Distributed Basis Pursuit

João F. C. Mota, João M. F. Xavier, Pedro M. Q. Aguiar, and Markus Püschel

***Abstract*—We propose a distributed algorithm for solving the optimization problem Basis Pursuit (BP). BP finds the least ℓ_1-norm solution of the underdetermined linear system $\boldsymbol{Ax} = \boldsymbol{b}$ and is used, for example, in compressed sensing for reconstruction. Our algorithm solves BP on a distributed platform such as a sensor network, and is designed to minimize the communication between nodes. The algorithm only requires the network to be connected, has no notion of a central processing node, and no node has access to the entire matrix $\boldsymbol{A}$ at any time. We consider two scenarios in which either the columns or the rows of $\boldsymbol{A}$ are distributed among the compute nodes. Our algorithm, named D-ADMM, is a decentralized implementation of the alternating direction method of multipliers. We show through numerical simulation that our algorithm requires considerably less communications between the nodes than the state-of-the-art algorithms.**

***Index Terms*—Basis pursuit, distributed optimization, sensor networks, augmented Lagrangian**

I. Introduction

Basis Pursuit (BP) is the convex optimization problem [1]

$$\begin{array}{ll} \text{minimize} & \|x\|_1 \\ \text{subject to} & Ax = b, \end{array} \tag{BP}$$

where the optimization variable is $x \in \mathbb{R}^n$, $\|x\|_1 = |x_1| + \cdots + |x_n|$ is the ℓ_1 norm of the vector x, and $A \in \mathbb{R}^{m \times n}$ is a matrix with more columns than rows: $m < n$. In words, BP seeks the "smallest" (in the ℓ_1 norm sense) solution of the underdetermined linear system $Ax = b$. To make sure that $Ax = b$ has at least one solution, we require the following.

Assumption 1. *A is full rank.*

BP has recently attracted attention due to its ability to find the sparsest solution of a linear system under certain conditions (see [2], [3]). In particular, BP is a convex relaxation of the combinatorial and nonconvex problem obtained by replacing the ℓ_1 norm in (BP) by the ℓ_0 pseudonorm $\|x\|_0$, which counts the number of nonzero elements of x. Note that the linear system $Ax = b$ has a unique k-sparse solution, i.e., a solution whose ℓ_0 norm is k, if every set of $2k$ columns of A is linearly independent.

João M. F. Xavier, Pedro M. Q. Aguiar, and João F. C. Mota are with Instituto de Sistemas e Robótica (ISR), Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal.

João F. C. Mota is also with the Department of Electrical and Computer Engineering at Carnegie Mellon University, USA.

Markus Püschel is with the Department of Computer Science at ETH Zurich, Switzerland.

This work was supported by the FCT grant CMU-PT/SIA/0026/2009, PTDC/EEA-ACR/73749/2006 and SFRH/BD/33520/2008 (through the Carnegie Mellon/Portugal Program managed by ICTI) from Fundação para a Ciência e Tecnologia and also by ISR/IST plurianual funding (POSC program, FEDER). This work was also supported by NSF through award 0634967.

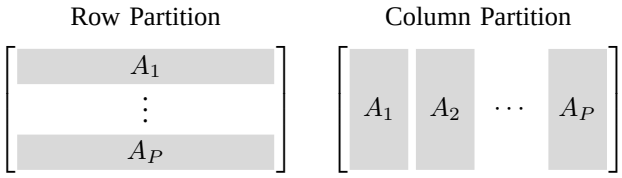


Figure 1. Row partition and column partition of A into P blocks. We assume there are P nodes and the pth node stores A_p. In the row partition a block is a set of rows, while in the column partition a block is a set of columns.

BP belongs to a set of optimization problems that has applications in many areas of engineering. Examples include signal and image denoising and restoration [1], [2], compression, fitting and approximation of functions [4], channel estimation and coding [3] and compressed sensing [5], [6] (for more applications see for example [7], [2] and the references therein). In particular, in the recent field of compressed sensing, BP plays a key role in the reconstruction of a signal.

Notice that Assumption 1 holds with probability one if the entries of A are independent and identically distributed (i.i.d.) and drawn from some (non-degenerate) probability distribution, as commonly seen in compressed sensing [5]. Also in compressed sensing, there are several strategies to deal with noisy observations, i.e., when the observation vector b is corrupted with noise. These include solving variations of (BP), namely BPDN [1] and LASSO [8].

Problem statement and contribution. Consider a network (e.g., a sensor network) with P compute nodes, and partition the matrix A into P blocks. Our goal is to solve BP in a distributed way. By distributed we mean that there is no notion of a central processing node and that the pth node has only access to the block A_p of A during the execution.

We partition A into blocks in two different ways, which we call *row partition* and *column partition*, visualized in Figure 1. In the row partition, the block A_p contains m_p rows of A, which implies $m_1 + \cdots + m_P = m$. In the column partition, A_p contains n_p columns of A, which implies $n_1 + \cdots + n_P = n$.

In summary: *given a network, we solve BP in a distributed way, either in the row partition or in the column partition.*

For the network we only require:

Assumption 2. *The given network is connected and static.*

Connected means that for any two nodes there is a path connecting them. Static means that the network topology does not change over time.

We propose an algorithm to solve this problem and show through extensive simulations that it improves over previous work (discussed below), by reducing the total number of communications to achieve a given solution accuracy. The number of communications in distributed algorithms is an

important measure of performance. For example, it is well known that communicating with the neighboring nodes is the most energy-consuming task for a wireless sensor [9]; as a consequence, many energy-aware algorithms and protocols for wireless sensor networks have been proposed [10]. An energy-aware algorithm minimizes the communications and/or allows the nodes to become idle for some time instants. On distributed supercomputing platforms, on the other hand, computation time is the main concern. In this case, the computational bottleneck is again the communication between the nodes, and thus algorithms requiring less communications have the potential of being faster.

Before we discuss related work, we provide possible applications of our algorithm.

Application: row partition. Given a network of P interconnected sensors, we try to capture an ultra-wide band but spectrally sparse signal, represented in vector form as $x \in \mathbb{R}^n$. For simplicity, we assume the pth sensor only stores one row $r_p^\top$ of A, i.e., $m = P$. Each sensor only captures some time samples at a rate far below the Nyquist rate, using for example a random demodulator [11], [8]. One can represent each measurement as the number b_p. Under certain conditions ([5], [6], [12]), it is possible to recover x by solving (BP) with $A = [r_1 \cdots r_P]^\top$ and $b = [b_1 \cdots b_P]^\top$. Further details about the matrix A and the vector b can be found in [8]. Since each vector r_p is associated with a sensor, this corresponds to our row partition case. This scenario applies, for example, to sparse event detection in wireless networks [13], and to distributed target localization in sensor networks [14].

Application: column partition. The work [15] introduces a method of speeding up seismic forward modeling in geological applications. The goal is to find the Green's functions of some model of a portion of the earth's surface. Given a set of sources and a set of receivers, from the knowledge of both the emitted and the received signals, the Green's function of the model, represented by x, has to be found. The authors of [15] propose to solve this problem when all sources emit at the same time and the receivers capture a linear superposition of all signals. The approach is then to solve BP, where a set of columns of A is associated with a source. Note that a distributed solution makes sense because the sources are physically far apart.

As another example for the column partition, we interpret BP as finding a sparse representation of a given signal b with respect to a dictionary of atomic signals (columns of A). It is common to assume that the dictionary (the matrix A) contains several families of functions, e.g., Fourier, DCT, wavelets, to become overcomplete. Suppose that we are given P processors, each of which is tuned to perform computations for a certain family of functions. In this case, solving BP in a column partition framework would arise naturally.

Algorithms for solving BP and related work. Since BP can be recast as a linear program (LP) [4], any algorithm that solves LPs can also solve BP. Among the many algorithms solving LPs [16], most cannot be readily adapted to our distributed scenario. For example, the (distributed) simplex algorithm [17], [18] can solve LPs only in complete networks, i.e., those with a link between any pair of nodes. In this paper, we aim to solve BP for every connected network topology.

In recent years, some approaches have been proposed for solving general optimization problems, including BP, in distributed networks. For example, [19] proposes a method based on subgradient algorithms, but these are known to converge very slowly. Other approaches to distributed optimization combine the method of multipliers (MM) with the nonlinear Gauss-Seidel (NGS) method or with Jacobi algorithms [20]. For example, [21] uses MM together with a Jacobi-type algorithm named diagonal quadratic approximation (DQA) to solve, in a distributed way, convex problems constrained by linear equations. Using a suitable reformulation of (BP), this method can be applied to our problem statement. In [22] we analyzed how well MM together with NGS solves BP in the row partition scenario; and in [23] we used a fast gradient algorithm in both loops. The algorithm we propose here has just one loop and requires considerably fewer iterations to converge than all the previous approaches.

Fast algorithms solving BP in a non-distributed way include spgl1 [24], fpc [25], LARS [26], C-SALSA [27], and NESTA [28]. These are faster than distributed algorithms but require that A and b are available at the same location. In contrast, a distributed algorithm can solve problems that can only fit into the combined memory of all the nodes.

The work [29] is closest related to ours. It solves the Basis Pursuit Denoising (BPDN) [1] (a noise-robust version of BP), which also produces sparse solutions of linear systems. The algorithm is called D-Lasso and can be adapted to solve our problem. Our simulations show that the algorithm we propose requires systematically less communications than D-Lasso.

Our algorithm is based on the alternating direction method of multipliers (ADMM). The work [30] also uses ADMM in a distributed scenario, but is only applicable to networks where all the nodes connect to a central node. Our algorithm, in contrast, is designed for decentralized scenarios (no central node) and applies to any connected network.

Our type of matrix partitioning has been considered before in the context of distributed algorithms for linear programs [17], [18] and in regression of distributed data [31].

II. Row Partition

In this section we partition the matrix A by rows:

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_P \end{bmatrix},$$

where each block $A_p \in \mathbb{R}^{m_p \times n}$ contains a subset of rows of A such that $m_1 + \cdots + m_P = m$. The vector b is partitioned similarly: $b = [b_1^\top \cdots b_P^\top]^\top$. We assume that A_p and b_p are available only at the pth node of a connected network with P compute nodes. We model the network as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, 2, \ldots, P\}$ is the set of nodes and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the set of edges. We represent the edge connecting nodes i and j by $\{i, j\}$ or $\{j, i\}$; E is the total number of edges. See Figure 2 for an example graph. If $\{i, j\}$ is an edge, then node i and node j can exchange

messages with each other. The set of neighbors of node p is written as $\mathcal{N}_p$, and its degree is $D_p = |\mathcal{N}_p|$.

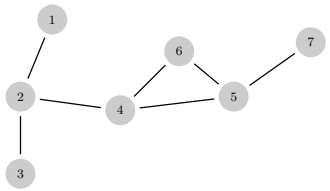


Figure 2. Example of a connected network with $P = E = 7$. The set of edges is $\mathcal{E} = \{\{1,2\},\{2,3\},\{2,4\},\{4,5\},\{4,6\},\{5,6\},\{5,7\}\}$.

Graph coloring. We assume that a proper coloring $\mathcal{C} = \{1,\dots,C\}$ of the graph is available. This means that each node is labeled with a number $c \in \mathcal{C}$, which we call color, such that no adjacent nodes (i.e., neighbors) have the same color. The minimum number of colors required for a proper coloring of a graph $\mathcal{G}$ is its chromatic number $\chi(\mathcal{G})$. Coloring a graph with $\chi(\mathcal{G})$ colors or just computing $\chi(\mathcal{G})$ is NP-hard for $\chi(\mathcal{G}) > 2$ [32]. Several distributed algorithms for coloring a graph exist [33], [34], [35], [36]. For example, [33] determines a coloring with $O(D_{\max})$ colors, where $D_{\max} = \max_p D_p$, using $O(D_{\max}/\log^2(D_{\max}) + \log^\star(P))$ iterations. If more colors are allowed, for example $O(D_{\max}^2)$, then $O(\log^\star(P))$ iterations suffice [36]. In this paper we assume that a proper coloring $\mathcal{C}$ with C colors is given.

