D-ADMM: A Communication-Efficient Distributed Algorithm For Separable Optimization

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

***Abstract*—We propose a distributed algorithm, named D-ADMM, for solving separable optimization problems in networks of interconnected nodes or agents. In a separable optimization problem, the cost function is the sum of all the agents' private cost functions, and the constraint set is the intersection of all the agents' private constraint sets. We require the private cost function and constraint set of a node to be known by that node only, during and before the execution of the algorithm. The application of our algorithm is illustrated with problems from signal processing and control, namely average consensus, compressed sensing, and support vector machines. It is well known that communicating in distributed environments is the most energy/time-demanding operation. Thus, algorithms using less communications are more prone to make networks live longer, e.g., sensor networks, or to execute faster, e.g., in supercomputing platforms. Through simulations for several network types and problems, we show that our algorithm requires less communications than the state-of-the-art algorithms.**

***Index Terms*—Distributed algorithms, alternating direction method of multipliers, consensus, compressed sensing, machine learning, sensor networks.**

I. Introduction

Recently, there has been a growing interest in distributed methods for data processing. Such interest was triggered both by the need of processing large quantities of data in distributed platforms, such as supercomputers, and by applications where data comes from spatially different locations and central processing is costly or impractical, as in sensor networks. On the other hand, over the last decades, convex optimization theory has proven to be an invaluable tool for deriving data processing algorithms [1]. Standard algorithms like interior-point methods are, however, unable to handle either large-scale or distributed problems. To solve these problems we thus need new algorithms.

In this paper, we aim to solve separable optimization problems in a distributed, decentralized way. In a separable optimization problem, the cost function can be decomposed as a sum of P functions f_p and the constraint set can be decomposed as the intersection of P sets X_p:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f_1(x) + f_2(x) + \cdots + f_P(x) \\ \text{subject to} & x \in X_1 \cap X_2 \cap \cdots \cap X_P \,, \end{array} \tag{1}$$

where $x \in \mathbb{R}^n$ is the global optimization variable. We will denote any solution of (1) with $x^\star$. We associate with problem (1) a network of P nodes, where only node p has access to its private cost function f_p and to its private constraint set X_p. This situation is illustrated in Figure 1. Each node can communicate with its neighbors using the network infrastructure. A method that does not use any kind of special or central node will be called *distributed algorithm*. Thus, in a distributed algorithm, aggregating data is not allowed; instead, each node has to exchange information only with its neighbors in order to achieve the common goal: arriving at a solution of (1). We propose a novel distributed algorithm based on the alternating direction method of multipliers (ADMM) for solving (1).

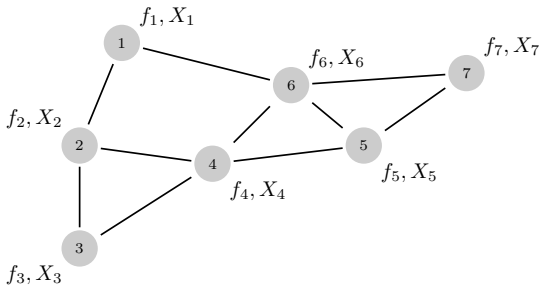


Figure 1. Network with $P = 7$ nodes. Node p only knows f_p and X_p, but cooperates with its neighbors in order to solve (1).

In applications such as sensor networks, nodes are battery-operated devices and thus energy is a scarce resource. It is well known that communication is the most energy-consuming operation in such a device [2]; therefore, algorithms using less communications can increase the lifespan of the network. Similarly, communication is known to be the algorithmic bottleneck in applications with more controlled environments, for example, in network protocols [3] or supercomputing platforms [4]. Algorithms using few communications are thus required for such applications.

We use our proposed distributed algorithm to solve several instances of (1) from several areas in engineering, namely average consensus, compressed sensing problems, and support vector machines. Our simulations show that, for each of these problems and for almost all network types, our proposed algorithm requires significantly less communication than prior

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 following grants from Fundação para a Ciência e Tecnologia (FCT): CMU-PT/SIA/0026/2009, PTDC/EEA-ACR/73749/2006, PEst-OE/EEI/LA0009/2011, and SFRH/BD/33520/2008 (through the Carnegie Mellon/Portugal Program managed by ICTI).

state-of-the-art algorithms, including those that are specifically designed for a particular problem and are not applicable to the entire problem class (1).

Next we formally state the problem we solve; then we give an overview of related work.

Formal problem statement. Given a network with P nodes, we associate f_p and X_p in (1) with the pth node of the network, $p = 1, \ldots, P$. We make the following assumptions:

Assumption 1. *Each $f_p : \mathbb{R}^n \to \mathbb{R}$ is a convex function over $\mathbb{R}^n$, and each set X_p is closed and convex.*

Assumption 2. *Problem* (1) *is solvable.*

Assumption 3. *The network is connected and its topology does not vary with time.*

Assumption 1 involves the concepts of convexity of a function and convexity of a set. We assume the reader is familiar with these concepts, as well as with Lagrangian duality [1], [5]. The second assumption states that there is at least one vector $x^\star$ that solves (1). This implies that the optimal cost $f^\star := \sum_{p=1}^{P} f_p(x^\star)$ is finite (and that $\cap_{p=1}^{P} X_p \neq \emptyset$). In Assumption 3, a network is connected if there is a path between every pair of nodes.

Under the previous assumptions, we solve the following problem: *given a network, design a distributed algorithm that solves* (1). By distributed algorithm, we mean that there is no notion of a central or special node; also, no node in the network, except node p, has access to f_p or X_p at any time before or during the algorithm.

Related work. The first method addressing the generic problem (1) was based on an average consensus algorithm coupled with a subgradient method [6], [7]. Such method has been extensively studied and is known to be robust to noise and link failures; however, it generally requires too many iterations to converge.

Another approach for solving (1) in a distributed way is via augmented Lagrangian methods, which consist of two loops: an outer loop updating the dual variables, and an inner loop updating the primal variables. There are some options for the algorithms in each loop. For the outer loop, the most common is the gradient method [8], [9], [10], which degenerates in the so-called method of multipliers [11]. For the inner loop, common choices are the nonlinear Gauss-Seidel method [9], [10] and Jacobi type algorithms, such as the diagonal quadratic approximation [8]. In [12] we applied fast gradient algorithms [13] in both loops. We mention that, although [8], [9], [12] do not address (1) specifically, they can be easily adapted to solve it, as explained in section III.

A special augmented Lagrangian-based algorithm, which uses one loop only, is the alternating direction method of multipliers (ADMM) [14], [15], [11]. In fact, ADMM is the method of multipliers concatenated with one iteration of the nonlinear Gauss-Seidel, and therefore, it consists of just one loop. ADMM is not directly applicable to (1): one has to reformulate that problem first. There are several options for such reformulation. For example, [16] proposes a distributed algorithm for solving specific instance of (1), based on a particular reformulation (the works [17], [18], [19], [20] apply the same algorithm to other instances of (1)). Another possible reformulation of (1) is explored in [21], [22], which also proposes an ADMM-based algorithm, but in contrast with [16], it uses two communication steps per iteration, or in other words, each node uses information from its second-order neighborhood. A comparison between [16] and [21] for the average consensus problem is provided in [20]; it is reported that both methods have similar convergence rates (and thus [21] uses twice as much communications as [16]), while [16] shows more resilience to noise. The algorithm we propose in this paper is also based on ADMM, but applied to a new reformulation of (1). We show through extensive simulations that the proposed algorithm requires less communications than any of the previous approaches.

All the above algorithms can solve (1) in a distributed way and will be briefly described in section III. There are, however, other algorithms that can solve (1), but are not distributed. For example, [23] applies ADMM to distributed scenarios, but the resulting algorithms are not distributed in the sense that they require a central node or a special network topology. Our algorithm and the ones described before, in contrast, are decentralized and can run on any connected network topology.

Contributions. We propose a distributed algorithm for solving (1). This algorithm is an extension of our prior work [24], where we solved a particular instance of (1), namely Basis Pursuit [25]. Here, we first show that the algorithm in [24] can be generalized to the problem class (1). Then we apply it to following problems: average consensus, two compressed sensing problems (different from the Basis Pursuit), and support vector machines. We also provide extensive simulations of several distributed algorithms solving (1). Our simulations show that the proposed algorithm outperforms, in terms of the number of used communications, any of the previous algorithms. One of the above problems, the average consensus, has been extensively studied due to its importance and simplicity. For example, many distributed algorithms, which cannot be trivially generalized to solve (1), have been proposed to solve it. Our algorithm, designed to solve the generic problem (1), can also solve the average consensus problem and, as our simulations show, it outperforms a state-of-the-art consensus algorithm [26], for many networks of interest.

Organization. The paper is organized as follows. In section II we derive the algorithm, by manipulating the problem and then applying ADMM. Section III is dedicated to related algorithms; in particular, we describe the competing algorithms and show how we compare them with ours. In section IV we take problems from several areas and see how they can be recast as (1). Additionally, we analyze the resulting algorithm for each one of these problems. Finally, in section V, we describe and show the results of our simulations. In Appendix A we provide a proof of convergence of the particular version of ADMM we use, making this paper self-contained.

II. Algorithm Derivation

In this section, we propose a distributed algorithm for solving (1). We begin by introducing some graph theory

concepts and notation. Next, we address the scenario of a bipartite network, where a proof of convergence is given and the exposition is simpler; then, we generalize the algorithm for arbitrary networks. In this case, although we do not provide a proof of convergence, the algorithm always converges in practice, as shown in section V.

Network notation. We represent a network as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, 2, \dots, P\}$ is the set of nodes and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the set of edges. The cardinality of these sets is represented respectively by P, the number of nodes, and E, the number of edges. An edge is represented by $\{i, j\} = \{j, i\}$, and $\{i, j\} \in \mathcal{E}$ means that nodes i and j can exchange data with each other. We define the neighborhood $\mathcal{N}_p$ of a node p as the set of nodes connected to node p, but excluding it; the cardinality of this set, $D_p := |\mathcal{N}_p|$, is the degree of node p.

Node coloring. Given a network $\mathcal{G}$, a (node) coloring scheme is an assignment of numbers, which we call colors, to the nodes of the network such that no adjacent nodes have the same color. The colors will be represented as a set $\mathcal{C} = \{1, 2, \dots, C\}$, where C is the total number of colors. For each color c, the set $\mathcal{C}_c \subset \mathcal{V}$ represents the nodes with color c, and C_c is the number of nodes with that color, i.e., $|\mathcal{C}_c| = C_c$. A network $\mathcal{G}$ can have several coloring schemes, and $\chi(\mathcal{G})$ denotes the minimum number of colors required to color it; it is called its chromatic number. Finding a coloring scheme that uses only $\chi(\mathcal{G})$ colors is NP-hard for $\chi(\mathcal{G}) > 2$. Node-coloring is very important in wireless networks, since it is used in some medium access (MAC) protocols; consequently, many distributed node-coloring algorithms have been proposed, e.g., [27], [28], [29], [30]. Henceforth, we will assume that the network $\mathcal{G}$ is given together with a coloring scheme $\mathcal{C}$.

