \hat{\boldsymbol{\lambda}}_j^{k}\Big) \qquad j \in \mathcal{V}, j \neq i, \quad (11)$$

where the estimates related with $j \in \mathcal{V}, j \neq i$, remain fixed.

In practice, one can either randomly choose a node from the graph or follow a predefined order of the nodes for parameter-update. We assume that given enough time, every node will be activated for parameter-updating. To speed up convergence, the frequencies of every node being activated should be roughly balanced.

3.3. Reducing computational complexities

We note from Subsection 3.1 and 3.2 that for both the synchronous and asynchronous schemes, each activated node i has to perform two minimizations: one for the primal variable $\boldsymbol{x}_i$ and the other one for the dual variable $\boldsymbol{\lambda}_i$. In this subsection, we identify a condition under which the two minimizations can be reduced to one minimization, thus reducing the computational complexities.

Proposition 1. *Consider computing $(\hat{\boldsymbol{x}}_i^{k+1}, \hat{\boldsymbol{\lambda}}_i^{k+1})$ for node i by following either (7) or (10). If the matrix $\boldsymbol{P}_{d,ij}$ for every neighbor $j \in \mathcal{N}(i)$ is chosen to be $\boldsymbol{P}_{d,ij} = \boldsymbol{P}_{p,ij}^{-1}$, then for every $j \in \mathcal{N}(i)$, there is*

$$\hat{\boldsymbol{\lambda}}_{i|j}^{k+1} = \hat{\boldsymbol{\lambda}}_{j|i}^{k} + \boldsymbol{P}_{p,ij}(\boldsymbol{c}_{ij} - \boldsymbol{A}_{j\to i}\hat{\boldsymbol{x}}_j^k - \boldsymbol{A}_{i\to j}\hat{\boldsymbol{x}}_i^{k+1}). \quad (12)$$

4. CONVERGENCE ANALYSIS

In this section, we show that BiADMM has a convergence rate $\mathcal{O}(K^{-1})$ under either the synchronous or asynchronous scheme.

4.1. Preliminary

In order to analyze the algorithm convergence, we first have to select the parameter set $\mathcal{P}$ properly. We impose a condition on each pair of matrices $(\boldsymbol{P}_{p,ij} \succ \boldsymbol{0}, \boldsymbol{P}_{d,ij} \succ \boldsymbol{0})$, $(i,j) \in \mathcal{E}$, in $L_{\mathcal{P}}$:

Condition 1. *In the function $L_{\mathcal{P}}$, each pair of matrices $(\boldsymbol{P}_{p,ij}, \boldsymbol{P}_{d,ij})$ can be represented in terms of a triplet $(\boldsymbol{P}_{b,ij}, \Delta\boldsymbol{P}_{p,ij}, \Delta\boldsymbol{P}_{d,ij})$ as follows:*

$$\boldsymbol{P}_{p,ij} = \boldsymbol{P}_{b,ij} + \Delta\boldsymbol{P}_{p,ij} \quad (i,j) \in \mathcal{E} \quad (13)$$

$$\boldsymbol{P}_{d,ij} = \boldsymbol{P}_{b,ij}^{-1} + \Delta\boldsymbol{P}_{d,ij} \quad (i,j) \in \mathcal{E}, \quad (14)$$

where $\boldsymbol{P}_{b,ij} \succ \boldsymbol{0}$, $\Delta\boldsymbol{P}_{p,ij} \succeq \boldsymbol{0}$ and $\Delta\boldsymbol{P}_{d,ij} \succeq \boldsymbol{0}$.

(13)-(14) implies that $\boldsymbol{P}_{p,ij}$ and $\boldsymbol{P}_{d,ij}$ can not be chosen arbitrarily for our convergence analysis. If $\boldsymbol{P}_{p,ij}$ is small, then $\boldsymbol{P}_{d,ij}$ has to be chosen big enough to make (13)-(14) hold, and vice versa. One special setup for $(\boldsymbol{P}_{p,ij}, \boldsymbol{P}_{d,ij})$ is to let $\boldsymbol{P}_{d,ij} = \boldsymbol{P}_{p,ij}^{-1}$. Correspondingly, the triplet in (13)-(14) takes a unique form: $(\boldsymbol{P}_{b,ij}, \Delta\boldsymbol{P}_{p,ij}, \Delta\boldsymbol{P}_{d,ij}) = (\boldsymbol{P}_{p,ij}, \boldsymbol{0}, \boldsymbol{0})$.