Problem reformulation. To solve BP in a distributed way we first rewrite (BP) to make the row partition explicit:

$$\begin{array}{ll}\text{minimize} & \|x\|_1 \\ \text{subject to} & A_p x = b_p\,, \quad p = 1,\dots,P\,.\end{array} \tag{1}$$

The variable x is coupling the problem. To decouple, we replace x with P copies of x. The pth copy is denoted with x_p. To ensure the necessary global consistency condition $x_1 = x_2 = \cdots = x_P$, we enforce the equivalent (since the network is connected) constraint $x_i = x_j$ for each edge $\{i,j\}$ of the network:

$$\begin{array}{ll}\text{minimize} & \frac{1}{P}\sum_{p=1}^P \|x_p\|_1 \\ \text{subject to} & A_p x_p = b_p\,, \quad p = 1,\dots,P \\ & x_i = x_j\,, \quad \{i,j\} \in \mathcal{E}\,.\end{array} \tag{2}$$

The optimization variable is $\bar{x} := (x_1,\dots,x_P) \in (\mathbb{R}^n)^P$. Note that (2) can be written more compactly as

$$\begin{array}{ll}\text{minimize} & \frac{1}{P}\sum_{p=1}^P \|x_p\|_1 \\ \text{subject to} & A_p x_p = b_p\,, \quad p = 1,\dots,P \\ & (B^\top \otimes I_n)\bar{x} = 0\,,\end{array} \tag{3}$$

where I_n is the $n \times n$ identity matrix, and $\otimes$ is the Kronecker product. The matrix B is the $P \times E$ node-arc incidence matrix of the graph: each edge $\{i,j\} \in \mathcal{E}$ corresponds to a column in B with the ith and jth entries equal to 1 and -1, respectively.

Algorithm for bipartite graphs. We first consider a simple case: $\mathcal{G}$ is bipartite and hence $\chi(\mathcal{G}) = 2$. The generalization to any connected graph will be straightforward. Bipartite graphs include trees and grid graphs.

Without loss of generality, assume nodes 1 to c have color 1 and the remaining have color 2. Then, (3) can be written as

$$\begin{array}{ll}\text{minimize} & \frac{1}{P}\sum_{p=1}^c \|x_p\|_1 + \frac{1}{P}\sum_{p=c+1}^P \|x_p\|_1 \\ \text{subject to} & A_p x_p = b_p\,, \quad p = 1,\dots,P \\ & (B_1^\top \otimes I_n)\bar{x}_1 + (B_2^\top \otimes I_n)\bar{x}_2 = 0\,,\end{array} \tag{4}$$

where $\bar{x} = (\bar{x}_1, \bar{x}_2) \in (\mathbb{R}^n)^c \times (\mathbb{R}^n)^{P-c}$ and B is partitioned as $B = \begin{bmatrix} B_1^\top & B_2^\top \end{bmatrix}^\top$. We propose the alternating direction method of multipliers (ADMM, briefly described in appendix A) to solve (4). The augmented Lagrangian of (4), dualizing only the last constraint, is

$$\begin{aligned} L(\bar{x}_1,\bar{x}_2;\lambda) = \frac{1}{P}\sum_{p\in\mathcal{C}_1}\|x_p\|_1 + \frac{1}{P}\sum_{p\in\mathcal{C}_2}\|x_p\|_1 + \phi_1(\bar{x}_1,\lambda) \\ + \phi_2(\bar{x}_2,\lambda) + \rho\bar{x}_1^\top(B_1 B_2^\top \otimes I_n)\bar{x}_2\,, \end{aligned} \tag{5}$$

where $\mathcal{C}_1 = \{1,\dots,c\}$, $\mathcal{C}_2 = \{c+1,\dots,P\}$, and

$$\begin{aligned} \phi_i(\bar{x}_i,\lambda) &= \lambda^\top(B_i^\top \otimes I_n)\bar{x}_i + \frac{\rho}{2}\|(B_i^\top \otimes I_n)\bar{x}_i\|^2 \\ &= ((B_i \otimes I_n)\lambda)^\top\bar{x}_i + \frac{\rho}{2}\bar{x}_i^\top(B_i B_i^\top \otimes I_n)\bar{x}_i\,, \end{aligned}$$

for $i = 1, 2$. Note that, since nodes in each $\mathcal{C}_i$ are not neighbors between themselves, $B_i B_i^\top$ is diagonal (with D_p in the pth diagonal entry). Hence,

$$\phi_i(\bar{x}_i,\lambda) = \sum_{p\in\mathcal{C}_i}\left(\gamma_p^\top x_p + \frac{\rho}{2}D_p\|x_p\|^2\right), \quad i = 1,2\,, \tag{6}$$

where $\gamma_p := \sum_{j\in\mathcal{N}_p}\text{sign}(j-p)\lambda_{\{p,j\}}$ and $\text{sign}(w)$ gives 1 if $w \geq 0$ and -1 otherwise. We decomposed the dual variable λ into $(\dots,\lambda_{\{i,j\}},\dots)$, where $\lambda_{\{i,j\}} = \lambda_{\{j,i\}}$ is associated with the constraint $x_i = x_j$.

Equations (5) and (6) show that minimizing $L(\bar{x}_1,\bar{x}_2;\lambda)$ with respect to (w.r.t.) $\bar{x}_1$ yields c optimization problems that can be executed in parallel; similarly, minimizing it w.r.t. $\bar{x}_2$ yields $P - c$ parallel optimization problems. Algorithm 1 shows the application of ADMM to our problem. We name our algorithm D-ADMM, after Distributed ADMM.

Algorithm 1 D-ADMM for bipartite graphs

Initialization: for all $p \in \mathcal{V}$, set $\gamma_p^{(1)} = x_p^{(1)} = 0$ and $k = 1$

1: **repeat**
2: **for all** $p \in \mathcal{C}_1$ [in parallel] **do**
3: Set $v_p^{(k)} = \gamma_p^{(k)} - \rho\sum_{j\in\mathcal{N}_p} x_j^{(k)}$ and find

$$x_p^{(k+1)} = \begin{array}{ll}\underset{x_p}{\text{argmin}} & \frac{1}{P}\|x_p\|_1 + {v_p^{(k)}}^\top x_p + \frac{D_p\rho}{2}\|x_p\|^2 \\ \text{s.t.} & A_p x_p = b_p\end{array}$$

4: Send $x_p^{(k+1)}$ to $\mathcal{N}_p$
5: **end for**
6: Repeat 2-5 for all $p \in \mathcal{C}_2$, replacing $x_j^{(k)}$ by $x_j^{(k+1)}$
7: **for all** $p \in \mathcal{C}_1 \cup \mathcal{C}_2$ [in parallel] **do**

$$\gamma_p^{(k+1)} = \gamma_p^{(k)} + \rho\sum_{j\in\mathcal{N}_p}(x_p^{(k+1)} - x_j^{(k+1)})$$

8: **end for**
9: $k \leftarrow k + 1$
10: **until** some stopping criterion is met

The optimization problem in step 3 results from minimizing the augmented Lagrangian $L(\bar{x}_1,\bar{x}_2;\lambda)$ w.r.t. x_p. To derive it,

note that (6) enables us to rewrite $L(\bar{x}_1, \bar{x}_2; \lambda)$ as

$$L(\bar{x}_1, \bar{x}_2; \lambda) = \sum_{i=1}^{2} \sum_{p \in \mathcal{C}_i} \Big(\frac{1}{P} \|x_p\|_1 + \gamma_p^\top x_p + \frac{\rho}{2} D_p \|x_p\|^2 \Big) + \rho\, \bar{x}_1 (B_1 B_2^\top \otimes I_n) \bar{x}_2 \,.$$

The (ij)th entry of $B_1 B_2^\top$ is -1 if $\{i,j\} \in \mathcal{E}$ and 0 otherwise. Therefore, $\rho\, \bar{x}_1 (B_1 B_2^\top \otimes I_n) \bar{x}_2 = -\rho \sum_{\{i,j\} \in \mathcal{E}} x_i^\top x_j$. Picking $p \in \mathcal{C}_i$ for any $i = 1, 2$ and minimizing $L(\bar{x}_1, \bar{x}_2; \lambda)$ w.r.t. x_p yields the optimization problem in step 3. Appendix B describes an efficient method for solving this problem.

Algorithm 1 shows that nodes with the same color operate in parallel, whereas nodes with different colors cannot. In other words, the nodes from $\mathcal{C}_1$ have to wait for the computation of the nodes from $\mathcal{C}_2$ and vice-versa. However, at the end of each iteration, every node will have communicated once (sending $x_p^{(k+1)}$ and receiving $x_j^{(k+1)}$) with all its neighbors.

Regarding the dual variable λ, its components do not appear explicitly in Algorithm 1. The reason is that node p only requires $\gamma_p = \sum_{j \in \mathcal{N}_p} \text{sign}(j-p) \lambda_{\{p,j\}}$ for its optimization problem. According to the canonical form of ADMM, we have to update $\lambda_{\{i,j\}}$, for each edge $\{i,j\} \in \mathcal{E}$ as

$$\lambda_{\{i,j\}}^{(k+1)} = \lambda_{\{i,j\}}^{(k)} + \rho\, \text{sign}(j-p)(x_i^{(k+1)} - x_j^{(k+1)}) \,. \qquad (7)$$

Inserting (7) into the expression of γ_p we obtain the update of step 7.

The following theorem establishes the convergence of Algorithm 1.

Theorem 1. *Assume the given graph is bipartite. Then, for all p, the sequence $\{x_p^{(k)}\}$ produced by Algorithm 1 converges to a solution of* (BP)*.*

Proof: We have already seen that when the graph is bipartite (BP) is equivalent to (4). We now show that (4) satisfies the conditions of Theorem 4 in appendix A. Let $f_i(\bar{x}_i) = (1/P) \sum_{c \in \mathcal{C}_i} \|x_p\|_1$, for $i = 1, 2$. Clearly, f_1 and f_2 are real-valued convex functions. Assumption 1 on the rank of the matrix A implies that (BP), and thus (4), is always solvable. Also, the non-dualized equations $A_p x_p = b_p$ in (4) define polyhedral sets.

Now we have to prove that the matrices $B_1^\top \otimes I_n$ and $B_2^\top \otimes I_n$ have full column rank, i.e., that $B_1^\top$ and $B_2^\top$ have full column rank. We have seen that $B_1 B_1^\top$ and $B_2 B_2^\top$ are diagonal matrices because the nodes within one class are not neighbors. Note that the pth entry of the diagonal of $B_1 B_1^\top$ (or $B_2 B_2^\top$) is the degree of the pth node. Due to Assumption 2, there are no isolated nodes and thus $B_1 B_1^\top$ and $B_2 B_2^\top$ are full-rank. The result then follows because $\text{rank}\,(BB^\top) = \text{rank}\,(B^\top)$ for any matrix B. ∎

Theorem 1 also shows that after Algorithm 1 terminates, every node will know a solution $x^\star$ of BP.

Algorithm for general graphs. We now generalize Algorithm 1 to arbitrary graphs with $\chi(\mathcal{G}) > 2$. The generalization is straightforward, but we cannot guarantee convergence as in Theorem 1. However, in our extensive experiments, shown later, the resulting algorithm never failed to converge.

Let $\mathcal{G}$ be a graph with a proper coloring $\mathcal{C}$ and let $C = |\mathcal{C}|$ be the number of colors. Let $\mathcal{C}_c$ be the set of nodes that have color c, $c = 1, \ldots, C$. Without loss of generality, suppose the nodes are numbered the following way: $\mathcal{C}_1 = \{1, \ldots, |\mathcal{C}_1|\}$, $\mathcal{C}_2 = \{|\mathcal{C}_1|+1, \ldots, |\mathcal{C}_1|+|\mathcal{C}_2|\}$, …, $\mathcal{C}_C = \{\sum_{c=1}^{C-1} |\mathcal{C}_c| + 1, \ldots, P\}$. This enables a partition of the matrix B as $B = \begin{bmatrix} B_1^\top & \cdots & B_C^\top \end{bmatrix}^\top$, making (3) equivalent to

$$\begin{array}{ll} \text{minimize} & \frac{1}{P} \sum_{c=1}^{C} \sum_{p \in \mathcal{C}_c} \|x_p\|_1 \\ \text{subject to} & A_p x_p = b_p \,, \quad p = 1, \ldots, P \\ & \sum_{c=1}^{C} (B_c^\top \otimes I_n) \bar{x}_c = 0 \,, \end{array} \qquad (8)$$

where $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_C)$ is the variable, and $\bar{x}_c \in (\mathbb{R}^n)^{|\mathcal{C}_c|}$ for $c = 1, \ldots, C$. From the proof of Theorem 1 we know that each matrix B_c has full row rank. Thus, we can apply the generalized ADMM to solve (8) (see Appendix A). That leads to the following algorithm.