Problem manipulations. We start by assuming the network $\mathcal{G}$ is bipartite or, equivalently, that $\chi(\mathcal{G}) = 2$, i.e., it can be colored with two colors. An example of a bipartite network is a grid. We start with this assumption for two reasons: the exposition is simpler, and we prove that our algorithm converges for this case. Without loss of generality, assume the nodes are ordered such that $\mathcal{C}_1 = \{1, 2, \dots, C_1\}$ and $\mathcal{C}_2 = \{C_1 + 1, C_1 + 2, \dots, C\}$.

A common technique to decouple problem (1) is to assign copies of the global variable x to each node and then constrain all copies to be equal. Denoting the copy held by node p with $x_p \in \mathbb{R}^n$, problem (1) is written equivalently as

$$\begin{array}{ll} \underset{\bar{x}=(x_1,\dots,x_P)}{\text{minimize}} & f_1(x_1) + f_2(x_2) + \cdots + f_P(x_P) \\ \text{subject to} & x_p \in X_p\,, \quad p = 1, \dots, P \\ & x_i = x_j\,, \quad \{i, j\} \in \mathcal{E}\,, \end{array} \tag{2}$$

where $\bar{x} = (x_1, \dots, x_P) \in (\mathbb{R}^n)^P$ is the optimization variable. Problem (2) is no longer coupled by the intersection of the sets X_p or by a common variable in all f_p's, as (1), but instead by the new equations $x_i = x_j$, for all pairs of edges in the network $\{i, j\} \in \mathcal{E}$. These equations enforce all copies to be equal since the network is connected, due to Assumption 3. Note that they can be written more compactly as $(B^\top \otimes I_n)\bar{x} = 0$, where $B \in \mathbb{R}^{P \times E}$ is the node arc-incidence matrix of the graph, I_n is the identity matrix in $\mathbb{R}^n$, and $\otimes$ is the Kronecker product. Each column of B is associated with an edge $\{i, j\} \in \mathcal{E}$ and has 1 and -1 in the ith and jth entry, respectively; the remaining entries are zeros. We can partition B into two matrices B_1 and B_2, where B_1 contains the first C_1 rows and B_2 contains the remaining rows. Therefore, $(B^\top \otimes I_n)\bar{x} = (B_1^\top \otimes I_n)\bar{x}_1 + (B_2^\top \otimes I_n)\bar{x}_2$, where $\bar{x}_1 = (x_1, \dots, x_{C_1}) \in (\mathbb{R}^n)^{C_1}$ and $\bar{x}_2 = (x_{C_1+1}, \dots, x_C) \in (\mathbb{R}^n)^{C_2}$. This enables rewriting (2) as

$$\begin{array}{ll} \underset{\bar{x}_1,\bar{x}_2}{\text{minimize}} & \sum_{p\in\mathcal{C}_1} f_p(x_p) + \sum_{p\in\mathcal{C}_2} f_p(x_p) \\ \text{subject to} & \bar{x}_1 \in \bar{X}_1\ ,\ \bar{x}_2 \in \bar{X}_2 \\ & (B_1^\top \otimes I_n)\bar{x}_1 + (B_2^\top \otimes I_n)\bar{x}_2 = 0\,, \end{array} \tag{3}$$

where $\bar{X}_i = \bigcap_{p\in\mathcal{C}_i} X_p$, for $i = 1, 2$. Problem (3) can be solved with ADMM, explained next.

ADMM. The alternating direction method of multipliers (ADMM) was proposed in the seventies by [14], [15]. Given two functions g_1 and g_2, two sets X_1 and X_2, and two matrices A_1 and A_2, ADMM solves

$$\begin{array}{ll} \underset{x_1,x_2}{\text{minimize}} & g_1(x_1) + g_2(x_2) \\ \text{subject to} & x_1 \in X_1\ ,\ x_2 \in X_2 \\ & A_1x_1 + A_2x_2 = 0\,. \end{array} \tag{4}$$

It uses the method of multipliers concatenated with an iteration of the Gauss-Seidel algorithm [11], i.e., it iterates on k:

$$x_1^{k+1} \in \arg\min_{x_1\in X_1} L_\rho(x_1, x_2^k; \lambda^k) \tag{5}$$

$$x_2^{k+1} \in \arg\min_{x_2\in X_2} L_\rho(x_1^{k+1}, x_2; \lambda^k) \tag{6}$$

$$\lambda^{k+1} = \lambda^k + \rho(A_1x_1^{k+1} + A_2x_2^{k+1})\,, \tag{7}$$

where

$$L_\rho(x_1, x_2; \lambda) := g_1(x_1) + g_2(x_2) + \lambda^\top(A_1x_1 + A_2x_2) + \frac{\rho}{2}\|A_1x_1 + A_2x_2\|^2 \tag{8}$$

is the augmented Lagrangian of (4), λ is the dual variable, and $\rho > 0$ is a predefined parameter. More information about ADMM, including variations, applications, and related algorithms can be found, for example, in [11], [23], [31]. The following theorem establishes its convergence.

Theorem 1 ([11], [23])**.** *Assume $g_i : \mathbb{R}^{n_i} \to \mathbb{R}$ is a convex function over $\mathbb{R}^{n_i}$, $X_i \subset \mathbb{R}^{n_i}$ is a closed convex set, and A_i is a full column-rank matrix, for $i = 1, 2$. Also, assume problem* (4) *is solvable. Then, the sequence $\{(x_1^k, x_2^k, \lambda^k)\}$ generated by* (5)-(7) *converges to $(x_1^\star, x_2^\star, \lambda^\star)$, where*

1) *$(x_1^\star, x_2^\star)$ solves* (4)
2) *$\lambda^\star$ solves the dual problem of* (4)*:*

$$\underset{\lambda}{\textit{minimize}}\ \ G_1(\lambda) + G_2(\lambda)\,, \tag{9}$$

where $G_i(\lambda) = \inf_{x_i\in X_i}(g_i(x_i) + \lambda^\top A_i x_i)$, $i = 1, 2$.

This theorem is more general than the versions in [23] and [11, Prop.4.2]. Namely, [23] only proves objective convergence, but since it does not assume A_1 and A_2 to be full column-rank, the sequence $\{(x_1^k, x_2^k, \lambda^k)\}$ might not even converge. On the other hand, [11] proves claims 1) and 2) only for the special case when one of the A_i's is the identity. Since we could not find a proof of this version of the theorem in

literature, we provide it in Appendix A. The proof is, however, very similar to the ones in [11], [23].

Applying ADMM. Clearly, (3) has the same format as (4): just make the following associations

$$g_i(x_i) = \sum_{p\in\mathcal{C}_i} f_p(x_p)\,, \quad X_i = \bar{X}_i\,, \quad A_i = B_i^\top \otimes I_n\,, \quad (10)$$

for $i = 1, 2$. The following theorem guarantees that, under Assumptions 1-3, problem (3) satisfies all the conditions of Theorem 1; thus, we can use ADMM to solve (3).

Theorem 2. *Let Assumptions 1-3 hold and consider problem* (3)*. Then, for $i = 1, 2$, the function $\sum_{p\in\mathcal{C}_i} f_p(x_p)$ is convex over $\mathbb{R}^n$, the set $\bar{X}_i$ is closed and convex, and the matrix $B_i^\top \otimes I_n$ has full column-rank. Furthermore, problem* (3) *is solvable.*

All conclusions of Theorem 2, except that $B_i^\top \otimes I_n$ has full column-rank, are derived straightforwardly Assumptions 1 and 2 and the equivalence between (3) and (1). To see how Assumption 3 implies that each $B_i^\top \otimes I_n$ has full column-rank for $i = 1, 2$, note that it is sufficient to prove that $B_i^\top$ has full column-rank. If we prove that $B_i B_i^\top$ has full rank, then the results follows because $\text{rank}(B_i B_i^\top) = \text{rank}(B_i^\top)$. Now note that $B_i B_i^\top$ is a diagonal matrix, where in the diagonal are the degrees of the nodes belonging to the subnetwork composed by the nodes in $\mathcal{C}_i$. Assumption 3 states that the network is connected and thus no node can have degree 0; consequently, $B_i B_i^\top$ has full rank.

Algorithm for bipartite networks. We now apply ADMM, i.e., (5)-(7) directly to (3). By analyzing the augmented Lagrangian (8), we will see that (5) yields C_1 optimization problems that can be solved in parallel. Similarly, (6) yields C_2 optimization problems that can be solved in parallel. In fact, developing the squared term in (8) we obtain for the x_1-minimization (5):

$$x_1^{k+1} \in \arg\min_{x_1\in X_1} g_1(x_1) + \bar{\eta}_1^\top x_1 + \frac{\rho}{2} x_1^\top (A_1^\top A_1) x_1\,, \quad (11)$$

where $\bar{\eta}_1 := A_1^\top \lambda^k + \rho(A_1^\top A_2) x_2^k$. Using (10), the quadratic term in (11) becomes $(\rho/2)\bar{x}_1^\top (B_1 B_1^\top \otimes I_n)\bar{x}_1$. As we had seen before, $B_1 B_1^\top$ is a diagonal matrix with the degree of node p, D_p, in the (pp)th entry. This quadratic term can then be rewritten as $(\rho/2)\sum_{p\in\mathcal{C}_1} D_p \|x_p\|^2$. Regarding the linear term,

$$\begin{aligned}\bar{\eta}_1^\top \bar{x}_1 &= \Big((B_1 \otimes I_n)\lambda^k + \rho(B_1 B_2^\top \otimes I_n)\bar{x}_2^k\Big)^\top \bar{x}_1 \\ &= \sum_{p\in\mathcal{C}_1}\Big(\sum_{j\in\mathcal{N}_p} \text{sign}(j-p)\lambda_{\{p,j\}}^k - \rho\sum_{j\in\mathcal{N}_p} x_j^k\Big)^\top x_p\,,\end{aligned} \quad (12)$$

where $\text{sign}(w) = 1$ if $w \geq 0$ and $\text{sign}(w) = -1$ otherwise. We decomposed the dual variable λ as $(\ldots, \lambda_{\{i,j\}}, \ldots)$, where $\lambda_{\{i,j\}} = \lambda_{\{j,i\}}$ is associated with the edge $\{i, j\} \in \mathcal{E}$. To see why (12) holds, note that the (ij)th entry of $B_1 B_2^\top$ is -1 if $\{i, j\} \in \mathcal{E}$ and 0 otherwise. In conclusion, using (10) on (11), we get C_1 optimization problems that can be solved in parallel:

$$x_p^{k+1} \in \arg\min_{x_p\in X_p} f_p(x_p) + ({\gamma_p^k}^\top - \rho\sum_{j\in\mathcal{N}_p} x_j^k)^\top x_p + \frac{\rho D_p}{2}\|x_p\|^2\,,$$

where $\gamma_p^k := \sum_{j\in\mathcal{N}_p} \text{sign}(j-p)\lambda_{\{p,j\}}^k$, for $p \in \mathcal{C}_1$, i.e., all nodes with color 1. A similar analysis for (6) yields the same problem for each node in $\mathcal{C}_2$, but using the neighbors' estimates at $k+1$.