Also in order to perform the convergence analysis, we have to define a new objective function

$$p(\boldsymbol{x}, \boldsymbol{\lambda}) = \sum_{i\in\mathcal{V}} \Big[f_i(\boldsymbol{x}_i) + f_i^*(\boldsymbol{A}_i^T\boldsymbol{\lambda}_i)\Big] - \sum_{(i,j)\in\mathcal{E}} \boldsymbol{c}_{ij}^T \frac{(\boldsymbol{\lambda}_{i|j} + \boldsymbol{\lambda}_{j|i})}{2}.$$

The function $p(\boldsymbol{x}, \boldsymbol{\lambda})$ at a saddle point $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ of $L_{\mathcal{P}}$ equals to zero, i.e., $p(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = 0$.

4.2. Synchronous updating scheme

The convergence result for the synchronous BiADMM is as follows:

Theorem 2. *Let the synchronous BiADMM runs for K iterations by following (7). Let $(\bar{\boldsymbol{x}}^K, \bar{\boldsymbol{\lambda}}^K) = (\frac{1}{K}\sum_{k=1}^K \hat{\boldsymbol{x}}^k, \frac{1}{K}\sum_{k=1}^K \hat{\boldsymbol{\lambda}}^k)$. If the set $\mathcal{P}$ is chosen by following (13)-(14), there is*

$$0 \le \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}(i)} \Big[(\bar{\boldsymbol{\lambda}}_{i|j}^K - \boldsymbol{\lambda}_{i|j}^*)^T\Big(\boldsymbol{A}_{j\to i}\bar{\boldsymbol{x}}_j^K - \frac{\boldsymbol{c}_{ij}}{2}\Big) - (\bar{\boldsymbol{x}}_i^K - \boldsymbol{x}_i^*)^T \cdot \boldsymbol{A}_{i\to j}^T\bar{\boldsymbol{\lambda}}_{j|i}^K\Big] + p(\bar{\boldsymbol{x}}^K, \bar{\boldsymbol{\lambda}}^K) \le \mathcal{O}(K^{-1}) \quad (15)$$

and for every directed edge $[i,j] \in \vec{\mathcal{E}}$

$$\lim_{K\to\infty}\Big[\boldsymbol{P}_{b,ij}^{\frac{1}{2}}(\boldsymbol{A}_{i\to j}\hat{\boldsymbol{x}}_i^K + \boldsymbol{A}_{j\to i}\hat{\boldsymbol{x}}_j^{K-1} - \boldsymbol{c}_{ij}) + \boldsymbol{P}_{b,ij}^{-\frac{1}{2}}(\hat{\boldsymbol{\lambda}}_{i|j}^K - \hat{\boldsymbol{\lambda}}_{j|i}^{K-1})\Big] = \boldsymbol{0} \quad (16)$$

$$\lim_{K\to\infty} \Delta\boldsymbol{P}_{p,ij}^{\frac{1}{2}}(\boldsymbol{c}_{ij} - \boldsymbol{A}_{i\to j}\hat{\boldsymbol{x}}_i^K - \boldsymbol{A}_{j\to i}\hat{\boldsymbol{x}}_j^{K-1}) = \boldsymbol{0} \quad (17)$$

$$\lim_{K\to\infty} \Delta\boldsymbol{P}_{d,ij}^{\frac{1}{2}}(\hat{\boldsymbol{\lambda}}_{i|j}^K - \hat{\boldsymbol{\lambda}}_{j|i}^{K-1}) = \boldsymbol{0}, \quad (18)$$

where $\boldsymbol{P}_{b,ij}^{\frac{1}{2}} \succ \boldsymbol{0}$ and $\boldsymbol{P}_{b,ij} = \boldsymbol{P}_{b,ij}^{\frac{1}{2}}\boldsymbol{P}_{b,ij}^{\frac{1}{2}}$. $\Delta\boldsymbol{P}_{p,ij}^{\frac{1}{2}}$ and $\Delta\boldsymbol{P}_{d,ij}^{\frac{1}{2}}$ are defined in a similar manner for every edge $(i,j) \in \mathcal{E}$.

The proof for the above theorem is inspired by the one in [8]. We have mainly used the variational inequalities (VIs) in the proof.

4.3. Asynchronous updating scheme

In this subsection, we characterize the convergence rate of the asynchronous BiADMM. In order to facilitate the analysis, we consider a predefined node-activation strategy (no randomness is involved). Without loss of generality, we suppose at each iteration k, the node $i = \text{mod}(k,m) + 1$ is activated for parameter update, where $\text{mod}(\cdot,\cdot)$ stands for the modulus operation and $m = |\mathcal{V}|$. Then naturally, after a segment of m consecutive iterations, all the nodes will be activated sequentially, one node at each iteration.