Algorithm 2 D-ADMM for general graphs

Initialization: for all $p \in \mathcal{V}$, set $\gamma_p^{(1)} = x_p^{(1)} = 0$ and $k = 1$

1: **repeat**
2: **for** $c = 1, \ldots, C$ **do**
3: **for all** $p \in \mathcal{C}_c$ [in parallel] **do**

$$v_p^{(k)} = \gamma_p^{(k)} - \rho \sum_{\substack{j \in \mathcal{N}_p \\ j<p}} x_j^{(k+1)} - \rho \sum_{\substack{j \in \mathcal{N}_p \\ j>p}} x_j^{(k)}$$

4: and find

$$x_p^{(k+1)} = \begin{array}{ll} \underset{x_p}{\text{argmin}} & \frac{1}{P}\|x_p\|_1 + {v_p^{(k)}}^\top x_p + \frac{D_p \rho}{2}\|x_p\|^2 \\ \text{s.t.} & A_p x_p = b_p \end{array}$$

5: Send $x_p^{(k+1)}$ to $\mathcal{N}_p$
6: **end for**
7: **end for**
8: **for all** $p = 1, \ldots, P$ [in parallel] **do**

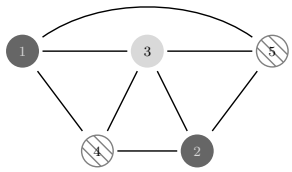
$$\gamma_p^{(k+1)} = \gamma_p^{(k)} + \rho \textstyle\sum_{j \in \mathcal{N}_p} (x_p^{(k+1)} - x_j^{(k+1)})$$

9: **end for**
10: $k \leftarrow k + 1$
11: **until** some stopping criterion is met

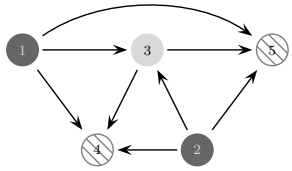
Algorithm 2 is a straightforward generalization of Algorithm 1. Now there are C classes of nodes and all the nodes in one class "work" in parallel, but the classes cannot work at the same time. Consequently, if we consider the time to solve one instance of the problem in step 4 as one unit, one (outer) iteration in Algorithm 2 takes C units.

In the bipartite case the coordination between the nodes was straightforward: node p only works after it has received x_j from all its neighbors. Here, according to the canonical format of Algorithm 2, all the nodes in one class should work at the same time. Since these nodes are not neighbors, neither there is a central node to coordinate them, in practice node p works after having received $x_j^{(k+1)}$'s from all its neighbors of lower color. An alternative way to see this is to transform the undirected graph of the network into a directed graph, as shown in Figure 3. The graph in Figure 3(b) is constructed from the graph in Figure 3(a) by assigning a direction to each edge $\{i,j\}$: $i \to j$ if the color of i is smaller than the color

of j, and $i \leftarrow j$ otherwise. Then, each node only starts working after having received the x_j's from all its inward links. In practice, this procedure can reduce the overall execution time since each node does not need to wait for its "color time." As described in step 5 (and in contrast to what Figure 3(b) may suggest), each node sends x_p^{k+1} to all its neighbors in each iteration.



(a) Undirected



(b) Directed

Figure 3. (a) undirected network with $\chi(\mathcal{G}) = 3$ and with classes $\mathcal{C}_1 = \{1, 2\}$, $\mathcal{C}_2 = \{3\}$, $\mathcal{C}_3 = \{4, 5\}$; (b) directed graph constructed from (a) by assigning a direction to each link: from smallest color node to the largest color node.

As stated earlier, we have no proof of convergence for Algorithm 2, only practical evidence.

III. Column Partition

In this section, we adapt the algorithm for the row partition to the column partition case:

$$A = \begin{bmatrix} A_1 & A_2 & \cdots & A_P \end{bmatrix}.$$

Each block $A_p \in \mathbb{R}^{m \times n_p}$ contains a subset of columns of $A \in \mathbb{R}^{m \times n}$ such that $n_1 + \cdots + n_P = n$. The block A_p is only available at the pth node of an arbitrary connected network, and the vector $b \in \mathbb{R}^m$ is known by all the nodes.

Duality: pros and cons. In section II we saw an algorithm that solves BP with a row partition. Here, we want to reutilize that algorithm for BP with a column partition. The first approach to that is to consider the dual problem of (BP):

$$\begin{array}{ll} \text{minimize} & b^\top \lambda \\ \text{subject to} & -1_n \leq A^\top \lambda \leq 1_n \,, \end{array} \tag{9}$$

where the dual variable is $\lambda \in \mathbb{R}^m$, and $1_n \in \mathbb{R}^n$ is the vector of all ones. For a derivation of (9), see for example [37, §1.3.3]. The matrix A now appears in the constraints of (9) as $A^\top$, i.e., we can partition the constraint matrix in (9) by rows. The problem is that there is no straightforward way to recover a solution of (BP) from a solution of (9). Hence we need an alternative approach.

Regularizing BP. Consider the following regularized version of (BP):

$$\begin{array}{ll} \text{minimize} & \|x\|_1 + \frac{\delta}{2}\|x\|^2 \\ \text{subject to} & Ax = b \,, \end{array} \tag{10}$$

where δ is a small positive number. While (BP) may have multiple solutions, (10) just has one, due to the strict convexity of its objective. When δ is small enough, (10) selects the least ℓ_2-norm solution of (BP):

Theorem 2. *There exists $\bar{\delta} > 0$ such that the solution of* (10) *is a solution of* (BP) *for all $0 < \delta < \bar{\delta}$.*

The proof of this theorem is based on exact regularization results for linear programming [38], [39]. To prove it, recast (BP) as a linear program [1], regularize it, and then rewrite the resulting problem as (10). Consequently, we recover a solution of (BP) if (10) is solved for a sufficiently small δ. The benefit of solving (10) is that it is immediate to recover the solution of (10) from its dual solution. We are unaware of any strategy for choosing δ without first solving (BP). We will thus adopt a trial-and-error strategy.

Dual problem. We use duality because the dual problem of (10) will have terms involving $A^\top$. Since A is partitioned by columns, $A^\top$ will be partitioned by rows. Therefore, the algorithm for the row partition will be applicable with some minor modifications.

The dual problem of (10) is

$$\begin{array}{ll} \text{maximize} \\ y \end{array} \quad L(y) \,, \tag{11}$$

where the dual function is $L(y) = -b^\top y + \inf_x(\|x\|_1 + (A^\top y)^\top x + \frac{\delta}{2}\|x\|^2)$, and $y \in \mathbb{R}^m$ the dual variable. To keep the notation consistent with the previous section, we recast (11) as a minimization problem:

$$\begin{array}{ll} \text{minimize} \\ y \end{array} \quad b^\top y + \Psi(y) \,, \tag{12}$$

where

$$\Psi(y) = -\inf_x \left(\|x\|_1 + (A^\top y)^\top x + (\delta/2)\|x\|^2 \right). \tag{13}$$

The objective of the inner optimization problem of (13) has a unique minimizer for each y, since it is strictly convex. Let $x(y)$ denote the solution of this problem, for a fixed y. Strong duality holds for (10) because its objective is convex and its constraints linear [4, §5.2.3], [40, prop.5.2.1]. Therefore, after we find a solution $y^\star$ to the dual problem (12), a (primal) solution of (10) is available as $x(y^\star)$. This follows directly from the KKT conditions [4, §5.5], [40, prop.5.1.5], and we express it in the following theorem.

Theorem 3. *Let $y^\star$ solve* (11)*. Then, $x(y^\star)$ solves* (10)*.*

Adapting the algorithm. Now we focus on solving (12). Let x be partitioned analogous to A, i.e., $x = (x_1, \ldots, x_P)$, where $x_p \in \mathbb{R}^{n_p}$. Note that $\Psi(y)$ can be decomposed as the sum of P functions: $\Psi(y) = \Psi_1(y) + \cdots + \Psi_P(y)$, where

$$\Psi_p(y) = -\inf_{x_p} \|x_p\|_1 + (A_p^\top y)^\top x_p + \frac{\delta}{2}\|x_p\|^2 \tag{14}$$

can only be computed at node p because A_p is only known there. We can then rewrite (12) as

$$\underset{y}{\text{minimize}} \quad \textstyle\sum_{p=1}^{P} \big(\frac{1}{P} b^\top y + \Psi_p(y)\big) \quad .$$

Notice that $\Psi_p(y)$ can be easily computed at node p, since the optimization problem defining it has a closed form solution. We now apply the same procedure as in section II: we clone the variable y into several y_p's, and constrain the problem with $y_i = y_j$, for all $\{i,j\} \in \mathcal{E}$. This yields

$$\begin{array}{ll} \text{minimize} & \sum_{p=1}^{P} \big(\frac{1}{P} b^\top y_p + \Psi_p(y_p)\big) \\ \text{subject to} & (B^\top \otimes I_n)\bar{y} = 0\,, \end{array} \tag{15}$$

where the variable is $\bar{y} = (y_1, \ldots, y_P) \in (\mathbb{R}^n)^P$. Note the similarity between (15) and (3). Having a proper coloring of the graph, the generalized ADMM is applicable:

Algorithm 3 D-ADMM for general graphs (column partition)

Initialization: for all $p \in \mathcal{V}$, set $\gamma_p^{(1)} = x_p^{(1)} = 0$ and $k = 1$

1: **repeat**
2: **for** $c = 1, \ldots, C$ **do**
3: **for all** $p \in \mathcal{C}_c$ [in parallel] **do**

$$v_p^{(k)} = \gamma_p^{(k)} - \rho \sum_{\substack{j \in \mathcal{N}_p \\ j<p}} x_j^{(k+1)} - \rho \sum_{\substack{j \in \mathcal{N}_p \\ j>p}} x_j^{(k)}$$

4: and find

$$y_p^{(k+1)} = \arg\min_{y_p} \Psi_p(y_p) + (v_p^{(k)} + \frac{1}{P} b)^\top y_p + \frac{D_p \rho}{2} \|y_p\|^2$$

5: Send $y_p^{(k+1)}$ to $\mathcal{N}_p$
6: **end for**
7: **end for**
8: **for all** $p = 1, \ldots, P$ [in parallel] **do**

$$\gamma_p^{(k+1)} = \gamma_p^{(k)} + \rho \textstyle\sum_{j \in \mathcal{N}_p} (y_p^{(k+1)} - y_j^{(k+1)})$$

9: **end for**
10: $k \leftarrow k + 1$
11: **until** some stopping criterion is met

Algorithm 3 is similar to Algorithm 2 except for some minor modifications: the size of the variable to be transmitted is smaller (instead of transmitting $x_p \in \mathbb{R}^n$, now the nodes transmit $y_p \in \mathbb{R}^m$), and the optimization problem to be solved at each node (see step 4) is slightly different. Since that problem is unconstrained and its objective is differentiable, we can solve it directly with the Barzilai-Borwein algorithm [41] (see appendix B for more details).

Another difference to Algorithm 2 is that after the algorithm finished (finding an optimal vector $y^\star$), node p will not know the entire solution $x(y^\star)$ to (10), but only a portion of it, $x_p(y^\star)$, as the solution to the optimization problem defining Ψ_p in (14). In case we want the entire solution $x(y^\star)$ to be available in all nodes, just a few additional communications are required because $x(y^\star)$ is expected to be sparse; furthermore, a spanning tree can be used to spread the x_p's over the network.

We remark that if the graph is bipartite, then Algorithm 3 is proven to converge to an optimal solution of (10) and, if δ is small enough, to a solution of (BP). An important issue is the possible ill-conditioning provoked by a small value of δ. In fact, a very small value for δ may lead to difficulties in finding $y_p^{(k+1)}$ in step 4. Note that this is the only step where δ appears. In our simulations, explained in section V, we used $\delta = 10^{-3}$ and this value allowed us to compute solutions to BP with a very large precision, without incurring into numerical problems.

IV. Other Algorithms

In this section we overview other methods that solve BP in a truly distributed way. We only cover the row partition case because corresponding algorithms for the column partition can always be derived as shown in the previous section.

We divide the algorithms into two categories according to the number of (nested) loops they have: single-looped and double-looped. D-ADMM is single-looped and, in each iteration, every node transmits a vector of size n to its neighbors.