Note that the optimization problem each node solves depends on the dual variable λ through γ_p. In fact, we can derive a formula for updating γ_p from (7): write (7) edgewise, $\lambda_{\{i,j\}}^{(k+1)} = \lambda_{\{i,j\}}^{(k)} + \rho\,\text{sign}(j-p)(x_i^{k+1} - x_j^{k+1})$, and insert this formula in the definition of γ_p^k; the result is $\gamma_p^{k+1} = \gamma_p^k + \rho\sum_{j\in\mathcal{N}_p}(x_p^{k+1} - x_j^k)$. In sum, we arrive at the following algorithm, named D-ADMM, after *Distributed*-ADMM.

Algorithm 1 D-ADMM for bipartite networks

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

$$x_p^{k+1} = \begin{array}{cc}\text{argmin} & f_p(x_p) + {v_p^k}^\top x_p + \frac{D_p\rho}{2}\|x_p\|^2 \\ \text{s.t.} & x_p \in X_p\end{array}$$

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} in v_p^k
7: **for all** $p \in \mathcal{V}$ [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

In Algorithm 1, there are two groups of nodes: $\mathcal{C}_1$ and $\mathcal{C}_2$. In each group, the nodes are not neighbors between themselves and operate at the same time. In particular, one node receives x_j^k or x_j^{k+1} from all its neighbors and then solves the optimization problem of step 3, finding its estimate x_p^{k+1}; it then broadcasts x_p^{k+1} to its neighbors. After having received all the estimates from its neighbors, node p can update γ_p as in step 7.

Algorithm 1 only applies to bipartite networks and its convergence is guaranteed by Theorems 1 and 2. Next, we generalize it for any type of network.

Algorithm for arbitrary networks. So far we assumed the network was bipartite, i.e., it could be colored with just two colors: $\chi(\mathcal{G}) = 2$. Now, suppose the network is colored with $C \geq 2$ colors. We derived problem (2) from problem (3) by partitioning the node-arc incidence matrix B into $\begin{bmatrix}B_1^\top & B_2^\top\end{bmatrix}^\top$, where B_1 and B_2 contain the rows corresponding to nodes in $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. For convenience, we had assumed the nodes were ordered such that $\mathcal{C}_1 = \{1, 2, \ldots, C_1\}$ and $\mathcal{C}_2 = \{C_1 + 1, C_1 + 2, \ldots, C\}$. Without loss of generality, assume a similar ordering for the nodes in the current case, i.e., $\mathcal{C}_1 = \{1, 2, \ldots, C_1\}$, $\mathcal{C}_2 = \{C_1{+}1, \ldots, C_1{+}C_2\}, \ldots, \mathcal{C}_C = \{\sum_{c=1}^{C-1} C_c{+}1, \ldots, C\}$. This

induces a natural partition of B as $\begin{bmatrix} B_1^\top & B_2^\top & \cdots & B_C^\top \end{bmatrix}^\top$. The counterpart of (3) for this case is then

$$\begin{array}{ll} \underset{\bar{x}_1,\ldots,\bar{x}_C}{\text{minimize}} & \sum_{c=1}^{C} \sum_{p\in\mathcal{C}_p} f_p(x_p) \\ \text{subject to} & \bar{x}_c \in \bar{X}_c\,, \quad c=1,\ldots,C \\ & \sum_{c=1}^{C} (B_c^\top \otimes I_n)\bar{x}_c = 0\,, \end{array} \tag{13}$$

where the variable is $(\bar{x}_1,\ldots,\bar{x}_C) \in (\mathbb{R}^n)^P$. To solve (13), we cannot apply ADMM (5)-(7) directly, but its generalization to solve

$$\begin{array}{ll} \underset{x_1,\ldots,x_C}{\text{minimize}} & g_1(x_1)+\cdots+g_C(x_C) \\ \text{subject to} & x_c \in X_c\,, \quad c=1,\ldots,C \\ & A_1x_1+\cdots+A_Cx_C = 0\,. \end{array}$$

Such a generalization is straightforward, but there are no convergence guarantees in the sense that there is no known proof of Theorem 1 for the resulting algorithm. Yet, experimental results, as the ones presented here, suggest that Theorem 1 might still hold.

The following algorithm is a straightforward generalization of Algorithm 1 to solve (13).

Algorithm 2 D-ADMM for general networks

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

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

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

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

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

In Algorithm 2 there are C groups of nodes, arranged by colors. As in the bipartite case, in each group, the nodes are not neighbors between themselves and operate at the same time. In practice, they cannot operate at the same time because there is no central coordination. As seen in the expression for v_p^k (step 3), one node, knowing the color of its neighbors, can operate only after having received the estimates x_k^{k+1} from the nodes with lower color. Excepting that, the algorithm is very similar to Algorithm 2.

As said before, although there is no convergence guarantee for Algorithm 2, it converges in practice for several kinds of problems, and we have never seen a nonconvergent example. We point out that, using a reasoning similar to that of the discussion after Theorem 2, each matrix $B_i^\top$ can be proved to be full column-rank.

III. Related Algorithms

We now review some algorithms for solving (1). We start with describing the performance measure we will use and its practical relevance.

Performance measure: communication steps. We divide the algorithms that solve (1) in two kinds: single-looped and double-looped. The single-looped algorithms have just one (nested) loop, and in each iteration, each node receives the neighbors' estimates, computes its on estimate, and broadcasts it to its neighbors. The double-looped algorithms have two nested loops and they are based on augmented Lagrangian methods (see (8)). The outer loop serves to update the dual variable of the augmented Lagrangian; the inner loop serves for each node to receive the neighbors' estimates, compute its own estimate, and broadcast it to the neighbors, as one iteration of a single-looped algorithm. The natural way of comparing all algorithms is then through a *communication step*, which will denote one iteration (for the single-looped algorithms) or one inner loop iteration (for the double-looped algorithms). We say that an algorithm is more efficient than another one if it uses less communication steps for the same solution accuracy.

Communication steps is a fair measure because it compares, not only similar computational operations at each node (as we will soon see), but also the amount of communications each node requires to achieve a solution with a given precision. Furthermore, one of the main concerns in designing algorithms and protocols for wireless networks is reducing the number of communications; the reason is because communicating is the slowest and most energy-consuming task [2], [32]. The same concern is shared in supercomputing environments [4]. In contrast with execution time, communication steps are application and implementation independent. For example, given a distributed algorithm, we get different execution times if we change any of the following: medium-access protocol, transmission medium, scheduling, platform for the nodes, or even the way the algorithm is coded. Communication steps are, on the other hand, invariant to any of these. Note that, given a network with E edges, if we multiply $2E$ by the number of communication steps, we obtain the total number of communications in the network.

We point out, however, that our algorithm, D-ADMM, is asynchronous, i.e., not all nodes perform the same tasks at the same time, in contrast with most of the algorithms presented next. More concretely, D-ADMM assumes a coloring for the nodes, and each node operates according to its color. Other algorithms do not rely on any coloring scheme, enabling them to be synchronous, i.e., all nodes can perform the same tasks at the same time.[1] Therefore, if one communication step in D-ADMM takes T time units, it would take T/C units in a synchronous algorithm, where C is the number of colors, in case the nodes were able to transmit messages at the same time.

Gradient/Subgradient method. The classical approach for solving (1) is to use a subgradient method combined with

[1] At our abstraction level, we are overlooking packet collision and other medium-access problems that may potentially destroy synchronism.

an averaging consensus algorithm. This approach was taken in [6], and consists of iterating in k

$$x_p^{k+1} = \Big[a_{pp}^k x_p^k + \sum_{j\in\mathcal{N}_p} a_{pj}^k x_j^k - \alpha^k g_p^k\Big]_{X_p}^+, \qquad (14)$$

for each node p. In (14), $[s]_S$ represents the projection of the point s onto the (convex) set S, x_p^k the estimate of node p at iteration k, g_p^k the gradient (or a subgradient, in case the objective is not differentiable) of f_p at the point $a_{pp}^k x_p^k + \sum_{j\in\mathcal{N}_p} a_{pj}^k x_j^k$, and a_{ij}^k represents the influence node j exerts on node i at iteration k. There are some conditions the weights a_{ij}^k have to satisfy [6], for example, for each k, the matrix with entries a_{ij}^k should be doubly stochastic. In each iteration, node p receives the estimates x_j^k from its neighbors, computes (14), and then broadcasts x_p^{k+1} to its neighbors. It is thus a (synchronous) single-looped algorithm. Although this algorithm is provably robust to noise, it is known to perform very slowly, requiring lots of iterations to converge.

ADMM-based algorithms. We now describe two algorithms that are based on ADMM, like ours. Both of them are based on different reformulations of (1), yielding two different algorithms. The first algorithm is [16] (it also appears in [17], [18], [19], [20]), described in Algorithm 3.

Algorithm 3 [16]

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

1: **repeat**
2: **for all** $p = 1, \dots, P$ [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

$$x_p^{k+1} = \begin{array}{ll} \text{argmin} & f_p(x_p) + {v_p^k}^\top x_p + \rho D_p \|x_p\|^2 \\ \text{s.t.} & x_p \in X_p \end{array}$$

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** $p = 1, \dots, P$ [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 derived from a different reformulation of (1), Algorithm 3 algorithm is very similar to D-ADMM. The three main differences are:

1) Algorithm 3 is synchronous, i.e., all the nodes can perform the same tasks at the same time;
2) while in Algorithm 3 node p uses x_p^k to construct the vector v_p^k (see step 3), D-ADMM does not;
3) Algorithm 3 is proven to converge for any connected network, and D-ADMM is only proven to converge in bipartite networks (although in practice it converges in any connected network).

We will see that D-ADMM requires almost always less communication steps to achieve a given accuracy than Algorithm 3, for several instances of (1).

Another ADMM-based algorithm is [21] (also appearing in [22], [20]). That algorithm, in contrast with D-ADMM and Algorithm 3, uses two communication steps per iteration. Since the number of iterations it takes to achieve a given accuracy in the average consensus problem is similar to Algorithm 3 [20], it takes the double of the communication steps.