To be able to derive the convergence rate, we consider segments of iterations, i.e., $k \in \{lm, lm+1, \ldots (l+1)m-1\}$, where $l \ge 0$. Each segment l consists of m iterations. By applying the mapping $i = \text{mod}(k,m) + 1$, it is immediate that $k = ml$ activates node 1 and $k = (l+1)m - 1$ activates node m. We then have the following convergence result:

Theorem 3. *Let the asynchronous BiADMM runs for K segments of m iterations. That is from $k = 0$ until $k = (K-1)m - 1$. Let $(\check{\boldsymbol{x}}^K, \check{\boldsymbol{\lambda}}^K) = (\frac{1}{K}\sum_{l=1}^K \hat{\boldsymbol{x}}^{lm}, \frac{1}{K}\sum_{l=1}^K \hat{\boldsymbol{\lambda}}^{lm})$. If the set $\mathcal{P}$ is chosen by following (13)-(14), there is*

$$0 \le \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}(i)} \Big[\Big(\check{\boldsymbol{\lambda}}_{i|j}^K - \boldsymbol{\lambda}_{i|j}^*\Big)^T\Big(\boldsymbol{A}_{j\to i}\check{\boldsymbol{x}}_j^K - \frac{\boldsymbol{c}_{ij}}{2}\Big) - \Big(\check{\boldsymbol{x}}_i^K - \boldsymbol{x}_i^*\Big)^T \cdot \boldsymbol{A}_{i\to j}^T\check{\boldsymbol{\lambda}}_{j|i}^K\Big] + p\Big(\check{\boldsymbol{x}}^K, \check{\boldsymbol{\lambda}}^K\Big) \le \mathcal{O}(K^{-1}), \quad (19)$$

and for every $(u,v) \in \mathcal{E}$ where $u < v$

$$\lim_{K\to\infty}\Big[\boldsymbol{P}_{b,uv}^{\frac{1}{2}}(\boldsymbol{A}_{u\to v}\hat{\boldsymbol{x}}_u^{Km} + \boldsymbol{A}_{v\to u}\hat{\boldsymbol{x}}_v^{Km} - \boldsymbol{c}_{uv}) - \boldsymbol{P}_{b,uv}^{-\frac{1}{2}}(\hat{\boldsymbol{\lambda}}_{u|v}^{Km} - \hat{\boldsymbol{\lambda}}_{v|u}^{Km})\Big] = \boldsymbol{0} \quad (20)$$

$$\lim_{K\to\infty}\Big[\boldsymbol{P}_{b,uv}^{\frac{1}{2}}(\boldsymbol{A}_{u\to v}\hat{\boldsymbol{x}}_u^{Km}+\boldsymbol{A}_{v\to u}\hat{\boldsymbol{x}}_v^{(K-1)m}-\boldsymbol{c}_{uv}) + \boldsymbol{P}_{b,uv}^{-\frac{1}{2}}(\hat{\boldsymbol{\lambda}}_{u|v}^{Km}-\hat{\boldsymbol{\lambda}}_{v|u}^{(K-1)m})\Big]=\boldsymbol{0} \quad (21)$$

$$\lim_{K\to\infty}\Delta\boldsymbol{P}_{p,uv}^{\frac{1}{2}}\Big[\boldsymbol{A}_{u\to v}\hat{\boldsymbol{x}}_u^{Km}+\boldsymbol{A}_{v\to u}\hat{\boldsymbol{x}}_v^{(K-1)m}-\boldsymbol{c}_{uv}\Big]=\boldsymbol{0} \quad (22)$$

$$\lim_{K\to\infty}\Delta\boldsymbol{P}_{p,uv}^{\frac{1}{2}}\Big[\boldsymbol{A}_{u\to v}\hat{\boldsymbol{x}}_u^{Km}+\boldsymbol{A}_{v\to u}\hat{\boldsymbol{x}}_v^{Km}-\boldsymbol{c}_{uv}\Big]=\boldsymbol{0} \quad (23)$$

$$\lim_{K\to\infty}\Delta\boldsymbol{P}_{d,uv}^{\frac{1}{2}}\Big[\hat{\boldsymbol{\lambda}}_{u|v}^{Km}-\hat{\boldsymbol{\lambda}}_{v|u}^{(K-1)m}\Big]=\boldsymbol{0} \quad (24)$$

$$\lim_{K\to\infty}\Delta\boldsymbol{P}_{d,uv}^{\frac{1}{2}}\Big[\hat{\boldsymbol{\lambda}}_{u|v}^{Km}-\hat{\boldsymbol{\lambda}}_{v|u}^{Km}\Big]=\boldsymbol{0}. \quad (25)$$