Performance measure: communication steps. We say that a communication step has occurred after all the nodes finish communicating their current estimates to their neighbors. All single-looped algorithms have one communication step per iteration. The double-looped algorithms have one communication step per iteration of the inner loop. In all algorithms, the size of the transmitted vector is n. Another feature common to all algorithms is that in every iteration (or in every inner iteration, for the double-looped algorithms) each node has to solve the optimization problem in step 4 of Algorithm 2 (or Algorithm 3, for the column partition). This means that the algorithms have a common ground for comparison: if each iteration (or inner iteration, for the double-looped algorithms) involves one communication step and all the nodes have to solve a similar optimization problem (same format, same dimensions, but possibly different parameters), then the number of iterations (or the sum of inner iterations) becomes a natural metric to compare the algorithms. We will then compare the algorithms by their number of communication steps, which is equal to the number of iterations in the single-looped algorithms and to the sum of inner iterations in the double-looped algorithms. Note that less communication steps can be expected to produce significant energy savings in scenarios such as sensor networks [9].

Although data is transmitted in every communication step, the quantity of the transmitted data might actually decrease with the iterations. The reason is because the solution to BP is sparse and, at some point, the nodes' estimates start being sparse, allowing a possible compression of the transmitted data (e.g., just transmit the nonzero entries).

We start with describing the single-looped algorithms.

Subgradient. Nedić and Ozdaglar were the first to propose a subgradient-based algorithm to solve general convex optimization problems in a completely distributed way [42]. However, they only addressed unconstrained optimization problems, which is not our case. Instead, we will use the method proposed in [19], which generalizes [42] to problems

with private constraints in each node. That is, [19] solves

$$\begin{array}{ll}\text{minimize} & \sum_{p=1}^{P} f_p(x) \\ \text{subject to} & x \in \cap_{p=1}^{P} X_p\,,\end{array}$$

where each f_p is convex and each X_p is a closed convex set. This method combines consensus algorithms [43] with subgradient algorithms [40, Ch.6], and for each node p, it takes the form

$$x_p^{(k+1)} = \Big[c_{pp}^{(k)} x_p^{(k)} + \sum_{j\in\mathcal{N}_p} c_{pj}^{(k)} x_j^{(k)} - \alpha^{(k)} g_p^{(k)}\Big]^{+}_{X_p}\,, \quad (16)$$

where c_{ij} are positive weights such that $\sum_i c_{ij}^{(k)} = \sum_j c_{ij}^{(k)} = 1$, the sequence $\{\alpha^{(k)} > 0 : k = 1, 2, \ldots\}$ is square summable but not summable, and $[p]_X^+$ is the projection of the point p onto the set X: $[p]_X^+ = \arg\min_x \{\frac{1}{2}\|x-p\|^2 \,:\, x \in X\}$. The vector $g_p^{(k)}$ is a subgradient of f_p at the point $c_{pp}^{(k)} x_p^{(k)} + \sum_{j\in\mathcal{N}_p} c_{pj}^{(k)} x_j^{(k)}$.

We apply (16) directly to problem (1), where we see $\|x\|_1$ as $\|x\|_1 = \frac{1}{P}\|x\|_1 + \cdots + \frac{1}{P}\|x\|_1$; in other words, we set $f_p(x) = \frac{1}{P}\|x\|_1$. We choose $\alpha^{(k)} = 1/(k+1)$ for the step-size sequence. In our case, since the network is static (Assumption 2), the weights c_{ij} are constant: for every p, $c_{pi} = 1/(D_p+1)$ for $i \in \mathcal{N}_p \cup \{p\}$, and 0 otherwise. The implementation of (16) in a network is now straightforward: first, node p transmits $x_p^{(k)}$ to its neighbors and receives $x_j^{(k)}$ from them; then, it updates its variable with (16). These two steps are repeated until convergence.

While (16) is proven to be robust to link failures, its convergence speed is too slow in practice.

D-Lasso. As mentioned in section I, Bazerque and Giannakis [29] proposed a distributed algorithm that solves a problem similar to ours. Here, we adapt it to solve BP. The starting point is problem (2), which by introducing a new variable z_{ij} for each edge $\{i,j\} \in \mathcal{E}$, is reformulated as

$$\begin{array}{ll}\text{minimize} & \frac{1}{P}\sum_{p=1}^{P}\|x_p\|_1 \\ \text{subject to} & A_p x_p = b_p\,, \quad p = 1,\ldots,P \\ & x_i = z_{ij}\,, \quad \{i,j\}\in\mathcal{E}, \\ & x_j = z_{ij}\,, \quad \{i,j\}\in\mathcal{E}.\end{array} \quad (17)$$

This problem is solved with ADMM by dualizing its last two constraints. We consider the problem partitioned in terms of the variable $\bar{z} = (\ldots, z_{ij}, \ldots)$ and $\bar{x} = (\ldots, x_p, \ldots)$. In short, ADMM minimizes the augmented Lagrangian of (17) w.r.t. $\bar{z}$ and then minimizes it w.r.t. $\bar{x}$, using the new value of $\bar{z}$. The minimization w.r.t. $\bar{z}$ has a closed form solution. After some manipulations, the algorithm for an arbitrary node p is:

Algorithm 4 D-Lasso (node p)

Initialization: for all $p \in \mathcal{V}$, set $\gamma_p^{(1)} = x_p^{(1)} = 0$ and $k = 1$

1: **repeat**
2: **for all** $p = 1, \ldots, P$ [in parallel] **do**
3: set $v_p^{(k)} = \gamma_p^{(k)} - \rho \sum_{j\in\mathcal{N}_p\cup\{p\}} x_j^{(k)}$ and find

$$x_p^{(k+1)} = \begin{array}{ll}\underset{x_p}{\text{argmin}} & \frac{1}{P}\|x_p\|_1 + {v_p^{(k)}}^{\top} x_p + \rho D_p \|x_p\|^2 \\ \text{s.t.} & A_p x_p = b_p\end{array}$$

4: Send $x_p^{(k+1)}$ to $\mathcal{N}_p$, and receive $x_j^{(k+1)}$, $j \in \mathcal{N}_p$
5: **end for**
6: **for all** $p = 1, \ldots, P$ [in parallel] **do**

$$\gamma_p^{(k+1)} = \gamma_p^{(k)} + \rho \sum_{j\in\mathcal{N}_p} (x_p^{(k+1)} - x_j^{(k+1)})$$

7: **end for**
8: $k \leftarrow k+1$
9: **until** some stopping criterion is met

Although D-Lasso and D-ADMM (Algorithm 2) have a similar format, they are different. For example, D-Lasso is synchronous and D-ADMM asynchronous, and the parameters of the optimization problem each node solves are different in both algorithms. Also, D-ADMM is proven to converge for bipartite graphs only, while D-Lasso is proven to converge for any connected graph. In the next section, we will see that, in practice, D-ADMM converges in less iterations than D-Lasso, despite their common underlying algorithm.

We now move to the double-looped algorithms.

Double-looped algorithms. All double-looped algorithms we will see have the same theoretical foundation, but use different subalgorithms. Namely, all solve the following dual problem of (3):

$$\begin{array}{ll}\underset{\lambda}{\text{maximize}} & L(\lambda)\end{array}\,, \quad (18)$$

where $L(\lambda)$ is the augmented dual function

$$L(\lambda) = \begin{array}{ll}\inf & \sum_{p=1}^{P}\frac{1}{P}\|x_p\|_1 + \sum_{\{i,j\}\in\mathcal{E}} \phi_{\lambda_{\{i,j\}}}(x_i - x_j) \\ \text{s.t.} & A_p x_p = b_p\,, \quad p = 1,\ldots,P\,,\end{array} \quad (19)$$

where $\phi_\lambda(z) = \lambda^\top z + \frac{\rho}{2}\|z\|^2$, and ρ is a positive parameter. The algorithms have an outer loop that solves (18), and an inner loop that solves the optimization problem in (19).

We consider three distributed, double-looped algorithms [22], [21], [23] to solve (18), and thus (3) because strong duality holds. While [22], [23] were designed to solve BP, [21] was designed to solve more general problems. We thus have to adapt the latter to our problem. The algorithms described in [22], [21], [23] will be denoted respectively by MM/NGS (method of multipliers and nonlinear Gauss-Seidel), MM/DQA (method of multipliers and diagonal quadratic approximation), and DN (double Nesterov).

All algorithms solve (18) with an iterative scheme in the outer loop. As in D-ADMM, the dual variable λ consists of several variables $\lambda_{\{i,j\}}$ associated with the edges $\{i,j\} \in \mathcal{E}$. It can be shown that the dual function $L(\lambda)$ in (19) is differentiable and that its gradient $\nabla L(\lambda) = (\ldots, x_i(\lambda) - x_j(\lambda), \ldots)$ is Lipschitz continuous with constant $1/\rho$ [44]. The vector $\bar{x}(\lambda) := (x_1(\lambda), x_2(\lambda), \ldots, x_P(\lambda))$ solves the optimization

problem in (19) for a fixed λ. The algorithm for solving this inner problem will be the inner loop and is considered later. These nice properties of $L(\lambda)$ enable the edge-wise application of the gradient method [40, §1.2]

$$\lambda_{\{i,j\}}^{(k+1)} = \lambda_{\{i,j\}}^{(k)} + \rho \nabla_{\lambda_{\{i,j\}}} L(\lambda^{(k)}), \tag{20}$$

or the edge-wise application of Nesterov's method [45]

$$\begin{aligned} \lambda_{\{i,j\}}^{(k+1)} &= \eta_{\{i,j\}}^{(k)} + \rho \nabla_{\eta_{\{i,j\}}} L(\eta^{(k)}) \\ \eta_{\{i,j\}}^{(k+1)} &= \lambda_{\{i,j\}}^{(k+1)} + \tfrac{k-1}{k+2}(\lambda_{\{i,j\}}^{(k+1)} - \lambda_{\{i,j\}}^{(k)}), \end{aligned} \tag{21}$$

to solve (18). Nesterov's method is proven to be faster than the gradient method. When we use the gradient method (20) to solve a dual problem, where duality here is seen in the augmented Lagrangian sense, the resulting algorithm is called method of multipliers (MM) [40, p.408]. While MM/NGS and MM/DQA use MM for their outer loop, DN uses (21).

So far, we assumed that a solution of the optimization problem in (19), for a given λ, was available. Nevertheless, solving this problem in a distributed way is more challenging than solving (18) (when $\nabla L(\lambda)$ is readily available). The reason is that we cannot decouple the term $\sum_{\{i,j\}\in\mathcal{E}} \phi_{\lambda_{\{i,j\}}}(x_i - x_j)$ into a sum of P functions, each one depending only on x_p. Both MM/NGS and MM/DQA use an iterative method that optimizes the objective of (19) w.r.t. one block variable x_p, while keeping the other blocks fixed. More concretely, let $g_\lambda(x_1, \ldots, x_P)$ denote the objective of (19) when λ is fixed. MM/NGS uses the nonlinear Gauss-Seidel (NGS) method [20, §3.3.5][46]:

$$\begin{aligned} x_1^{(t+1)} &= \arg\min_{x_1 \in X_1} g_\lambda(x_1, x_2^{(t)}, x_3^{(t)}, \ldots, x_P^{(t)}) \\ x_2^{(t+1)} &= \arg\min_{x_2 \in X_2} g_\lambda(x_1^{(t+1)}, x_2, x_3^{(t)}, \ldots, x_P^{(t)}) \\ &\vdots \\ x_P^{(t+1)} &= \arg\min_{x_P \in X_P} g_\lambda(x_1^{(t+1)}, x_2^{(t+1)}, x_3^{(t+1)}, \ldots, x_P), \end{aligned} \tag{22}$$

where $X_p := \{x_p : A_p x_p = b_p\}$, $p = 1, \ldots, P$. It can be proven that any limit point of the sequence generated by (22) solves problem (19); see [46], [37]. Each optimization problem in (22) is solved at one node. It turns out that these are equivalent to the problem in step 4 of Algorithm 2. Note that the nodes in (22) cannot operate in parallel, akin to the algorithm we propose here. MM/DQA, on the other hand, solves the problem in (19) with a parallel scheme called diagonal quadratic approximation (DQA):

$$\begin{aligned} u_1 &= \arg\min_{x_1 \in X_1} g_\lambda(x_1, x_2^{(t)}, x_3^{(t)}, \ldots, x_P^{(t)}) \\ u_2 &= \arg\min_{x_2 \in X_2} g_\lambda(x_1^{(t)}, x_2, x_3^{(t)}, \ldots, x_P^{(t)}) \\ &\vdots \\ u_P &= \arg\min_{x_P \in X_P} g_\lambda(x_1^{(t)}, x_2^{(t)}, x_3^{(t)}, \ldots, x_P) \\ x_p^{(t+1)} &= \tau u_p + (1-\tau) x_p^{(t)}, \quad p = 1, \ldots, P, \end{aligned} \tag{23}$$

where $\tau = 1/P$. For a proof that (23) solves (19) see [21], [37]. The difference between (22) and (23) is that the latter allows all the nodes to operate in parallel, and after the minimization step, each node combines the solution of the optimization problem it has just solved with the previous estimate of the solution: $x_p^{(t)}$. Note that a communication step has to occur after each iteration.