Double-looped algorithms. Several double-looped algorithms were proposed to solve particular optimization problems, which as D-ADMM, can be generalized to solve (1). In particular, they can solve (1), by solving the following dual problem of (2):

$$\underset{\lambda}{\text{maximize}} \; L_\rho(\lambda)\,, \qquad (15)$$

where $L_\rho(\lambda)$ is the (augmented) dual function

$$L_\rho(\lambda) = \begin{array}{ll} \inf\limits_{\bar{x}} & \sum_{p=1}^P f_p(x_p) + \sum_{\{i,j\}\in\mathcal{E}} \phi^\rho_{\lambda_{\{i,j\}}}(x_i - x_j) \\ \text{s.t.} & x_p \in X_p\,, \quad p = 1, \dots, P\,, \end{array} \qquad (16)$$

and $\phi_v^\rho(\eta) := v^\top\eta + \frac{\rho}{2}\|\eta\|^2$. It can be proved that $L_\rho(\lambda)$ is differentiable, and thus (15) can be solved with a gradient method, or even with a fast (Nesterov) gradient method. The outer loop consists of applying such algorithm: it updates the dual variables $\lambda_{\{i,j\}}$. However, to compute the gradient of $L_\rho(\lambda)$, one needs to solve the optimization problem in (16); and this requires another iterative algorithm, consisting of the second nested loop. Several approaches have been used to solve that problem, e.g., nonlinear Gauss-Seidel methods [9], [10], Jacobi-type methods [8], and Nesterov methods [12].

Of all the algorithms described in this section, only gradient/subgradient method does not solve an optimization problem at each iteration. All the remaining algorithms solve the same problem as D-ADMM (step 4 of Algorithm 2), but possibly with different parameters. As illustrated in section V, in practice, our D-ADMM performs generally better than any of algorithms described in this subsection. The only true competitor is Algorithm 3.

IV. Applications

We now take some optimization problems from signal processing and see how they can be recast as (1). We also review some of the conventional methods to solve them in a distributed way. Then, in section V, we present simulation results of D-ADMM and other algorithms applied to these problems.

Consensus. Consensus is one of the most fundamental problems in networks. Given a network with P nodes, node p generates a number or measurement, say θ_p, and the goal is to compute the average $\theta^\star = (1/P)\sum_{p=1}^P \theta_p$ in every node. The classical approach to consensus [33], [26] is

$$x^{k+1} = Ax^k\,, \qquad x^0 = \theta\,, \qquad (17)$$

where the pth entry of $x^k = (x_1^k, \dots, x_P^k) \in \mathbb{R}^P$ contains the estimate of node p at time k and $\theta = (\theta_1, \dots, \theta_P) \in \mathbb{R}^P$ is the vector collecting the measurements. The matrix A reflects the topology of the network by having $a_{ij} = 0$ for $\{i,j\} \notin \mathcal{E}$. Thus, (17) is equivalent to $x_p^{k+1} = \sum_{j\in\mathcal{N}_p\cup\{p\}} a_{pj} x_j^k$, i.e., the estimate of node p at $k+1$ will be a weighted average of its estimate and the neighbors' estimates at k. There are some constraints on choosing the weights, for example,

$\sum_{j\in\mathcal{N}_p\cup\{p\}} a_{pj} = 1$. See [26] for a recent survey on consensus algorithms using the approach (17).

Another approach to consensus, recently adopted in [20], is to solve the optimization problem

$$\underset{x}{\text{minimize}} \ \sum_{p=1}^{P}(x-\theta_p)^2 \tag{18}$$

in a distributed way. The solution of (18) is $\theta^\star = (1/P)\sum_{p=1}^{P}\theta_p$. Clearly, (18) has the format of (1) by assigning $f_p(x) = (x-\theta_p)^2$, $X_p = \mathbb{R}$; thus, it can be solved by D-ADMM or any of the methods described in the last section. The work [20] applies Algorithm 3 for solving (18).

In all algorithms, node p has to solve at each iteration

$$\underset{x}{\text{minimize}} \ (x-\theta_p)^2 + v^\top x + cx^2\,,$$

which has the closed-form solution $x^\star = (2\theta_p - v)/(2(1+c))$.

Compressed sensing problems. Compressed sensing is a new paradigm for the acquisition of signals. For an introductory survey see, for example, [34]. In compressed sensing, a signal is compressed at the same time as it is acquired, in contrast to the classical acquire-then-compress paradigm. Although compressed sensing makes the acquisition process simpler, the reconstruction of a compressed signal is more complicated, since an optimization problem has to be solved. The goal of this optimization problem is to find the sparsest solution of a linear system $Ax = b$, where $A \in \mathbb{R}^{m\times n}$ is a matrix related with the acquisition process and $b \in \mathbb{R}^m$ is the acquired/compressed signal. If A satisfies some technical conditions [34], then the solution to the *Basis Pursuit* (BP) [25],

$$\begin{array}{ll}\underset{x}{\text{minimize}} & \|x\|_1 \\ \text{subject to} & Ax = b\,,\end{array} \tag{19}$$

recovers the sparsest solution of $Ax = b$. In (19), $\|x\|_1 = |x_1| + \cdots + |x_n|$ is the ℓ_1-norm of x. A variation of (19) is the LASSO [34]:

$$\begin{array}{ll}\underset{x}{\text{minimize}} & \|x\|_1 \\ \text{subject to} & \|Ax-b\| \le \sigma\,,\end{array} \tag{20}$$

where $\sigma > 0$ is a fixed parameter and $\|\cdot\|$ the ℓ_2-norm. Problem (20) is more appropriate when there is noise in the acquisition process and we have an upper bound σ on the norm of the noise. Closely related with (20), is

$$\underset{x}{\text{minimize}} \ \|Ax-b\|^2 + \beta\|x\|_1\,, \tag{21}$$

called *Basis Pursuit Denoising* (BPDN) [25]. The parameter $\beta > 0$ regulates the tradeoff between the sparsity of the signal and the weight of the noise.

Here, we are interested in solving distributed versions of (19)-(21). Namely, we assume the matrix A is partitioned into P blocks, either by rows or by columns, as depicted in Figure 2. We assign the pth block to the pth node of a network with P nodes. For applications of this problem see, for example, our previous work [24]. In [24], we solved BP with D-ADMM for both the row and the column partition. In this paper, we solve (20) for the column partition, and (21) for the row partition. The reverse cases, i.e., (20) for the row partition or (21) for the column partition cannot be trivially recast as (1).

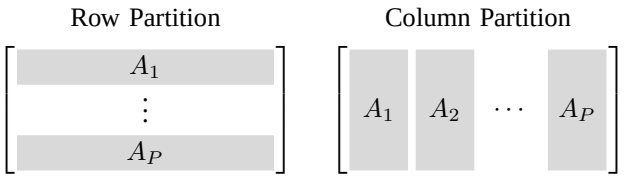


Figure 2. Row partition and column partition of A into P blocks. In the row partition a block is a set of rows; in the column partition a block is a set of columns.

LASSO: column partition. Assume the matrix A is partitioned by columns, $A = \begin{bmatrix}A_1 & \cdots & A_P\end{bmatrix}$, where the pth block of A is only known at node p. Assume vector b, parameter σ, and the number of nodes P are available at all nodes. This problem cannot be directly recast as (1); we will have to do it through duality. However, if we solve the "simple dual" of (20), we will not be able to recover solution to the primal problem afterwards. The reason is that the objective of (20) is not strictly convex. We thus start by regularizing (20) by adding a small term to its objective, making it strictly convex:

$$\begin{array}{ll}\underset{x}{\text{minimize}} & \|x\|_1 + \frac{\delta}{2}\|x\|^2 \\ \text{subject to} & \|Ax-b\| \le \sigma\,,\end{array} \tag{22}$$

where $\delta > 0$ is sufficiently small. This regularization is inspired by [35], which establishes some conditions for regularizations of this type to be exact. By exact, we mean that there exists $\bar{\delta} > 0$ such that the solution of (22) is always a solution of (20), for $\delta \le \bar{\delta}$. One of the conditions for exact regularization is that the objective is linear and the constraint set is the intersection of a linear system with a closed polyhedral cone. LASSO (20) can be recast as

$$\begin{array}{ll}\underset{x,t,u,v}{\text{minimize}} & 1_n^\top t \\ \text{subject to} & \|u\| \le v \\ & u = Ax - b\,,\ \ v = \sigma \\ & x \le t\,,\ \ -x \le t\,,\end{array} \tag{23}$$

where $1_n \in \mathbb{R}^n$ is the vector of ones, and (x,t,u,v) the variable. Although the objective of (23) is linear and its constraint set is the intersection of a linear system with a closed convex cone $\mathcal{K} := \{(u,v) : \|u\| \le v\}$, there is not a proof of exact regularization for (23), because $\mathcal{K}$ is not polyhedral. However, experimental results in [35] suggest that exact regularization might occur for non-polyhedral cones. We will also see in the results of our simulations that approximating (20) with (22) can still yield very accurate solutions.

Before computing the dual of (22), we first introduce a new variable $y \in \mathbb{R}^m$ and make the column partition explicit:

$$\begin{array}{ll}\underset{x,y}{\text{minimize}} & \sum_{p=1}^{P}(\|x_p\|_1 + \frac{\delta}{2}\|x_p\|^2) \\ \text{subject to} & \|y\| \le \sigma \\ & y = \sum_{p=1}^{P} A_p x_p - b\,.\end{array} \tag{24}$$

The dual problem of (24), by dualizing the last constraint only,

can be shown to be equivalent to

$$\underset{\lambda}{\text{minimize}} \ \sum_{p=1}^{P} g_p(\lambda)\,,$$

where $g_p(\lambda) := \frac{1}{P}(b^\top\lambda+\sigma\|\lambda\|)-\inf_{x_p}(\|x_p\|_1+(A_p^\top\lambda)^\top x_p+\frac{\delta}{2}\|x_p\|^2)$ is the function associated to node p. Each function g_p is convex because it is the pointwise supremum of convex functions [5, Prop.1.2.4]. The problem each node has to solve in each iteration is addressed in Appendix B.

We point out that problem (22) can be ill-conditioned due to the small value that has to be used for δ, if we want to get a good approximation of the original problem. In particular, the problem solved at each node can be very time-consuming, since it may require lots of internal iterations. We used $\delta = 10^{-3}$ in our simulations. While such value for δ required a considerable amount of iterations for solving the internal problem of each node (we set 500 as the number of maximum iterations), it also allowed to get solutions of the original problem (20) with relative errors of 0.1%, using few communications. BPDN, addressed next, handles the same type of problems as LASSO, but its distributed implementation does not require solving ill-conditioned problems.