5. APPLICATION TO DISTRIBUTED AVERAGING

In this section, we consider applying BiADMM to distributed averaging. Distributed averaging is one of the basic operations for advanced distributed signal processing. For instance, in [10], distributed averaging has been applied successfully for distributed speech enhancement. Since the pioneering work by Boyd et al. [3], many algorithms have been proposed for distributed averaging. See [11] for a thorough overview of the developed algorithms.

5.1. Problem formulation and algorithm setup

Suppose every node i in a graph $G=(\mathcal{V},\mathcal{E})$ carries a scalar value, denoted as t_i. The problem is to compute the average value $t_{ave}=\frac{1}{m}\sum_{i\in\mathcal{V}}t_i$ iteratively only through message-passing between neighboring nodes in the graph. The above averaging problem can be formulated as a quadratic optimization over the graph as

$$\min_{\{x_i\}}\sum_{i\in\mathcal{V}}\frac{1}{2}(x_i-t_i)^2 \quad \text{s.t. } x_i-x_j=0 \quad \forall(i,j)\in\mathcal{E}. \quad (26)$$

One can easily show that the optimal solution is $x_1^*=\ldots=x_m^*=t_{ave}$. The above problem falls within the general formulation (1).

Before applying BiADMM to solve (26), we first configure the the set $\mathcal{P}$ in $L_{\mathcal{P}}$. For distributed averaging, all the matrices in $\mathcal{P}$ become scalars. For the purpose of simplicity, we let $\boldsymbol{P}_{p,ij}=\boldsymbol{P}_{d,ij}=1$ for every edge $(i,j)\in\mathcal{E}$.

5.2. Experimental results

In the experiment, the tested graph was a 10×10 two-dimensional grid (corresponding to $m=|\mathcal{V}|=100$), implying that each node may have two, three or four neighbors. The mean squared error (MSE) $\frac{1}{m}\|\hat{\boldsymbol{x}}-t_{ave}\mathbf{1}\|_2^2$ was taken as the performance measurement. For comparison, besides BiADMM, we also implemented the broadcast-based algorithm in [12] (referred to as *broadcast*), the randomized gossip algorithm in [3] (referred to as *gossip*) and ADMM. Both *broadcast* and *gossip* algorithms only work under the asynchronous updating scheme. While *broadcast* algorithm randomly activates one node per iteration, *gossip* algorithm randomly activates one edge per iteration for parameter-updating.

In order to implement ADMM, we first reformulate (26) into (2). We then apply ADMM to optimize an augmented Lagrangian function constructed from (2), where at each iteration the estimates for $\boldsymbol{x}$, $\boldsymbol{z}$ and the Lagrangian multiplier were updated via a Gauss-Seidel procedure (see [13]). We refer to the above implementation as the *synchronous ADMM*. Recently in [14], the authors proposed to update only a few components of $\boldsymbol{x}$, $\boldsymbol{z}$ and the Lagrangian multiplier per iteration. Specifically, in [14], only one edge in the graph is activated per iteration and the estimates of its associated variables are updated accordingly. We refer to the alternative implementation in [14] as the *asynchronous ADMM*. The asynchronous ADMM is similar to the *gossip* algorithm in the sense that both algorithms activates one edge per iteration.

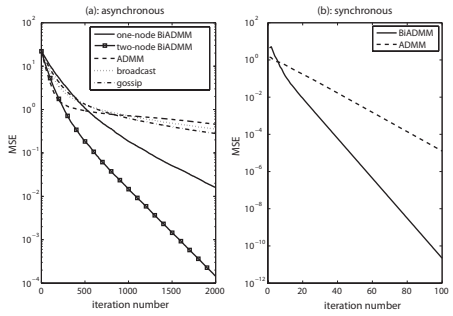

Fig. 2. Experimental comparison.

We note that the asynchronous ADMM essentially activates two neighboring nodes (or equivalently one edge) per iteration. To make a fair comparison between BiADMM and ADMM, we implemented two versions of BiADMM for the asynchronous scheme. The first version follows Subsection 3.2 where at each iteration only one node is activated, referred to as *one-node BiADMM*. The second version of BiADMM activates two neighboring nodes per iteration, referred to as *two-node BiADMM*.