Regarding DN, we made some modifications to the inner loop of the method proposed in [23], so that we could get an algorithm comparable with what we propose here.

Double Nesterov (DN). In [23], BP is recast as a linear program by increasing the size of the variable to $2n$. The result is that the problem defining the dual function has a differentiable objective with a Lipschitz continuous gradient, and thus Nesterov's method is directly applicable. However, the size of the variable transmitted in each communication step is $2n$. Here, we do not recast BP as an LP. As seen before, the dual problem (18) is solved with Nesterov's method (21) in the outer loop. Now, to solve the optimization problem in (19), Nesterov's method is not applicable because the objective is not differentiable. However, that objective can be written as the sum of a nondifferentiable function $h(\bar{x}) = \sum_{p=1}^{P} \frac{1}{P}\|x_p\|_1$ with a differentiable one $g(\bar{x}) = \sum_{\{i,j\}\in\mathcal{E}} \phi_{\lambda_{\{i,j\}}}(x_i - x_j)$. The gradient of $g(\bar{x})$ w.r.t. x_p is $\nabla_{x_p} g(\bar{x}) = \gamma_p + \rho D_p x_p - \rho \sum_{j\in\mathcal{N}_p} x_j$. Therefore, to compute $\nabla_{x_p} g(\bar{x})$, each node needs only to communicate with its neighbors. The gradient $\nabla g(\bar{x})$ is Lipschitz continuous with constant $\rho\lambda_{\max}(\mathcal{L})$, where $\lambda_{\max}(\mathcal{L})$ denotes the maximum eigenvalue of the graph Laplacian. FISTA [47] is an algorithm that adapts Nesterov's method to this scenario. It operates the following way:

Algorithm 5 FISTA (for node p)

Initialization: choose $\alpha = 1/(\rho\lambda_{\max}(\mathcal{L}))$, $x_p^{(0)} = y_p^{(0)} = 0$, $t = 0$

1: **repeat**
2: $\quad u_p = y_p^{(t)} - \alpha \nabla g(y_p^{(t)})$
3: $\quad x_p^{(t+1)} = \arg\min_{x_p} \frac{1}{2\alpha}\|x_p - u_p\|^2 + h(x_p)$
4: $\quad y_p^{(k+1)} = x_p^{(k+1)} + \frac{k-1}{k+2}\left(x_p^{(k+1)} - x_p^{(k)}\right)$
5: $\quad k \leftarrow k+1$
6: **until** some stopping criterion is met

This modification to [23] allows us to compare the resulting algorithm with ours, because the size of the variable is now n. Furthermore, the problem in step 3 is equivalent to the one in step 4 of Algorithm 2.

Tuning parameter ρ. Note that all algorithms (except the subgradient) share the same tuning parameter ρ, because all are based on an augmented Lagrangian reformulation. It is known that ρ influences the convergence rate of augmented Lagrangian methods. Albeit there are self-adaptive schemes to update ρ during the algorithm [30], [48], [49], making the algorithms less sensitive to ρ, we were not able to implement these schemes in a distributed scenario. We will hence assume ρ is constant during the execution of the algorithms.

Execution times in wireless networks. In contrast with all the algorithms described here (except MM/NGS), D-ADMM assumes a coloring scheme based on which the nodes operate asynchronously. Suppose all the algorithms are implemented on an ideal network, where packet collisions do not occur, i.e., two neigboring nodes can transmit messages at the same time without causing interference at the reception. If a communication step by D-ADMM takes T time units, then a

Table I
ALGORITHMS FOR COMPARISON IN THE SIMULATIONS.

Acronym	Algorithm(s)	Source
D-ADMM	Alternating direction MM	This paper
Subgradient	Subgradient method	[19]
D-Lasso	Alternating direction MM	[29]
MM/NGS	MM + nonlinear Gauss-Seidel	[22]
MM/DQA	MM + diagonal quadratic approximation	[21]
DN	Nesterov + Nesterov	[23]

Table II
SCENARIOS FOR ROW PARTITION EXPERIMENTS.

Scenario	Sparco Id	m	n	P
1	—	500	2000	50
2	7	600	2560	50
3	3	1024	2048	64
4	902	200	1000	50
5	11	256	1024	64

communication step by the other algorithms takes T/C units, where C is the number of colors we used for the network (we are ignoring the optimizations that can be made from the procedure described in Figure 3). Therefore, although D-ADMM requires less communication steps, as shown next, it might actually take longer than competing algorithms. However, in a real wireless network, packet collisions occur and medium-access (MAC) protocols have to be implemented to avoid them. Hence, synchronous algorithms cannot operate synchronously in wireless networks. The execution time of an algorithm, among other factors, is highly dependent on the MAC protocol. Comparing execution times is thus beyond the scope of this paper.

V. EXPERIMENTAL RESULTS

In this section we compare our algorithm against the prior work discussed in the previous section and listed in Table I. We focus on the row-partitioned case since the algorithm for the column partition is derived from it. We start describing how the data and the networks were generated, and how the experiments were carried out. In the first type of experiments we compare all the algorithms on moderate-sized networks (around 50 nodes) and conclude that D-ADMM and D-Lasso are the "fastest" algorithms. In the second type of experiments we compare only these two algorithms in a more thorough way for the same networks, and we also see how their performance varies as the network size increases (from 2 nodes to 1024 nodes). Finally, we address the column partition case.

Experimental setup. We considered five distinct scenarios with different dimensions and different types of data, shown in Table II. The data (matrix $A \in \mathbb{R}^{m\times n}$ and vector $b \in \mathbb{R}^m$) was taken from the Sparco toolbox [50], except in scenario 1, where we used a 500×2000 matrix with i.i.d. Gaussian entries with zero mean and variance $1/\sqrt{m}$. In each scenario, each node stores $m_p = m/P$ rows of A. We ensured that $m_p = m/P$ is an integer by considering two values for P: 50 and 64, chosen depending on the scenario.

Table III
NETWORK MODELS FOR THE EXPERIMENTS.

Network number	Model	Parameters
1	Erdős-Rényi	$p = 0.25$
2	Erdős-Rényi	$p = 0.75$
3	Watts-Strogatz	$(n, p) = (4, 0.6)$
4	Watts-Strogatz	$(n, p) = (2, 0.8)$
5	Barabasi-Albert	——
6	Geometric	$d = 0.75$
7	Lattice	——

In the following, $x^\star$ denotes the solution of BP obtained by the Sparco toolbox, or in scenario 1, the one obtained by CVX [51], solving BP as a linear program. Note that due to the dimensions of the matrices and their randomness/structure, $x^\star$ is guaranteed to be unique with overwhelming probability.

For each scenario we ran all algorithms for the seven different networks shown in Table III. For each network in Table III we generated two networks: one with 50 nodes (used in scenarios with $P = 50$), the other with 64 nodes (used in scenarios with $P = 64$). The parameters of the networks were chosen so that the generated network would be connected with high probability. Only for network 4, $P = 50$ we did not get a connected network at first, so we changed the parameters to $(3, 0.8)$. If the generated network had self-connections or multiple edges between the same pair of nodes, we would remove them. We also generated 10 networks with 2^i nodes ($i = 1, \ldots, 10$), all following the model of network 3. These are used in the type II experiments (explained below).

The Erdős-Rényi model [52] has one parameter p, which specifies the probability of any two nodes in the network being connected. The Watts-Strogatz model [53] has two parameters: the number of neighbors n and the rewiring probability p. First it creates a lattice where every node is connected with n other nodes; then, every link is rewired, or not, with probability p. If a rewiring occurs in link $\{i, j\}$, then we pick node i or j (with equal probability) and connect it with other node in the network, chosen uniformly. The Barabasi-Albert model [54] starts with one node; at each step, one node is added to the network and is connected to one of the nodes already in the network. However, the probability of the new node "choosing" to connect to the other nodes is not uniform: it is proportional to the nodes' degrees such that the new node has a greater probability of connecting to the nodes with larger degrees. The geometric model [55] deploys P nodes randomly (uniformly) in the unit square; then, two nodes are connected if their distance is less than d. Finally, the Lattice model has no randomness. For P nodes, it generates a rectangular grid graph in the plane such that the shape is as square as possible (5×10 for $P = 50$ and 8×8 for $P = 64$). Each node has four neighbors except for the borders. This lattice network is the only one guaranteed to be bipartite, and thus Algorithm 2 is only guaranteed to converge for this network.

We used an heuristic from the Matgraph toolbox [56] to

Table IV
TYPES OF EXPERIMENTS.

Type of experiment	Value of ρ
I	$\rho = 1$ for D-ADMM and D-Lasso $\rho = 10$ for MM/NGS, MM/DQA, and DN
II	$\rho \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$ the value that leads to the best results is picked

find a coloring for these networks. It is then possible that the number of colors is larger than $\chi(\mathcal{G})$. We checked that the optimal solution of two colors was found for the Lattice model.

Results. As mentioned before, we keep the parameter ρ fixed during the execution of the algorithms (except for the subgradient method, which has no ρ). We picked ρ in two different ways, yielding two types of experiments, shown in Table IV. In type I, ρ was always the same for all scenarios and all networks: $\rho = 1$ for D-ADMM (Algorithm 2) and for D-Lasso (Algorithm 4), and $\rho = 10$ or the double-looped algorithms MM/NGS, MM/DQA, and DN. These values were chosen based on some pre-testing. In the type II experiments, given a fixed scenario and network, we execute each algorithm for several ρ's and pick the one that yields the best result. For the type II experiments, we only considered the best two algorithms: D-ADMM and D-Lasso.

The two types of experiments reflect two different philosophies in the assessment of algorithms that depend on parameters: type I represents real-world applications (the parameters are tuned for known data and are then used unchanged); type II is more suited to assess the true capabilities of the algorithm.

Type I experiments. Figure 5 shows the results of the type I experiments. The left-hand (resp. right-hand) side plots show, for each network, the number of communication steps until each algorithm achieves a precision of 1% (resp. $10^{-3}\%$) at a randomly selected node p. This means we count the number of communication steps until $\|x_p^{(k)} - x^\star\|/\|x^\star\| \leq 10^{-2}$ or 10^{-5}. We allowed a maximum number of 10^4 communication steps.

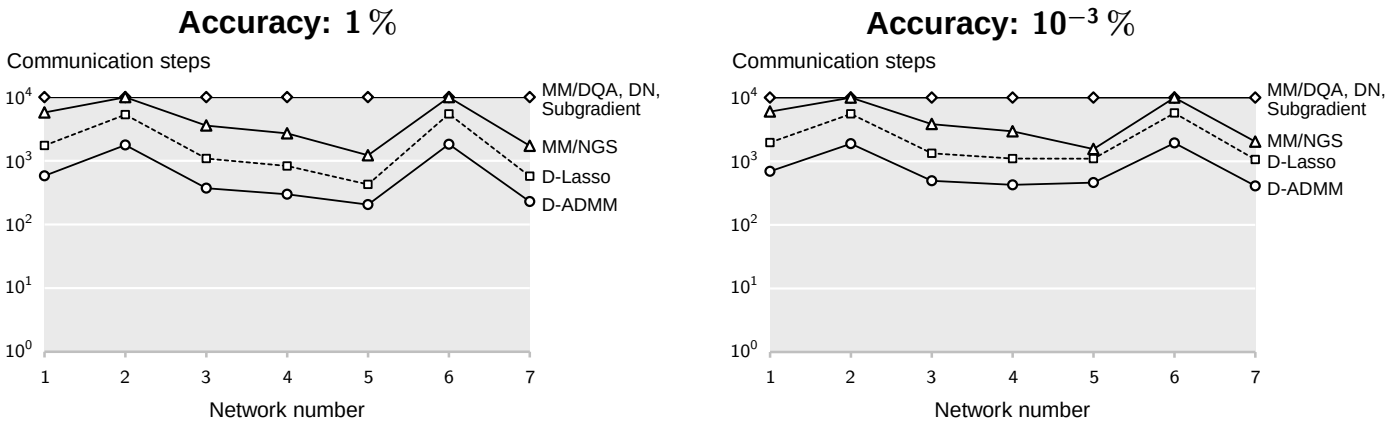
In Figure 5 we observe that the behavior of the algorithms in all scenarios, except in scenario 3, is identical, so we will focus only on scenarios 1 and 3. Figure 5(**??**SubFig:RPExperimentsScen1) shows that, for scenario 1, D-ADMM requires the least number of communications to achieve both accuracies regardless the network. We can also see that for this scenario MM/DQA, DN, and Subgradient always reached the maximum number of 10^4 iterations before achieving any of the prescribed accuracies. As stated before, the behavior of the algorithms for the remaining scenarios (except scenario 3) is very similar. In scenario 3, Figure 5(**??**SubFig:RPExperimentsScen3), we see a different behavior: while D-ADMM required less communications than any of the ρ-dependent algorithms, the Subgradient required less communications to achieve the accuracy 1% for networks 1, 2, and 6. However, if we let the algorithms continue executing, the Subgradient reaches the maximum number of communications before achieving the $10^{-3}\%$ of accuracy, as can be seen in the right-hand plot of Figure 5(**??**SubFig:RPExperimentsScen3). Note that the relative behavior of the remaining algorithms is roughly the same for both accuracies.