BPDN: row partition. In BPDN (21), we assume that both the matrix A and the vector b are partitioned by rows: $A = \begin{bmatrix}A_1^\top & \cdots & A_P^\top\end{bmatrix}^\top$, $b = \begin{bmatrix}b_1^\top & \cdots & b_P^\top\end{bmatrix}^\top$. Therefore, (21) can be readily rewritten as

$$\underset{x}{\text{minimize}} \ \sum_{p=1}^{P}\Big(\|A_p x - b_p\|^2 + \frac{\beta}{P}\|x\|_1\Big)\,. \tag{25}$$

The identification between (1) and (25) is immediate: set $f_p(x) := \|A_p x - b_p\|^2 + \frac{\beta}{P}\|x\|_1$. The problem that has to be solved at node p,

$$\underset{x}{\text{minimize}} \ \|A_p x - b_p\|^2 + \frac{\beta}{P}\|x\|_1 + v^\top x + c\|x\|^2\,, \tag{26}$$

can be solved, for example, with the GPSR solver [36].

Distributed support vector machines. In a *Support Vector Machine* (SVM) [37, Ch.7], we are given data in a matrix $A \in \mathbb{R}^{m\times n}$, where each row of A represents a data point in $\mathbb{R}^n$. Each data point can be classified as belonging to group $\mathcal{X}$ or to group $\mathcal{Y}$. If data point a_i belongs to $\mathcal{X}$ (resp. $\mathcal{Y}$), we set the ith entry of a vector $d \in \mathbb{R}^m$ to $+1$ (resp. -1). The goal is to find the parameters $(s, r) \in \mathbb{R}^n \times \mathbb{R}$ of a linear separator $z(a) = s^\top a + r$ such that $z(a) > 0$ if $a \in \mathcal{X}$, and $z(a) < 0$ if $a \in \mathcal{Y}$. Once the pair (s, r) is found, we can attempt to determine if a new point a belongs to $\mathcal{X}$ or $\mathcal{Y}$ just by analyzing the sign of $z(a)$. There are several ways of finding (s, r); one of them is solving the quadratic program

$$\begin{array}{ll}\underset{s,r}{\text{minimize}} & \|s\|^2 \\ \text{subject to} & D(As - 1_m r) \geq 1_m\,,\end{array} \tag{27}$$

where $D = \text{Diag}(d_1, \ldots, d_m)$, and 1_m is the vector of ones in $\mathbb{R}^m$. Many times, the data is distributed among several entities, and processing it in a distributed way is prohibitive for complexity or privacy reasons. Here, we assume the model of [38], [19]: there are P entities, and each entity p has m_p data points, or equivalently, a block $A_p \in \mathbb{R}^{m_p\times n}$ of rows of A, and their corresponding classification, i.e., the pth diagonal block $D_p \in \mathbb{R}^{m_p\times m_p}$ of D, such that $m_1+\cdots+m_P = m$. This enables us to rewrite (27) as

$$\begin{array}{ll}\underset{s,r}{\text{minimize}} & \sum_{p=1}^{P}\|s\|^2 \\ \text{subject to} & D_p(A_p s - 1_{m_p} r) \geq 1_{m_p}\,, \quad p = 1, \ldots, P\,.\end{array} \tag{28}$$

Problem (28) has the format of (1): just set $f_p(s, r) = \|s\|^2$ and $X_p = \{(s, r) \in \mathbb{R}^n \times \mathbb{R} : D_p(A_p s - 1_{m_p} r) \geq 1_{m_p}\}$. The problem each node has to solve at each iteration is a quadratic program, which we solve with Matlab's `quadprog` function. Algorithm 3 was applied in [19] to solve a version of (28) that takes into account possible violations of the linear separability of the data. Here, for simplicity, we just consider (28) and assume the data can be separated linearly.

V. Experimental Results

In this section, we show some simulation results of D-ADMM, and the algorithms from section III, applied to the problems just described. Of these applications, consensus is the one that assesses the algorithms the best, since neither the problem solution, nor the problem each node solves at each iteration, requires iterative algorithms or any kind of solver: they have closed-form solutions. We will, therefore, give more emphasis to consensus than to the other applications. We start with describing the network models.

Network models. We generated networks according to the models described in Table I. All networks were generated using the igraph package in R [43], [44]. Table II shows the 56 different networks we generated: given a model and fixed its parameters, we generated 8 networks with different number of nodes, ranging from 10 to 2000; we used 7 different combinations of models and parameters. All networks, except Lattice networks, were generated randomly, and we had to guarantee that all networks were connected (cf. Assumption 3). So, every time we would get a non-connected network, we would generate another network with slightly different parameters, changed in the direction to make the next network connected with greater probability. Therefore, there were some exceptions to the parameters described in Table II. For example, network number 1 with 10 nodes was generated with $p = 0.27$, instead of with $p = 0.25$, or network number 6 with 10 nodes was generated with $d = 0.36$, instead of with $d = 0.2$.

Table II also shows the average node degree and the number of colors used for each network. The average node degree can be as low as 3, for example in the Lattice network with 50 nodes, or as high as 1499, in the Erdős-Rényi network with 2000 nodes. The number of colors each network was colored with is also varied. Note that only the Lattice networks were colored with 2 colors, and thus these are the only networks for which D-ADMM is proven to converge.

We use all networks of Table II for the consensus problem, since we study that problem in more detail. To illustrate the algorithms for the remaining applications, we will only use the networks with 10 and 50 nodes.

Table I
NETWORK MODELS.

Name	Parameters	Description
Erdős-Rényi [39]	p	Every pair of nodes $\{i, j\}$ is connected or not with probability p
Watts-Strogatz [40]	(n, p)	First, it creates a lattice where every node is connected to n nodes; then, it rewires every link with probability p. If link $\{i, j\}$ is to be rewired, it removes the link, and connects node i or node j (chosen with equal probability) to another node in the network, chosen uniformly.
Barabasi-Albert [41]	——	It starts with one node. At each step, one node is added to the network by connecting it to 2 existing nodes: the probability to connect it to node p is proportional to D_p.
Geometric [42]	d	It drops P points, corresponding to the nodes of the network, randomly in a $[0, 1]^2$ square; then, it connects nodes whose (Euclidean) distance is less than d.
Lattice	——	Creates a lattice of dimensions $m \times n$; m and n are chosen to make the lattice as square as possible.

Table II
NETWORK PARAMETERS, AVERAGE DEGREE, AND NUMBER OF COLORS.

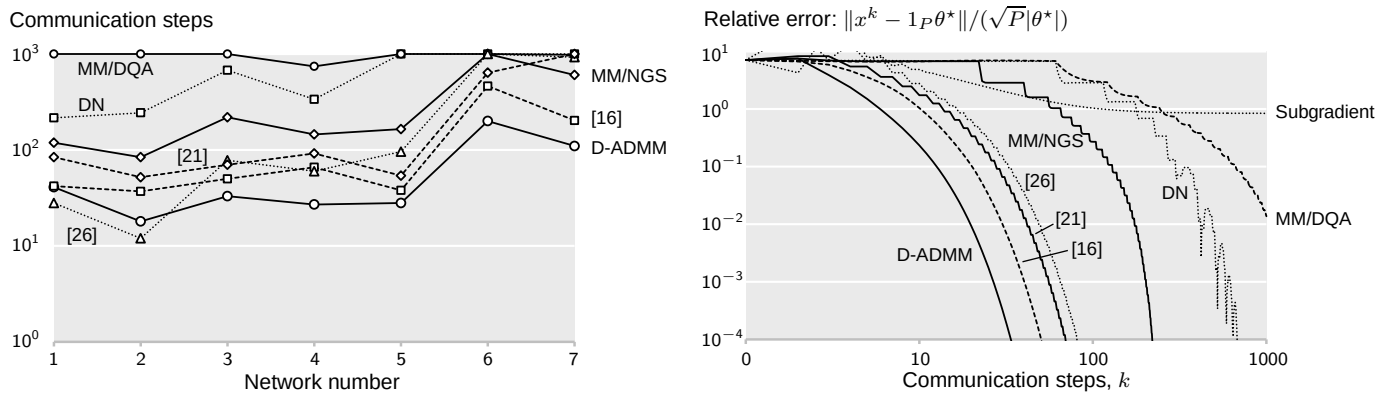
Number	Model	Parameters	Average degree (top), Number of colors (bottom)							
			Number of nodes							
			10	50	100	200	500	700	1000	2000
1	Erdős-Rényi	0.25	3 3	12 7	26 12	49 18	125 37	174 47	250 62	499 109
2	Erdős-Rényi	0.75	6 5	37 19	75 32	150 53	375 115	523 155	748 213	1499 380
3	Watts-Strogatz	$(2, 0.8)$	3 4	5 4	6 5	8 5	8 6	6 5	6 5	10 7
4	Watts-Strogatz	$(4, 0.6)$	5 4	7 5	8 6	8 6	8 6	10 7	8 6	10 7
5	Barabasi-Albert	——	3 3	4 3	4 4	4 4	4 4	4 4	4 4	4 4
6	Geometric	0.2	3 4	5 8	10 11	20 17	52 35	72 46	105 64	209 120
7	Lattice	——	3 2	3 2	4 2	4 2	4 2	4 2	4 2	4 2

Choosing ρ. In section III we reviewed the distributed algorithms in literature that can solve (1). All these algorithms, including D-ADMM, but excluding the gradient/subgradient method, are based on augmented Lagrangian duality. This means that, for each algorithm, we have to choose the parameter ρ of the augmented Lagrangian (see, e.g., (8)). We are unaware of any scheme that chooses this parameter in an optimal way before the execution of the algorithm. There are, however, schemes for updating it during the execution of the algorithm [23], [45], but it is not straightforward to implement them in a distributed scenario. We will thus adopt the following scheme for choosing ρ: given a fixed network, we execute several times all the algorithms that depend on ρ, each for a different ρ, chosen from the set $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}\}$; then, we pick the ρ that leads to the best result, i.e., the least number of communication steps. This scheme was adopted in all experiments.

Consensus. We now present simulation results for the consensus problem. The data was generated the following way: node p generates, independently of the other nodes, one number θ_p from a Gaussian distribution with mean 10 and standard deviation 100. We chose such a large standard deviation to make sure that the nodes' numbers differed significantly.

Figure 3 shows the results of the simulations of D-ADMM, the algorithms described in section III, and the additional algorithm from [26], which uses the approach (17). Note that [26] only solves the consensus problem and cannot be trivially generalized to solve (1). Figure 3(a) depicts the number of communication steps, which is equal to the number of iterations for the single-looped algorithms and to the number of inner iterations for the double-looped algorithms, for all the networks with 50 nodes. All algorithms stopped after reaching a relative error of $10^{-2}\%$, i.e., $\|x^k - 1_P\theta^\star\|/(\sqrt{P}|\theta^\star|) \leq 10^{-4}$, or after reaching the maximum number of 1000 communication steps. The optimal solution $\theta^\star$ had been computed before. It can been seen from the plot of Figure 3(a) that D-ADMM is the algorithm requiring the least communication steps to converge, i.e., to achieve a precision of $10^{-2}\%$, for all networks except the first two, where [26] was better. Figure 3(b) shows the error evolution along the communication steps for network 3 (cf. with point 3 of Figure 3(a)). It can be observed that D-ADMM required always the least number of communication steps to reach any accuracy between 10% and $10^{-2}\%$.