In the experiment, the *gossip* and *broadcast* algorithms were initialized according to [3] and [12], respectively. The initial estimates of $\{x_i\}$ for BiADMM and ADMM were set to $\{\hat{x}_i^0=t_i\}$ whiles all the others were set to zeros. Finally, in ADMM, the scalar parameter in the augmented Lagrangian function was set to 1 to be consistent with the configuration of BiADMM.

The performance of the algorithms is displayed in Fig. 2, where subplot (a) and (b) are for the asynchronous and synchronous schemes, respectively. We first focus on subplot (a) for the asynchronous scheme. Each curve in the subplot was obtained by averaging over 100 simulations to mitigate the effect of randomness introduced in node or edge-activation. It is seen that the *two-node BiADMM* converges the fastest among all the algorithms. After 700 iterations, the *one-node BiADMM* outperforms ADMM, *gossip* and *broadcast* algorithms. Conversely, ADMM converges relatively fast at the beginning. As the iteration increases, the algorithm gradually converges slowly as compared to other algorithms. The above results suggest that BiADMM leads to fast information-spread over the graph than the other three algorithms.

Fig. 2 (b) demonstrates the performance of BiADMM and ADMM for the synchronous scheme. Both algorithms appear to have linear convergence rates. This may because the objective functions in (26) are strongly convex and have gradients which are Lipschitz continuous. Again BiADMM converges significantly faster than ADMM, which might be due to the fact that BiADMM avoids the auxiliary variable $\boldsymbol{z}$ used in ADMM.

6. CONCLUSION

In this paper, we have extended BiADMM for optimization over a graph of multiple nodes. The augmented primal-dual Lagrangian function is carefully designed to be in line with the graph topology. Theoretically, we have shown that under both synchronous and asynchronous updating schemes, BiADMM possesses a convergence rate of $\mathcal{O}(1/K)$ for general closed, proper and convex functions defined over the graph. As an example, we have applied BiADMM for distributed averaging, which shows that BiADMM significantly outperforms the state-of-art methods.

7. REFERENCES

[1] T. Richardson and R. Urbanke, *Modern Coding Theory*, Cambridge University Press, 2008.

[2] G. Zhang, R. Heusdens, and W. Bastiaan Kleijn, "Large Scale LP Decoding with Low Complexity," *IEEE Communications Letters*, vol. 17, no. 11, pp. 2152–2155, 2013.

[3] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized Gossip Algorithms," *IEEE Trans. Information Theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[4] D. Sontag, A. Globerson, and T. Jaakkola, "Introduction to Dual Decomposition for Inference," in *Optimization for Machine Learning*. 2011, MIT Press.

[5] J. Duchi, A. Agarwal, and M. J. Wainwright, "Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling," in *IEEE Trans. Automatic Control*, 2012, vol. 57, pp. 592–606.

[6] A. Nedić and A. Ozdaglar, "Distributed Subgradient Methods for Multi-agent Optimization," *IEEE Transactions on Automatic Control*, 2008.

[7] J. Chen and A.H. Sayed, "Diffusion Adaptation Strategies for Distributed Optimization and Learning Over Networks," *IEEE Trans. Signal Processing*, vol. 60, no. 8, pp. 4289–4305, 2012.

[8] G. Zhang, R. Heusdens, and W. Bastiaan Kleijn, "On the Convergence Rate of the Bi-Alternating Direction Method of Multipliers," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2014, pp. 3897–3901.

[9] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[10] Y. Zeng and R. C. Hendriks, "Distributed Delay and Sum Beamformer for Speech Enhancement via Randomized Gossip," *IEEE/ACM Trans. Audio, Speech and Language Processing*, vol. 22, no. 1, pp. 260–273, 2014.

[11] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione, "Gossip Algorithms for Distributed Signal Processing," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1847–1864, 2010.

[12] F. Iutzeler, P. Ciblat, and W. Hachem, "Analysis of Sum-Weight-Like Algorithms for Averaging in Wireless Sensor Networks," *IEEE Trans. Signal Processing*, vol. 61, no. 11, pp. 2802–2814, 2013.

[13] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *In Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[14] E. Wei and A. Ozdaglar, "On the O(1/k) convergence of asynchronous distributed alternating direction method of multipliers," arXiv:1307.8254v1 [math.OC], 2013.