In Figure 6 we show how the error of the estimate x_p at a random node p varies along the iterations, for each algorithm. Figure 6(a) shows the error for scenario 1 when the algorithms are executed in network number 4. Notice that the number of communications to achieve accuracies of 1% and $10^{-3}\%$ agree with the plots of Figure 5(a), for example D-ADMM takes less than 10^3 communication steps to achieve a 10^{-5} precision. Figure 6(b) shows the errors for scenario 3 when we use network 3 (cf. with the plots of Figure 5(c)). Note the similarity of the curves of D-ADMM and D-Lasso: they have the same shape but the D-ADMM error is always smaller. This might happen because both methods use the same internal algorithm, albeit applied to different reformulations. Finally, note in Figure 6(b) how the error of the Subgradient evolves for scenario 3, network 3: the rate of convergence is very fast at the beginning, but after the first 1000 iterations it becomes very slow. This agrees with what was observed in Figure 5(c).

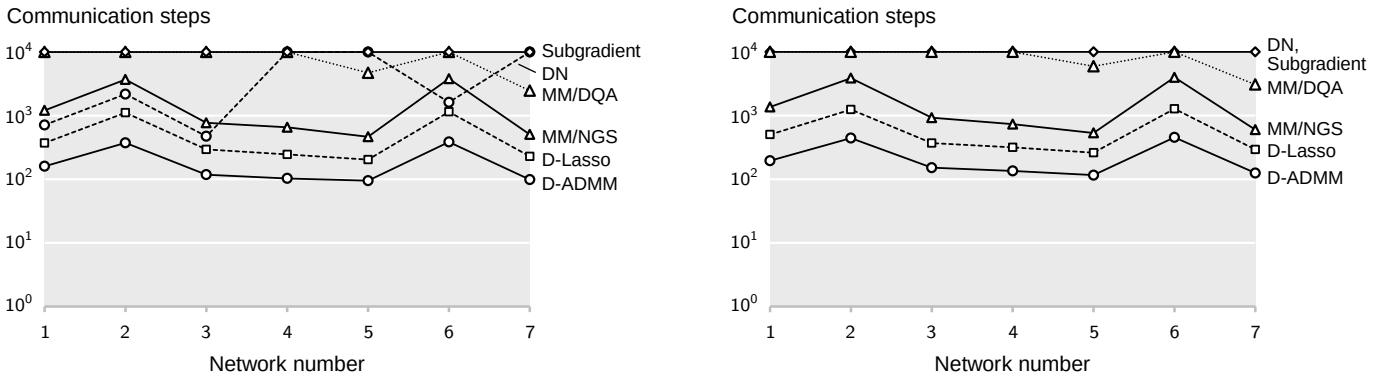
Type II experiments. For the type II experiments we only considered the two best algorithms: D-ADMM and D-Lasso. Figure 7 shows for each network the number of communication steps to reach an accuracy of $10^{-3}\%$. We allowed for maximally 3000 communication steps (these were only achieved by D-Lasso in scenario 3 for networks 3, 4, and 5, as can be seen in Figure 7(b)). We observed that the best values of ρ for D-ADMM were always 10^{-2}, 10^{-1}, or 1. For example, D-ADMM had the best performance for $\rho = 1$ for scenarios 1, 3, and 5 when the networks were either 5 or 7. For instance, for scenario 1, network 5 D-ADMM took 462 communication steps (see Figure 7(a)), the same number observed in the type I experiments, in right-hand plot of Figure 5(a). Recall that ρ was fixed at 1 for D-ADMM in the type I experiments. This also means that in the type II experiments the number of communications for D-ADMM decreased except for scenarios 1, 3, and 5 when the networks were either 5 or 7. The same phenomenon happened for D-Lasso: the optimal ρ was 1 only in scenarios 1 and 5 for the 5th network; and the optimal ρ's were 10^{-2}, 10^{-1}, or 1.

We conclude from Figure 7 that D-ADMM requires less communication steps than D-Lasso, independently of the scenario or network type. Excluding the cases D-Lasso reached the maximum number of iterations, we see that in average D-ADMM uses 51% of D-Lasso's number of communications (11% of standard deviation). The largest difference occurred in scenario 3, network 6, where D-ADMM used 35% of the communications D-Lasso used; this number was 78% for scenario 4, network 1, the smallest difference that occurred.

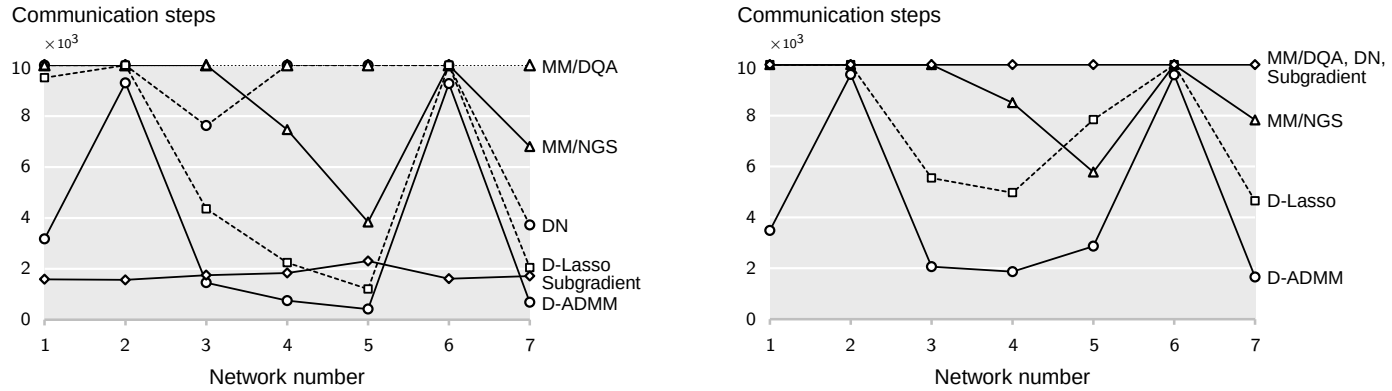
Figure 8 shows another type II experiment: we fixed the scenario and network type: Scenario 3, Watts-Strogatz with parameters $(4, 0.6)$; and observed how the number of communication steps varies as the size of the network increases. The number of nodes varied from 2 (each node stores 512 rows) to 1024 (each node stores 1 row) and was always a power of 2. D-ADMM and D-Lasso stopped after reaching 0.1% of accuracy. As shown by the gray straight lines in Figure 8, the communication steps in both algorithms increases approx-



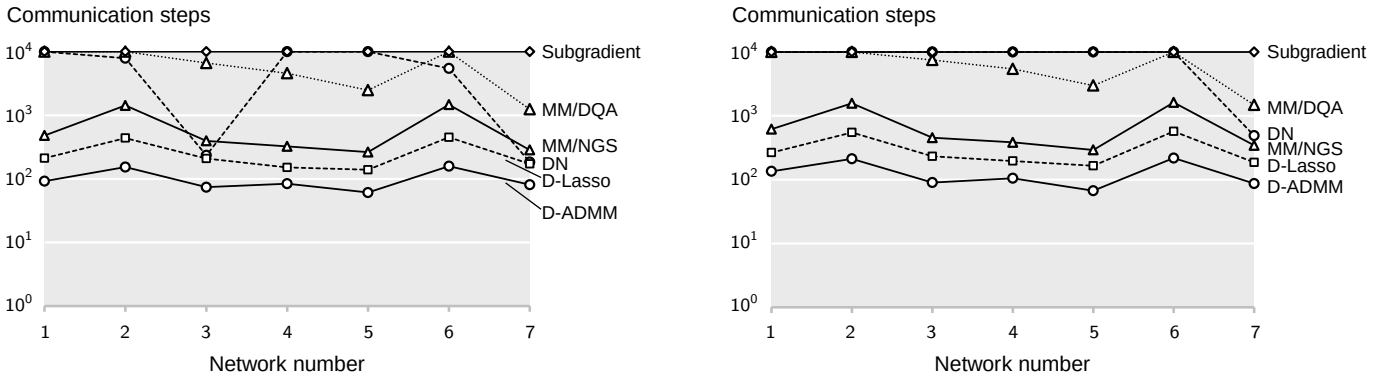
(a) Scenario 1



(b) Scenario 2

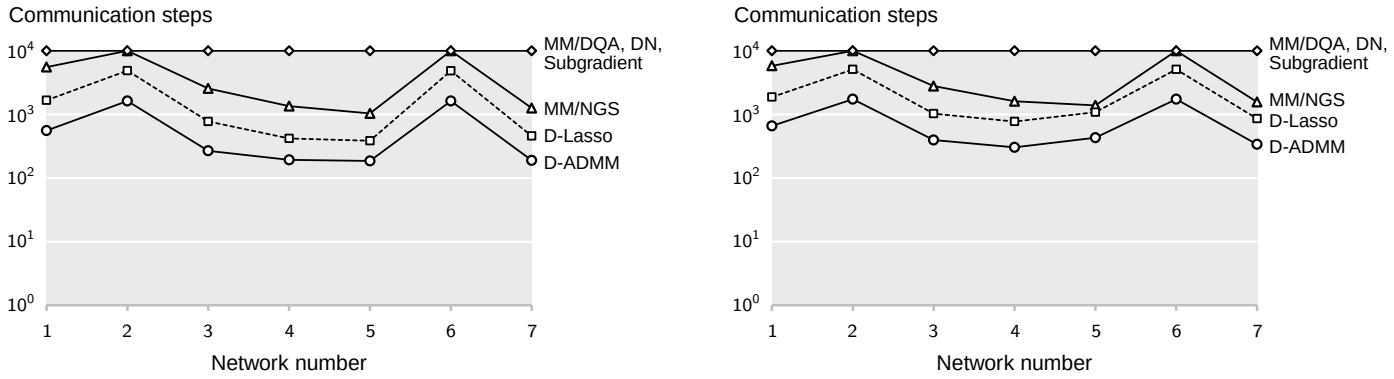


(c) Scenario 3



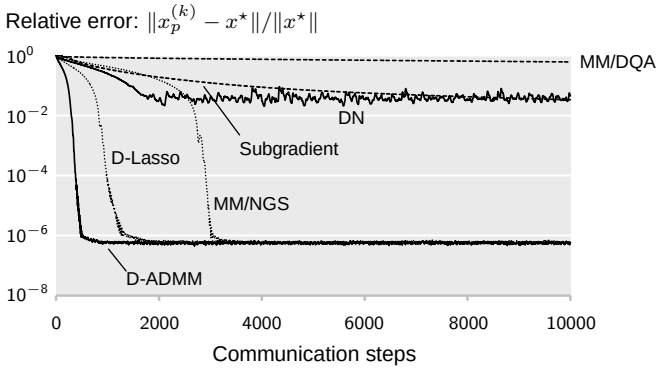
(d) Scenario 4

Figure 4. Type I experiments: number of communication steps to reach accuracies of 1% and $10^{-3}\%$ as a function of the network (see Table III).

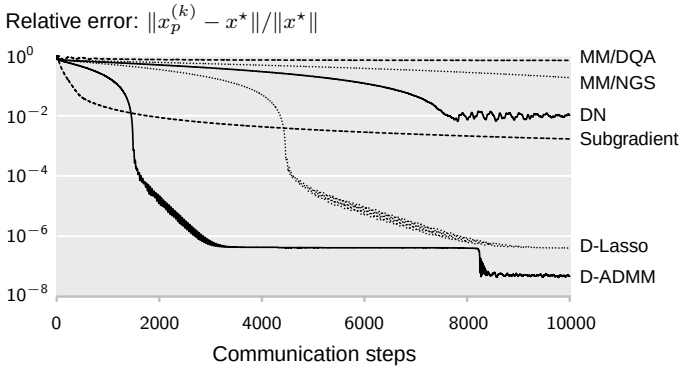


(a) Scenario 5

Figure 5. Type I experiments: number of communication steps to reach accuracies of 1% and $10^{-3}\%$ as a function of the network (see Table III).