We only consider the three best algorithms (D-ADMM, [16], and [26]) for a more thorough analysis, in Figure 4; here, we simulate these algorithms in all the networks of Table II. In each of the plots of Figure 4 it is shown the



(a) Networks with 50 nodes.

(b) Error along iterations for network number 3.

Figure 3. Comparison between all algorithms for the consensus problem: (a) number of communication steps for each network model (see Table II); (b) error along iterations for network 3.

number of communication steps required to achieve $10^{-2}\%$ of accuracy as a function of the number of nodes. It can be seen that D-ADMM always required less communication steps than [16], except for the smallest network in almost all scenarios. D-ADMM also required less communication steps than [26], except for the Erdős-Rényi networks (Figures 4(a) and 4(b)). [26] was the only algorithm achieving the maximum number of 1000 iterations, which occurred for the Geometric and Lattice networks (Figures 4(f) and 4(g)). We also note a curious fact from these plots: the number of communication steps is almost invariant to the size of the networks (for the considered range).

We thus conclude that D-ADMM is a competing algorithm for solving the consensus problem, since it requires, in general, less communication steps to converge than other state-of-the-art algorithm, and it has approximately the same behavior for very different networks.

BPDN. We also executed some experiments comparing D-ADMM and [16] (Algorithm 3), the two best algorithms, solving BPDN (21). Figure 5 shows the results of such experiments, where we only used the networks with 50 nodes. For generating data, we used the Sparco toolbox [46], namely problems with ID's 7 and 902. Problem with ID 7 contains a 600×2560 matrix A with Gaussian entries, and problem with ID 902 contains a 200×1000 partial DCT matrix. In both problems, we added a small amount of noise to the provided vector b, in order to make the application of (21) more realistic. The trade-off parameter β was set to 1 for problem ID 7 and to 0.3 for problem ID 902. We ran the algorithms until they reached a relative error of $10^{-2}\%$ or the maximum number of iterations 1000, only reached by [16] in network 6. At each iteration, each node solved problem (26) with GPSR [36].

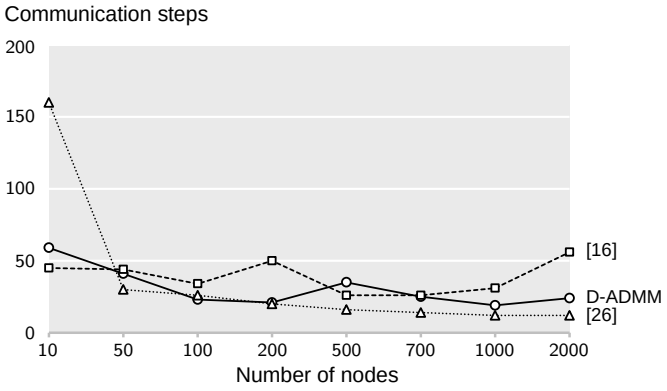
The results for problems with ID's 7 and 902 are shown respectively in Figures 5(a) and 5(b). These show the number of communication steps for each network model. Again, each point in the plots is the best result for $\rho \in \{10^{-4}, 10^{-3}, \ldots, 10^{2}\}$. It can be observed that D-ADMM always required less communication steps than [16]; however, there were some cases where both algorithms required almost the same communication steps, namely, for networks 2 and 3 of Figure 5(a), and for networks 2 and 4 of Figure 5(b). The similarity of the shape of the curves of Figures 5(a) and 5(b) indicates that changing the network models causes more variability in the difficulty of the problem than the problem data.

LASSO. The results of our experiments for LASSO (20) are shown in Figure 6. Again, we just compared the best two algorithms: D-ADMM and [16]. For the matrix A and vector b, we used the same data as in BPDN: problems 7 and 902 from the Sparco toolbox. The bound for the noise was set to $\sigma = 0.5$ for problem ID 7, and to $\sigma = 0.1$ for problem ID 902. In all experiments, which were executed for the networks with $P = 10$ nodes, the regularizing parameter in (22) was always $\delta = 10^{-3}$. With such a choice, both algorithms were able to achieve the relative error of $10^{-1}\%$ in all instances (the maximum number of iterations was never achieved). Note that the error was measured with respect to the solution of the original problem (20), which was obtained beforehand using the algorithm SPGL1 [47].

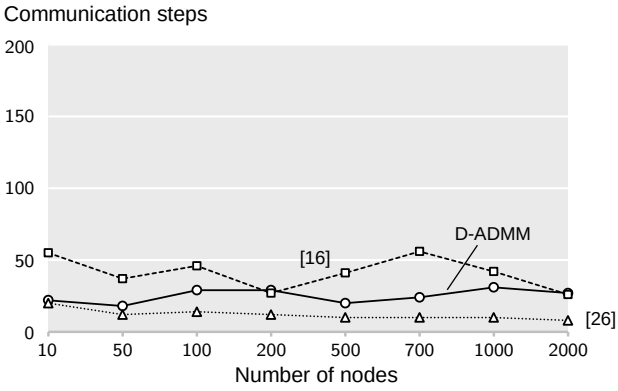
In Figure 6, we can see that, in spite of solving a slightly ill-conditioned problem, D-ADMM and [16] required considerably few communication steps to converge. Also note that the behavior of the algorithms is approximately uniform over all the types of networks. In contrast with BPDN, it was changing the data what produced different curves: the curves for ID 902 are on top of the curves for ID 7, which reveals that the former was harder to solve. For each problem, D-ADMM always took less communications steps than [16] to converge.

SVM. Quite surprisingly, the problem that required the largest number of communication steps was SVM. For this problem, we used data from the UCI machine learning repository [48], namely the Iris dataset, which contains two clusters of points that are linearly separable. Each point has $n = 4$ features and there are $m = 100$ total points. We first solved (27) with the Matlab function `quadprog`. Next, we executed D-ADMM and [16], which stopped after achieving a relative error of $10^{-2}\%$ or 10^4 communication steps.

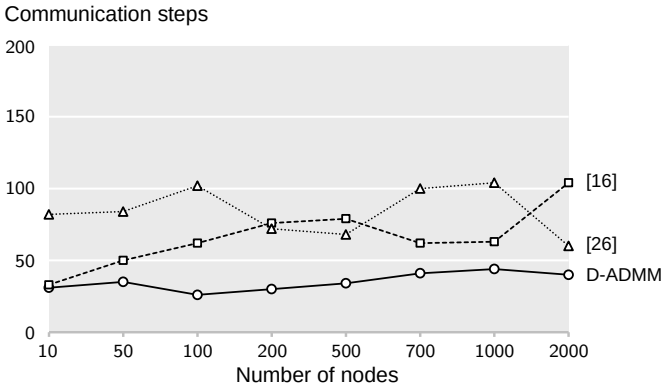
Figure 7 shows the results for the networks with 50 nodes. Again, D-ADMM required always less communication steps than [16] to converge; however, both algorithms required an



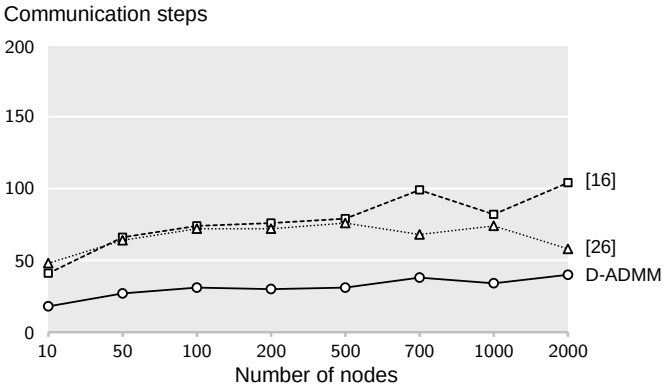
(a) Network 1: Erdős-Rényi, $p = 0.25$



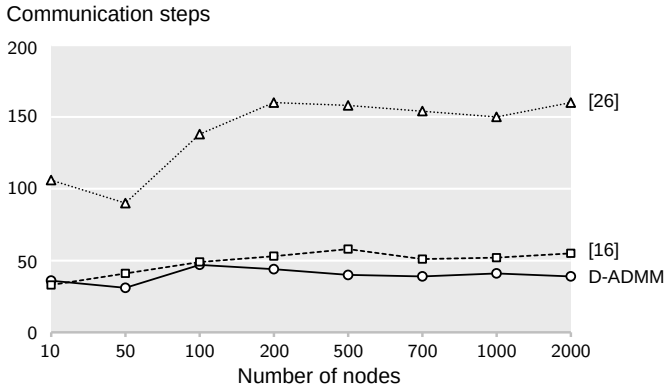
(b) Network 2: Erdős-Rényi, $p = 0.75$



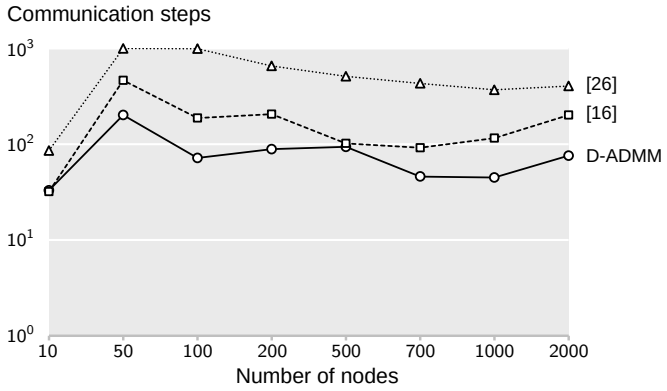
(c) Network 3: Watts-Strogatz, $(n, p) = (2, 0.8)$



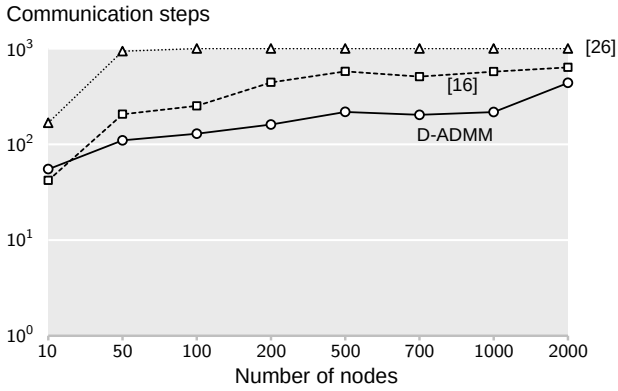
(d) Network 4: Watts-Strogatz, $(n, p) = (4, 0.6)$