(a) Scenario 1, network 4

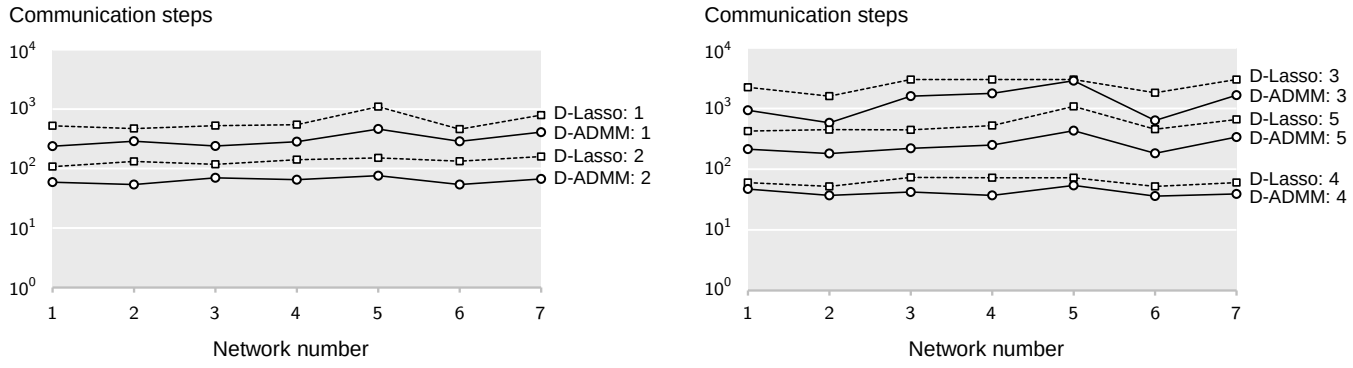


(b) Scenario 3, network 3

Figure 6. Type I experiments: errors along the iterations (communication steps) of the algorithms for fixed scenarios and networks.

imately linearly in a log-log plot. The model we used to compute those lines was $\log_{10} C = \alpha \log_2 P + \beta$, where C is the number of communication steps, P the number of nodes, and α and β are the parameters to be found for each line. The minimum least squares error yielded $(\alpha, \beta) = (0.243, 1.07)$ for D-ADMM and $(\alpha, \beta) = (0.233, 1.47)$ for D-Lasso. Therefore, $C \simeq 11.7 \cdot P^{0.8}$ for D-ADMM and $C \simeq 29.5 \cdot P^{0.77}$ for D-Lasso, showing a less-than-linear increase of the communication steps with the number of nodes, for both algorithms. Also, the difference between the lines' offsets reveals that D-Lasso took in average 2.5 times more communications than D-ADMM. The average number of colors was 4.6, which means that in a collision-free network D-ADMM would be 1.8 times slower than D-Lasso. Again, the optimal ρ's were 10^{-2}, 10^{-1}, or 1, but we noticed a curious pattern on both algorithms: the optimal value for ρ decreased as the size of the network increased.

Results for the column partition. For the column partition we only executed type II experiments. While the scenarios were the same as before (Table II), we changed the networks: they now have 10 nodes (for scenarios 1, 2, and 4) or 8 nodes (for scenarios 3 and 5). All nodes thus store the same number of columns, i.e., the number of columns n is divisible by the number of nodes P. The model for generating these networks



(a) Scenarios 1 and 2

(b) Scenarios 3, 4, and 5

Figure 7. Type II experiments: number of communication steps to reach $10^{-3}\%$ of accuracy or 3000 communication steps.

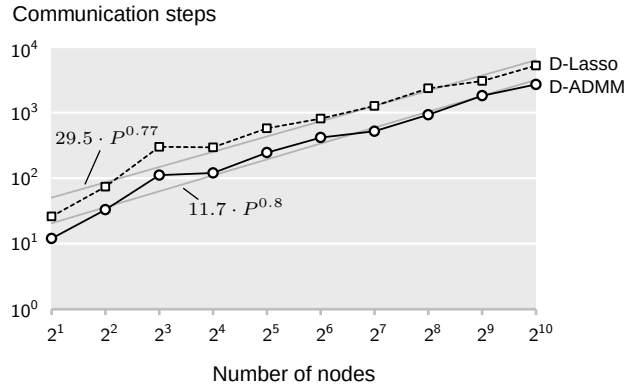


Figure 8. Type II experiments for row partition: number of communication steps to reach 0.1% of accuracy as a function of the network size. The straight lines represent a linear fit.

is the same as in Table III. In all experiments we set the regularization parameter to $\delta = 10^{-3}$, a value that always allowed the recovery of the solution of BP, as we will see.

Figure 9 shows the plots with the results of the type II experiments. As before, D-ADMM always required less communication steps to achieve a $10^{-3}\%$ of accuracy. In particular, D-ADMM used in average 42% of the communications D-Lasso used; the standard deviation was 10%. The largest difference in the number of communications occurred in scenario 2, network 4, where D-ADMM only used 28% of the communications that D-Lasso used. The smallest difference was 72% and it occurred in scenario 5, network 5. We mention that, in contrast with the row partition, there were cases in which the optimal value for ρ was 10^{-3} and 10 (cf. Table IV), the "boundary" values of the set of ρ's we used. Therefore, we might improve the results if we try a wider range of ρ's.

VI. Final Remarks and Conclusions

We proposed an algorithm for solving BP in two distributed frameworks. In one framework, the BP matrix is partitioned by rows, with its rows distributed over a network with an arbitrary number of nodes; in the other framework, it is the columns of the matrix that are distributed. The only requirement on the topology of the network through which the nodes communicate is connectivity (and we also assume that this topology does not change along the algorithm). Therefore, our algorithms can be applied to several scenarios, ranging from sensor networks, where the communication network is usually sparse, to super-computing platforms, characterized by dense networks.

We simulated our algorithms for several types of data and networks and conclude that they always require less communications than competing algorithms. This is paramount in energy-constrained environments such as sensor networks.

References

[1] S. Chen, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comp.*, vol. 20, no. 1, pp. 33–61, 1998.

[2] A. Bruckstein, D. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Review*, vol. 51, no. 1, pp. 34–81, 2009.

[3] E. Candès and T. Tao, "Decoding by linear programming," *IEEE Trans. Info. Th.*, vol. 51, no. 12, pp. 4203–4215, 2005.

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

[5] E. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Trans. Info. Th.*, vol. 52, no. 2, pp. 489–509, 2006.

[6] D. Donoho, "Compressed sensing," *IEEE Trans. Info. Th.*, vol. 52, no. 4, pp. 1289–1306, 2006.

[7] J. Tropp, "Just relax: Convex programming methods for identifying sparse signals," *IEEE Trans. Info. Theory*, vol. 51, no. 3, pp. 1030–1051, 2006.

[8] E. Candès and M. Wakin, "An introduction to compressive sampling," *IEEE Sig. Proc. Mag.*, vol. 25, no. 2, 2008.

[9] I. Akyildiz, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, 2002.

[10] B. Krishnamachari, *Networking Wireless Sensors*, Cambridge University Press, 2005.

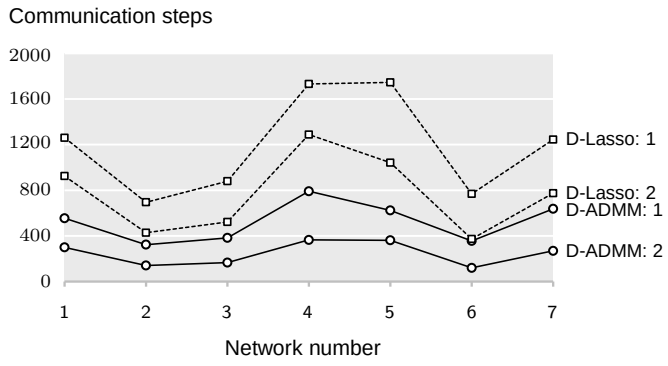
[11] J. Tropp, J. Laska, M. Duarte, J. Romberg, and R. Baraniuk, "Beyond nyquist: Efficient sampling of sparse bandlimited signals," *IEEE Trans. Info. Th.*, vol. 56, no. 1, pp. 520–544, 2010.

[12] E. Candès and T. Tao, "Near-optimal signal recovery from random projections: Universal encoding strategies?," *IEEE Trans. Info. Th.*, vol. 52, no. 12, pp. 5406–5425, 2006.

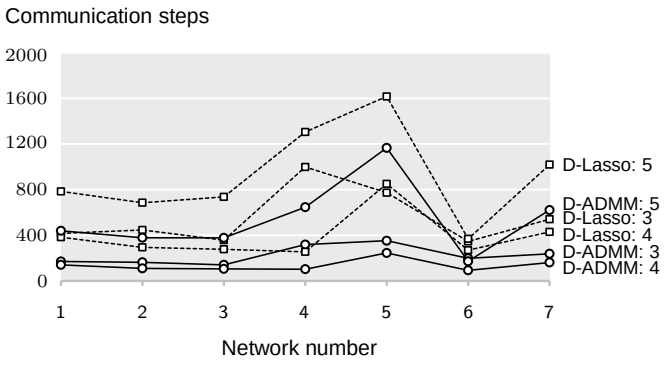
[13] J. Meng, L. Husheng, and Z. Han, "Sparse event detection in wireless sensor networks using compressive sensing," in *43rd Annual Conf. on Info. Sciences and Systems (CISS)*, 2009.

[14] V. Cevher, M. Duarte, and R. Baraniuk, "Distributed target localization via spatial sparsity," in *16th European Sig. Proc. Conf. (Eusipco)*, 2008.

[15] J. Romberg, R. Neelamani, C. Krohn, J. Krebs, M. Deffenbaugh, and J. Anderson, "Efficient seismic forward modeling using simultaneous random sources and sparsity," in *Soc. Expl. Geophysicists Annual Meeting*, 2008.



(a) Scenarios 1 and 2



(b) Scenarios 3, 4, and 5

Figure 9. Type II experiments for the column partition: number of communication steps to reach $10^{-3}\%$ of accuracy.

[16] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*, Athena Scientific, 1997.

[17] C. Stunkel and D. Reed, "Hypercube implementation of the simplex algorithm," in *Proc. 3rd Conf. Hypercube on concurrent computers and applications*, 1989, vol. 2, pp. 1473–1482.

[18] H. Dutta and H. Kargupta, "Distributed linear programming and resource management for data mining in distributed environments," in *IEEE Inter. Conf. Data Mining Workshops*, 2008, pp. 543–552.

[19] I. Lobel, A. Ozdaglar, and D. Feijer, "Distributed multi-agent optimization with state-dependent communication," Tech. Rep., LIDS report 2834, 2010.

[20] D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Athena Scientific, 1997.

[21] A. Ruszczynski, "Augmented lagrangian decomposition for sparse convex optimization," *Inter. Inst. Applied Systems Analysis*, 1992.

[22] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "Distributed algorithms for basis pursuit," in *2nd Intern. Workshop Sig. Proc. with Adaptive Sparse Structured Representations, Saint-Malo, France*, 2009.

[23] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "Basis pursuit in sensor networks," in *IEEE Proc. Inter. Conf. Acoustics, Speech, and Sig. Proc. (ICASSP)*, 2011.

[24] E. Berg and M. Friedlander, "Probing the pareto frontier for basis pursuit solutions," *SIAM J. Sci. Comput.*, vol. 31, no. 2, pp. 890–912, 2008.

[25] Z. Wen, W. Yin, and Y. Zhang, "A fast algorithm for sparse reconstruction based on shrinkage, subspace optimization and continuation," *SIAM J. Sci. Comp.*, vol. 32, no. 4, pp. 1832–1857, 2010.

[26] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *Annals of Statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[27] M. Afonso, J. Bioucas-Dias, and M. Figueiredo, "An augmented lagrangian approach to the constrained optimization formulation of imaging inverse problems," *IEEE Trans. Im. Proc.*, vol. 20, no. 3, pp. 681–695, 2011.

[28] S. Becker, J. Bobin, and E. Candès, "NESTA: a fast and accurate first-order method for sparse recovery," Tech. Rep., Caltech, 2009.