(e) Network 5: Barabasi

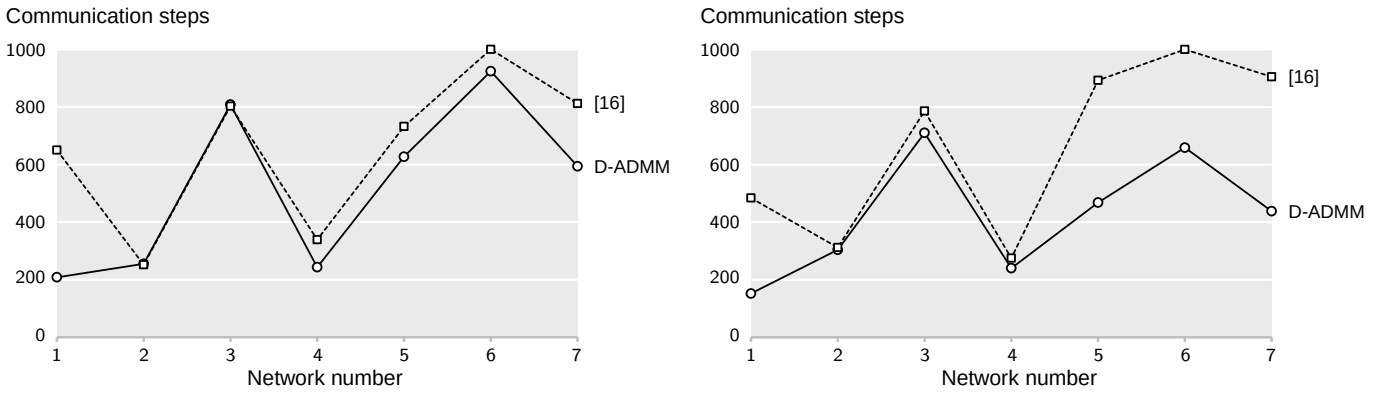


(f) Network 6: Geometric, $d = 0.2$



(g) Network 7: Lattice

Figure 4. Number of communication steps to achieve a $10^{-2}\%$ precision as a function of the number of nodes, for the consensus problem. The networks were generated according to Table II, and the algorithms compared are D-ADMM (Algorithm 2), [16] (Algorithm 3), and [26]. Note that (f) and (g) have a scale for the vertical axis different from the other plots.



(a) ID 7 (b) ID 902

Figure 5. Results for BPDN: communication steps to achieve $10^{-2}\%$ of accuracy in the networks with 50 nodes. The dataset is from the Sparco toolbox: problems (a) ID 7, and (b) ID 902.

amount of communications one order of magnitude larger than all the previous applications. And this happened with a relatively small problem. This might indicate that problem (27) is inherently difficult to solve in a distributed way.

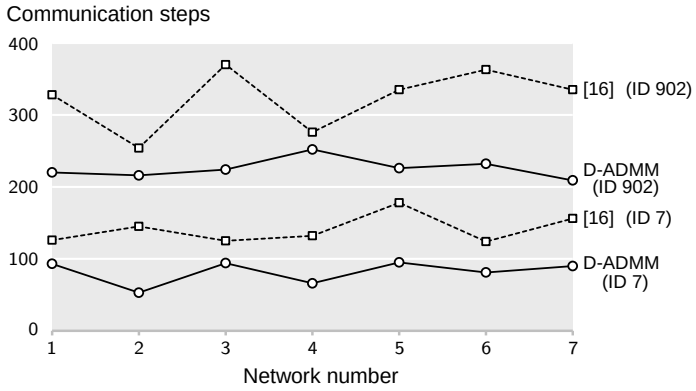


Figure 6. Results for LASSO: communication steps to achieve 0.1% of accuracy in the networks with 10 nodes.

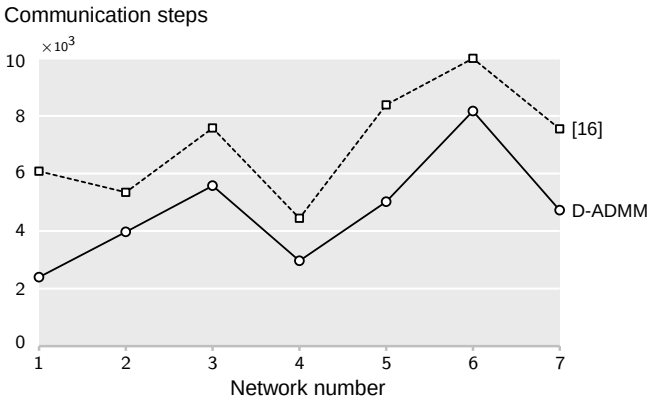


Figure 7. Results for SVM: communication steps to achieve $10^{-2}\%$ of accuracy in the networks of 50 nodes.

VI. Conclusions

We proposed an algorithm for solving separable problems in a distributed way. This means that, given a connected network, each node has a cost function and a constraint set associated. While keeping their costs and constraints sets private all the time, all nodes cooperate in order to solve a global optimization problem, which minimizes the sum of all the nodes' costs, and constrains the solution to be in the intersection of all sets. We showed how several problems from signal processing and control can be solved with the proposed algorithm, named D-ADMM, and presented extensive simulation results. These results show that D-ADMM requires almost always less communications to converge than any other state-of-the-art distributed optimization algorithm.

References

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

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

[3] J. Kurose and K. Ross, *Computer networking: A top-down approach featuring the internet*, Addison Wesley, 3rd edition, 2005.

[4] S. Graham, M. Snir, and C. Patterson, *Getting up to speed: the future of supercomputing*, National Academies Press, 3rd edition, 2005.

[5] D. Bertsekas, A. Nedić, and A. Ozdaglar, *Convex Analysis and Optimization*, Athena Scientific, 2003.

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

[7] A. Nedić and Ozdaglar, *Convex Optimization in Signal Processing and Communications*, chapter Cooperative distributed multi-agent optimization, Cambridge University Press, 2010.

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

[9] 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.

[10] D. Jakovetic, J. Xavier, and J. Moura, "Cooperative convex optimization in networked systems: Augmented lagrangian algorithms with directed gossip communication," *IEEE Trans. Sig. Proc.*, vol. 59, no. 8, 2011.

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

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

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

[14] R. Glowinski and A. Marrocco, "Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité, d'une classe de problèmes de dirichelet non linéaires," *Revue Française d'Automatique, Informatique, et Recherche Opérationelle*, vol. 9, no. 2, pp. 41–76, 1975.

[15] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximations," *Computers and Mathematics with Applications*, vol. 2, no. 1, pp. 17–40, 1976.

[16] H. Zhu, G. Giannakis, and A. Cano, "Distributed in-network channel decoding," *IEEE Trans. Sig. Proc.*, vol. 57, no. 10, 2009.

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

[18] S. Kim, E. Dall'Anese, and G. Giannakis, "Cooperative spectrum sensing for cognitive radios using kriged kalman filtering," *IEEE J. Sel. Topics Sig. Proc.*, vol. 5, no. 1, 2011.

[19] P. Forero, A. Cano, and G. Giannakis, "Consensus-based distributed support vector machines," *J. Machine Learning Research*, vol. 11, pp. 1663–1707, 2010.

[20] T. Erseghe, D. Zennaro, E. Dall'Anese, and L. Vangelista, "Fast consensus by the alternating direction multipliers method," *IEEE Trans. Sig. Proc.*, vol. 59, no. 11, 2011.

[21] I. Schizas, A. Ribeiro, and G. Giannakis, "Consensus in *ad hoc* wsns with noisy links - part i: Distributed estimation of deterministic signals," *IEEE Trans. Sig. Proc.*, vol. 56, no. 1, pp. 350–364, 2008.

[22] I. Schizas, G. Giannakis, S. Roumeliotis, and A. and Ribeiro, "Consensus in *ad hoc* wsns with noisy links - part ii: Distributed estimation and smoothing of random signals," *IEEE Trans. Sig. Proc.*, vol. 56, no. 4, pp. 1650–1666, 2008.

[23] 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.

[24] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "Distributed basis pursuit," to appear in Trans. Sig. Proc., 2012.

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

[26] A. Olshevsky and J. Tsitsiklis, "Convergence speed in distributed consensus and averaging," *SIAM Review*, vol. 53, no. 4, 2011.

[27] 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.

[28] 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.

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

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

[31] P. Combettes and J. Pesquet, *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, chapter Proximal splitting methods in signal processing, Springer-Verlag, 2010.

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

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

[34] E. Candès and M. Wakin, "An introduction to compressive sampling," *IEEE Sig. Proc. Mag.*, vol. 25, no. 2, pp. 21–30, 2008.

[35] M. Friedlander and P. Tseng, "Exact regulatization of convex programs," *SIAM J. Optim.*, vol. 18, no. 4, 2007.

[36] M. Figueiredo, R. Nowak, and S. Wright, "Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems," *IEEE J. Sel. Top. Sig. Proc.: Special Issue on Convex Optimization Methods for Signal Processing*, vol. 1, no. 4, 2007.

[37] C. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[38] O. Mangasarian and E. Wild, "Privacy-preserving classification of horizontally partitioned data via random kernels," Tech. Rep., Data Mining Institute, 07-03, 2007.

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

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

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

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

[43] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, 2006.

[44] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, 2008.

[45] 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.

[46] 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.

[47] 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.

[48] A. Frank and A. Asuncion, *UCI Machine Learning Repository*, University of California, School of Information and Computer Science, 2010.

APPENDIX A
PROOF OF THEOREM 1

First, note that the assumptions imply that strong duality holds for (4). This follows directly from [5, Prop.6.4.2] and it means that the optimal cost of (4) and of (9) have the same (finite) value, which will be denoted by $p^\star$. It also means that (9) is solvable. Let then $(x_1^\star, x_2^\star, \lambda^\star)$ denote any primal-dual solution. We will use the notation $p^k = g_1(x_1^k) + g_2(x_2^k)$, $r^k = A_1 x_1^k + A_2 x_2^k$. Using only strong duality and our assumptions on X_1 and X_2, we can prove that

$$p^\star - p^{k+1} \leq \lambda^{\star\top} r^{k+1} \tag{29}$$

$$p^{k+1} - p^\star \leq -(\lambda^{k+1})^\top r^{k+1} - \rho(A_2(x_2^{k+1} - x_2^k))^\top (A_2(x_2^{k+1} - x_2^\star) - r^{k+1}) \tag{30}$$

$$V^{k+1} \leq V^k - \rho\|r^{k+1}\|^2 - \rho\|A_2(x_2^{k+1} - x_2^k)\|^2\,, \tag{31}$$

where V^k is the Lyapunov function

$$V^k := \frac{1}{\rho}\|\lambda^k - \lambda^\star\|^2 + \rho\|A_2(x_2^k - x_2^\star)\|^2\,.$$

This is done in [23] and we omit the proof here. In the course of that proof, the following fact is used: if ϕ and Ψ are convex functions and Ψ is continuously differentiable, then $x^\star \in \arg\min_x\{\phi(x) + \Psi(x) : x \in X\}$ implies $x^\star \in \arg\min_x\{\phi(x) + \nabla\Psi(x^\star)^\top(x - x^\star) : x \in X\}$, for any closed convex set X. Using $\lambda^k = \lambda^{k-1} + \rho r^k$ and applying the previous fact to (5) and (6), we get, respectively,

$$x_1^k \in \arg\min_{x_1 \in X_1} g_1(x_1) + (\lambda^k - \rho A_2(x_2^k - x_2^{k-1}))^\top A_1 x_1 \tag{32}$$

$$x_2^k \in \arg\min_{x_2 \in X_2} g_2(x_2) + \lambda^{k\top} A_2 x_2\,. \tag{33}$$

Limit points of $\{(x_1^k, x_2^k)\}$ are primal optimal. From (31), we see that $0 \leq V^k \leq V^0$ for any k, and thus $\{\lambda^k\}$ and $\{A_2 x_2^k\}$ are bounded, showing that they have limit points. Since A_2 has full column-rank, $\{x_2^k\}$ is also bounded and thus has limit points. To show existence of limit points of $\{x_1^k\}$, we first have to show that $r^k \to 0$. Iterating (31) we get

$$\rho\sum_{k=0}^{\infty}(\|r^{k+1}\|^2 + \|A_2(x_2^{k+1} - x_2^k)\|^2) \leq V^0\,,$$

from which we conclude $r^k \to 0$ and $A_2(x_2^{k+1} - x_2^k) \to 0$ as $k \to 0$. Now, $r^k = A_1 x_1^k + A_2 x_2^k \to 0$, together with the boundedness of $\{A_2 x_2^k\}$, shows that $\{A_1 x_1^k\}$ is bounded and thus also $\{x_1^k\}$, because A_1 has full column-rank. Therefore, $\{x_1^k\}$ has limit points.

We showed the existence of limit points of $\{(x_1^k, x_2^k)\}$; now, we show their optimality. In fact, taking $k \to \infty$ in (29) and (30), using the boundedness of $\{A_2 x_2^k\}$ and $\{\lambda^k\}$, and

the facts that $r^k \to 0$ and $A_2(x_2^{k+1} - x_2^k) \to 0$, we conclude that $p^k = g_1(x_1^k) + g_2(x_2^k) \to p^\star$.

Limit points of $\{\boldsymbol{\lambda^k}\}$ are dual optimal. We had seen that $\{\lambda^k\}$ has limit points because it is bounded. Let $\bar{\lambda}$ be such limit point, and let $\mathcal{K}$ be the set of indices such that $\{\lambda^k\}_{k\in\mathcal{K}} \to \bar{\lambda}$ and also that $\{(x_1^k, x_2^k)\}_{k\in\mathcal{K}} \to (\bar{x}_1, \bar{x}_2)$, where $(\bar{x}_1, \bar{x}_2)$ is a limit point of $\{(x_1^k, x_2^k)\}$. Now, define $\hat{\lambda}^k = \lambda^k - \rho A_2(x_2^k - x_2^{k-1})$. Since $A_2(x_2^k - x_2^{k-1}) \to 0$, we have $\lim_{k\to\infty}\{\hat{\lambda}^k\}_{k\in\mathcal{K}} = \lim_{k\to\infty}\{\lambda^k\}_{k\in\mathcal{K}} = \bar{\lambda}$. Now, using the definition of G_1 and G_2, and (32)-(33), we see that

$$\begin{aligned}
G_1(\hat{\lambda}^k) &= \inf_{x_1\in X_1} g_1(x_1) + (\hat{\lambda}^k)^\top A_1 x_1 \\
&= g_1(x_1^k) + (\hat{\lambda}^k)^\top A_1 x_1^k && (34)\\
&\le g_1(x_1) + (\hat{\lambda}^k)^\top A_1 x_1\,, \qquad \forall_{x_1\in X_1}\,, && (35)\\
G_2(\lambda^k) &= \inf_{x_2\in X_2} g_2(x_2) + (\lambda^k)^\top A_2 x_2 \\
&= g_2(x_2^k) + (\lambda^k)^\top A_2 x_2^k && (36)\\
&\le g_2(x_2) + (\lambda^k)^\top A_2 x_2\,, \qquad \forall_{x_2\in X_2}\,. && (37)
\end{aligned}$$

Adding (34) and (36) and taking the limit $k \to +\infty$ ($k \in \mathcal{K}$),

$$\begin{aligned}
\lim_{\mathcal{K}\ni k\to\infty}(G_1(\hat{\lambda}^k) + G_2(\lambda^k)) &= \sum_{i=1}^{2}(g_i(\bar{x}_i) + \bar{\lambda}^\top A_i \bar{x}_i) \\
&= p^\star = L(\lambda^\star)\,, \quad (38)
\end{aligned}$$

where the second-to-last equality follows from primal optimality (and feasibility) of $(\bar{x}_1, \bar{x}_2)$, and the last equality follows from strong duality. Adding (35) and (37) and taking the limit $k \to +\infty$ ($k \in \mathcal{K}$),

$$\lim_{\mathcal{K}\ni k\to\infty}(G_1(\hat{\lambda}^k) + G_2(\lambda^k)) \le \sum_{i=1}^{2}(g_i(x_i) + \bar{\lambda}^\top A_i x_i)\,,$$

for all $x_i \in X_i$, $i = 1, 2$. In particular, taking the infimum on the right-hand side:

$$\lim_{\mathcal{K}\ni k\to\infty}(G_1(\hat{\lambda}^k) + G_2(\lambda^k)) \le G_1(\bar{\lambda}) + G_2(\bar{\lambda})\,. \quad (39)$$

From (39) and (38) we conclude that $G_1(\bar{\lambda}) + G_2(\bar{\lambda}) \ge L(\lambda^\star)$, showing that $\bar{\lambda}$ is dual optimal.

The sequence $\{(\boldsymbol{x_1^k, x_2^k, \lambda^k})\}$ converges. We have seen that $\{(x_1^k, x_2^k, \lambda^k)\}$ has limit points and they are primal-dual optimal. Now, we will see that this sequence has only one limit point, and thus converges. Let $(\bar{x}_1, \bar{x}_2, \bar{\lambda})$ be a limit point of $\{(x_1^k, x_2^k, \lambda^k)\}$. Since it is primal-dual optimal, (29)-(31) hold with $(x_1^\star, x_2^\star, \lambda^\star)$ replaced by $(\bar{x}_1, \bar{x}_2, \bar{\lambda})$. We had seen that V^k was convergent because it was bounded and nonincreasing. With the previous replacement, we now see that its limit is 0. Therefore, $\lambda^k \to \bar{\lambda}$ and $x_2^k \to \bar{x}_2$ (because A_2 has full column-rank). Since $r^k = A_1(x_1 - \bar{x}_1) + A_2(x_2 - \bar{x}_2) \to 0$, we also have $x_1^k \to \bar{x}_1$ (because A_1 has full-column rank), i.e., $\{(x_1^k, x_2^k, \lambda^k)\}$ converges. □

APPENDIX B
PROBLEM FOR EACH NODE IN LASSO

When LASSO (20) is solved in the column partition case, node p has to solve, in each iteration,

$$\underset{\lambda}{\text{minimize}}\;\; g_p(\lambda) + v^\top\lambda + c\|\lambda\|^2\,, \quad (40)$$

for some vector v and scalar c, and with $g_p(\lambda) = \frac{1}{P}(b^\top\lambda + \sigma\|\lambda\|) - \inf_{x_p}(\|x_p\|_1 + (A_p^\top\lambda)^\top x_p + \frac{\delta}{2}\|x_p\|^2)$. Define

$$\phi_p(\lambda) = -\inf_{x_p}\;\|x_p\|_1 + (A_p^\top\lambda)^\top x_p + \frac{\delta}{2}\|x_p\|^2\,, \quad (41)$$

and let $x_p(\lambda)$ denote the solution of the optimization problem in (41) for a given λ. Since the objective in that problem is strictly convex, ϕ_p is differentiable and its gradient is given by $\nabla\phi_p(\lambda) = -A_p x_p(\lambda)$. Furthermore, each component of $x_p(\lambda)$ can be found in closed-form: $-(r_i+1)/\delta$ if $r_i < -1$, $-(r_i-1)/\delta$ if $r_i > 1$, and 0 otherwise, where r_i denotes the ith row of $A_p^\top\lambda$. Problem (40) then becomes equivalent to

$$\underset{\lambda}{\text{minimize}}\;\; \phi_p(\lambda) + \frac{1}{P}b^\top\lambda + \frac{\sigma}{P}\|\lambda\| + v^\top\lambda + c\|\lambda\|^2\,,$$

or introducing an epigraph variable t,

$$\begin{array}{ll}\underset{x,t}{\text{minimize}} & \phi_p(\lambda) + \frac{1}{P}b^\top\lambda + \frac{\sigma}{P}t + v^\top\lambda + c\|\lambda\|^2 \\ \text{subject to} & \|\lambda\| \le t\,.\end{array} \quad (42)$$

Problem (42) can be solved using a projected gradient method. We will use FISTA [13, §4], whose complexity is $O(1/k^2)$. FISTA solves

$$\underset{\lambda}{\text{minimize}}\;\; f(\lambda) + g(\lambda)\,, \quad (43)$$

where f is a convex continuously differentiable function with a Lipschitz continuous gradient ∇f with constant L (i.e., $\|\nabla f(\eta) - \nabla f(\lambda)\| \le L\|\eta - \lambda\|$ for any λ, η), and g is a convex function. FISTA consists of iterating $\lambda^{k+1} = \left[\lambda^k - \frac{1}{L}\nabla f(\lambda^k))\right]^+$, where

$$[\eta]^+ = \arg\min_z\;\; g(z) + \frac{L}{2}\|z - \eta\|^2\,. \quad (44)$$

We set $f(\lambda)$ to be the objective of (42), and $h(\lambda)$ to be the indicator function of the set $\mathcal{S} := \{(\lambda, t) : \|\lambda\| \le t\}$. In this case, (44) becomes the projection onto $\mathcal{S}$, which can be shown to be

$$[(\lambda,t)]_{\mathcal{S}}^+ = \begin{cases} (\lambda, t) & , \quad t \ge \|\lambda\| \\ (0,0) & , \quad t \le -\|\lambda\| \\ \frac{t+\|\lambda\|}{2}(\frac{\lambda}{\|\lambda\|}, 1) & , \quad -\|x\| < t < \|x\|\,. \end{cases}$$

The Lipschitz constant of ∇f can also be shown to be $\sigma_{\max}^2(A_p)/\delta + 2c$, where $\sigma_{\max}(A_p)$ is the largest singular value of A_p.