[29] J. Bazerque and G. Giannakis, "Distributed spectrum sensing for cognitive radio networks by exploiting sparsity," *IEEE Trans. Sig. Proc.*, vol. 58, no. 3, pp. 1847–1862, 2010.

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

[31] S. Ram, A. Nedić, and V. Veeravalli, "A new class of distributed optimization algorithms: application to regression of distributed data," *Optim. Methods and Software*, pp. 1029–4937, 2010.

[32] M. Garey and D. Johnson, *Computers and Intractability*, W. H. Freeman and Co., 1979.

[33] F. Kuhn and R. Wattenhofer, "On the complexity of distributed graph coloring," in *PODC'06 Proc. 25th anual ACM symposium Principles of distributed computing*, 2006.

[34] D. Leith and P. Clifford, "Convergence of distributed learning algorithms for optimal wireless channel allocation," in *IEEE Inter. Conf. Decision and Contr. (CDC)*, 2006, pp. 2980–2985.

[35] K. Duffy, N. Connell, and A. Sapozhnikov, "Complexity analysis of a decentralised graph colouring algorithm," *Info. Proc. Letters*, 2008.

[36] N. Linial, "Locality in distributed graph algorithms," *SIAM J. Comput.*, vol. 21, no. 1, pp. 193–201, 1992.

[37] J. Mota, "Distributed algorithms for sparse approximation," M.S. thesis, IST, Portugal, 2008, http://users.isr.ist.utl.pt/~jmota/publications.html.

[38] M. Friedlander, "Exact regularization of linear programs," Tech. Rep., Univ. of British Columbia, 2006.

[39] O. Mangasarian and R. Meyer, "Nonlinear perturbation of linear programs," *SIAM J. Contr. Optim.*, vol. 17, no. 6, pp. 745–752, 1979.

[40] D. Bertsekas, *Nonlinear Programming*, Athena Scientific, 2nd edition, 1999.

[41] M. Raydan, "The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem," *SIAM J. Optim.*, vol. 7, no. 1, pp. 26–33, 1997.

[42] A. Nedić and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Tran. Aut. Contr.*, vol. 54, no. 1, 2009.

[43] M. DeGroot, "Reaching a consensus," *J. American Statistical Association*, vol. 69, no. 345, pp. 118–121, 1974.

[44] I. Konnov, *Equilibrium models and variational inequalities*, vol. 210, Elsevier, 2007.

[45] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Kluwer Academic Publishers, 2003.

[46] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *J. Optim. Th. and App.*, vol. 109, pp. 475–494, 2001.

[47] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Im. Sc.*, vol. 2, no. 1, pp. 183–202, 2009.

[48] B. He, H. Yang, and S. Wang, "Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities," *J. Optim. Th. and App.*, vol. 106, no. 2, pp. 337–356, 2000.

[49] S. Wang and L. Liao, "Decomposition method with a variable parameter for a class of monotone variational inequality problems," *J. Optim. Th. and App.*, vol. 109, no. 2, pp. 415–429, 2001.

[50] E. Berg, M. Friedlander, G. Hennenfent, F. Herrmann, R. Saab, and Ö. Yilmaz, "Sparco: a testing framework for sparse reconstruction," Tech. Rep., Dept. Computer Science, University of British Columbia, Vancouver, 2007.

[51] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 1.21," .

[52] P. Erdős and A. Rényi, "On random graphs," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.

[53] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 409–10, 1998.

[54] A. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.

[55] M. Penrose, *Random Geometric Graphs*, Oxford University Press, 2004.

[56] E. Scheinerman, "Matgraph: a graph theory toolbox for MATLAB," http://www.ams.jhu.edu/~ers/matgraph/.

[57] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "A proof of convergence for the alternating direction method of multipliers applied to polyhedral-constrained functions," http://arxiv.org/abs/1112.2295, 2011.

[58] B. He and X. Yuan, "On the O(1/t) convergence rate of alternating direction method," http://www.optimization-online.org/DB_HTML/2011/09/3157.html, 2011.

[59] P. Combettes and J. Pesquet, *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, chapter Proximal splitting methods in signal processing, Springer, 2010.
[60] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging," *J. Math. Imaging and Vision*, vol. 40, no. 1, 2011.
[61] B. He, M. Tao, and X. Yuan, "Alternating direction method with Gaussian back substitution for separable convex programmming," http://www.optimization-online.org/DB_HTML/2010/12/2871.html, 2010.
[62] B. He, M. Tao, and X. Yuan, "A splitting method for separate convex programming with linking linear constraints," http://www.optimization-online.org/DB_HTML/2010/06/2665.html, 2010.
[63] B. He, M. Tao, M. Xu, and X. Yuan, "Alternating directions based contraction method for generally separable linearly constrained convex programming problems," http://www.optimization-online.org/DB_HTML/2009/11/2465.html, 2010.
[64] B. He and X. Yuan, "Linearized alternating direction method with gaussian back substitution for separable convex programming," http://www.optimization-online.org/DB_HTML/2011/10/3192.html, 2011.
[65] Y. Narushima, T. Wakamatsu, and H. Yabe, "Extended Barzilai-Borwein method for the unconstrained minimization problems," *Pacific J. Opt.*, vol. 6, no. 3, 2010.

APPENDIX A
ALTERNATING DIRECTION METHOD OF MULTIPLIERS

Let f and g be two real-valued convex functions and X and Y two polyhedral sets. Let also A and B be two full column-rank matrices, and consider the problem

$$\begin{array}{ll}\underset{x\in X, y\in Y}{\text{minimize}} & f(x)+g(y)\\ \text{subject to} & Ax+By=0\,,\end{array} \tag{24}$$

with variables x and y. The *alternating direction method of multipliers* (ADMM) [30], [20] solves (24) by applying the method of multipliers [40, p.408] concatenated with one single loop of the nonlinear Gauss-Seidel [40, p.272]:

$$x^{(k+1)} = \arg\min_{x\in X} f(x)+\phi_{\lambda^{(k)}}(Ax+By^{(k)}) \tag{25}$$

$$y^{(k+1)} = \arg\min_{y\in Y} g(y)+\phi_{\lambda^{(k)}}(Ax^{(k+1)}+By) \tag{26}$$

$$\lambda^{(k+1)} = \lambda^{(k)}+\rho(Ax^{(k+1)}+By^{(k+1)})\,, \tag{27}$$

where $\phi_\lambda(z)=\lambda^\top z+\frac{\rho}{2}\|z\|^2$ and $\rho>0$ is a tuning parameter. In words, the augmented Lagrangian

$$L(x,y;\lambda)=f(x)+g(y)+\lambda^\top(Ax+By)+\frac{\rho}{2}\|Ax+By\|^2\,,$$

is first minimized with respect to x and then, keeping the value of x fixed at the just computed value $x^{(k+1)}$, the augmented Lagrangian is minimized with respect to y. Thus, (25) and (26) cannot be carried out simultaneously. After these minimization steps, the dual variable λ is updated in a gradient-based way via (27). The following theorem guarantees its convergence.

Theorem 4 ([30], [20], [57])**.** *Let $f:\mathbb{R}^{n_1}\to\mathbb{R}$ and $g:\mathbb{R}^{n_2}\to\mathbb{R}$ be convex over $\mathbb{R}^{n_1}$ and $\mathbb{R}^{n_2}$, respectively. Let $X\subset\mathbb{R}^{n_1}$ and $X\subset\mathbb{R}^{n_2}$ be polyhedral sets and let A and B two full column-rank matrices. Assume* (24) *is solvable. Then,*

1) *$\{(x^{(k)},y^{(k)})\}$ converges to a solution of* (24)*;*
2) *$\{\lambda^{(k)}\}$ converges to a solution of the dual problem*

$$\underset{\lambda}{\text{maximize}} \quad F(\lambda)+G(\lambda)\,,$$

where $F(\lambda)=\inf_{x\in X} f(x)+\lambda^\top Ax$ and $G(\lambda)=\inf_{y\in Y} g(y)+\lambda^\top By$.

Furthermore, [58] recently proved that ADMM converges with rate $O(1/k)$. This rate holds even if the quadratic term of $\phi_\lambda(z)$ in (25) is linearized, which can many times simplify the solution of that optimization problem. For more properties of ADMM and its relation to other algorithms see [59], [60].

We now present a generalization of ADMM, which we call "generalized ADMM." The generalized ADMM solves:

$$\begin{array}{ll}\text{minimize} & \sum_{i=1}^I f_i(x_i)\\ \text{subject to} & x_i\in X_i\,,\quad i=1,\ldots,I\\ & \sum_{i=1}^I A_ix_i=0\,,\end{array} \tag{28}$$

where $(x_1,\ldots,x_I)$ is the variable, $I>2$, the functions f_i are convex, A_i are full column-rank matrices, and X_i are polyhedral sets. The generalized ADMM solves (28) with:

$$\begin{aligned}
x_1^{(k+1)} &= \arg\min_{x_1\in X_1} f_1(x_1)+\phi_{\lambda^{(k)}}(A_1x_1+\sum_{j>1}A_jx_j^{(k)})\\
&\vdots\\
x_i^{(k+1)} &= \arg\min_{x_i\in X_i} f_i(x_i)+\phi_{\lambda^{(k)}}(A_ix_i+\sum_{j<i}A_jx_j^{(k+1)}\\
&\qquad\qquad +\sum_{j>i}A_jx_j^{(k)})\\
&\vdots\\
x_I^{(k+1)} &= \arg\min_{x_I\in X_I} f_I(x_I)+\phi_{\lambda^{(k)}}(A_Ix_I+\sum_{j<I}A_jx_j^{(k+1)})\\
\lambda^{(k+1)} &= \lambda^{(k)}+\rho\sum_{i=1}^I A_ix_i^{(k+1)}\,.
\end{aligned}$$

This algorithm is then the natural generalization of (25)-(27). It is not known yet if Theorem 4 also applies to the generalized ADMM. The latest efforts for doing that can be found in [61], [62], [63], [64]. In spite of this fact, we apply the generalized ADMM in this paper and the resulting algorithm never failed to converge in our simulations.

APPENDIX B
PROBLEM FOR EACH NODE: ROW PARTITION

In the distributed algorithm we propose, each node has to solve, in each iteration, the problem

$$\begin{array}{ll}\text{minimize} & \|x\|_1+v^\top x+c\|x\|^2\\ \text{subject to} & Ax=b\,,\end{array} \tag{29}$$

where $x\in\mathbb{R}^n$ is the variable, and $v\in\mathbb{R}^n$, $c>0$, $A\in\mathbb{R}^{m\times n}$, and $b\in\mathbb{R}^m$ are given. We propose to solve (29) by solving its dual problem:

$$\underset{\lambda}{\text{maximize}} \quad \lambda^\top b+\sum_{i=1}^n \inf_{x_i}\left(|x_i|+u_i(\lambda)x_i+cx_i^2\right)\,, \tag{30}$$

where the dual variable is $\lambda\in\mathbb{R}^m$ and $u(\lambda)=v-A^\top\lambda$. To compute the objective of this dual problem for a fixed λ, we need to find the minimum $x_i(\lambda)$ of the function $|x_i|+u_i(\lambda)x_i+cx_i^2$ for $i=1,\ldots,n$. Each one of these functions is strictly convex due to $c>0$, and hence it has a unique minimizer $x_i(\lambda)$. It follows from Danskin's theorem [40, prop. B25] that the objective of (30) is differentiable with

gradient $b - Ax(\lambda)$, where $x(\lambda) = (x_1(\lambda), \ldots, x_n(\lambda))$. By the optimal conditions for convex problems [40, prop.B24],

$$x_i(\lambda) = \begin{cases} 0 & , -1 \leq u_i(\lambda) \leq 1 \\ -(u_i(\lambda)+1)/(2c) & , u_i(\lambda) < -1 \\ -(u_i(\lambda)-1)/(2c) & , u_i(\lambda) > 1 \end{cases} .$$

The unicity of the minimizers $x_i(\lambda)$ also implies that, once a solution $\lambda^\star$ of (30) is known, the solution of (29) is given by $x(\lambda^\star)$. To solve (30), we propose using the method in [41], a very efficient Barzilai-Borwein (BB) algorithm. Per iteration, BB consumes $O(n)$ flops plus the flops necessary to compute the gradient. Furthermore, BB is known to converge R-superlinearly for generic unconstrained optimization problems [65, Th.4].

As a final note, the number of iterations to solve (29) can be drastically reduced by using warm-starts. This means that, at iteration $k+1$, node p will solve (29) by starting the BB algorithm with the solution found in iteration k. The solutions of these two consecutive problems are expected to be close, since only v and c changed, possibly just by a small quantity.