DISTRIBUTED ALGORITHMS FOR BASIS PURSUIT

João F. C. Mota[†‡], *João M. F. Xavier*[†], *Pedro M. Q. Aguiar*[†], *Markus Püschel*[‡]

[†]Institute for Systems and Robotics / IST, Lisboa, Portugal
{jmota,jxavier,aguiar}@isr.ist.utl.pt

[‡]Department of Electrical and Computer Engineering
Carnegie Mellon University Pittsburgh, PA, USA
pueschel@ece.cmu.edu

ABSTRACT

The Basis Pursuit (BP) problem consists in finding a least ℓ_1 norm solution of the underdetermined linear system $Ax = b$. It arises in many areas of electrical engineering and applied mathematics. Applications include signal compression and modeling, estimation, fitting, and compressed sensing. In this paper, we explore methods for solving the BP in a distributed environment, *i.e.*, when the computational resources and the matrix A are distributed over several interconnected nodes. Special instances of this distributed framework include sensor networks and distributed memory and/or processor platforms. We consider two distribution paradigms: either the columns or the rows of A are distributed across the nodes. The several algorithms that we present impose distinct requirements on the degree of connectivity of the network and the per-node computational complexity.

Index Terms— Basis pursuit, distributed algorithms, dual methods, subgradient method, method of multipliers, sensor networks

1. INTRODUCTION

Representing a given signal as a sparse linear combination of elementary signals, drawn from an over–complete dictionary, is a problem of great theoretical and practical importance [1]. Formally, given a vector $b \in \mathbb{R}^m$, which may represent a signal of interest, and a matrix $A \in \mathbb{R}^{m \times n}$ with $m < n$, whose columns form an over–complete dictionary of elementary functions, the task of expressing b as a linear combination of a small subset of columns of A can be approximated by the convex problem

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

where $x \in \mathbb{R}^n$ is the variable to optimize and $\|x\|_1 = |x_1| + \ldots + |x_n|$. Hereafter, we assume that A has full rank, in order to secure the feasibility of (BP) for any given b. Problem (BP) is known as the *basis pursuit* problem [2].

Problem statement. The goal in this paper is to devise methods that solve (BP) when the matrix A is not available in a single node, but is rather spread among several nodes, each one having some computational power. We consider that either the columns or the rows of A are distributed across the nodes. This leads us to two problem statements: 1) (resp. 2)) *Given P nodes, each one storing a subset of columns (resp. rows) of A, find network topologies along with distributed algorithms for solving* (BP). In problem 1) we say that the partition of A is horizontal, while in problem 2) it is vertical.

Related work. To the best of our knowledge, there is no previous work concerning solving (BP) in a distributed environment, as studied herein. It is well known that (BP) can be reformulated as a linear program (LP) [3] and the work in [4] proposed a generalization of the LP-simplex method for a distributed scenario, assuming a horizontal partition of A. However, the resulting algorithm requires a fully connected topology in the sense that, at each iteration, every node must be able to communicate with all the remaining ones. In this paper, we present methods with less demanding graph topologies (moreover, they are not based on LP reformulations of (BP)).

2. HORIZONTAL PARTITION

In this section, we consider that the columns of A are distributed among P nodes. The pth node stores the pth submatrix[1] in the horizontal partition $A = [\,A_1 \cdots A_p \cdots A_P\,]$, where $A_p \in \mathbb{R}^{m \times n_p}$, $n_1 + \cdots + n_P = n$. The P nodes have to be connected (we seek sparse network topologies) and cooperate in order to solve (BP). The methods we propose are based on duality [5]. It is well known that, under the appropriate conditions (*e.g.*, separability of cost function and constraints), dual programs can be tackled by distributed methods. We propose two equivalent reformulations of (BP) whose associated dual programs are amenable to distributed solution methods. These are explained in the two remaining subsections.

2.1. Bounded BP

Let $R > 0$ be a strict upper bound on the ℓ_∞ norm of any solution of (BP). Then, (BP) is equivalent to the *bounded basis pursuit* (BBP) problem

$$\begin{array}{ll} \text{minimize} & \|x\|_1 \\ \text{subject to} & Ax = b \\ & \|x\|_\infty \le R \end{array} . \qquad \text{(BBP)}$$

Before we proceed, we note that such an R is easily found: given any x such that $Ax = b$, pick $R > n\|x\|_\infty$. Indeed, if $x^\star$ denotes a solution of (BP) then $\|x^\star\|_\infty \le \|x^\star\|_1 \le \|x\|_1 \le n\,\|x\|_\infty < R$.

Partially supported by grants SIPM 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.

[1] It is well known that explicit storage might not be required in some cases, only the ability of implementing the operator A_p and its adjoint $A_p^\top$.

Dual problem. Dualizing only the equality constraints in (BBP) leads to the unconstrained dual program

$$\underset{\lambda}{\text{maximize}} \quad L(\lambda) \ , \qquad \text{(DBBP)}$$

where $L(\lambda) = b^\top \lambda + \sum_{p=1}^{P} L_p(\lambda)$ and

$$L_p(\lambda) = \inf\{\|x_p\|_1 - \lambda^\top A_p x_p \ : \ \|x_p\|_\infty \leq R\}.$$

We used the decomposition $x = (x_1, \ldots, x_p, \ldots, x_P)$ where $x_p \in \mathbb{R}^{n_p}$. It can be shown that the dual of (BP) is not unconstrained. This is the motivation for inserting the redundant constraint $\|x\|_\infty \leq R$: it makes the corresponding dual unconstrained, and therefore directly applicable to a distributed setting (since the cost function is already separable).

Solving the dual. We propose to solve (DBBP) by a standard subgradient method[2] [5], *i.e.*,

$$\lambda^{k+1} = \lambda^k + \alpha^k g(\lambda^k), \qquad (1)$$

where k denotes the iteration number, $\alpha^k > 0$ is a stepsize and $g(\lambda^k)$ is a supergradient[3] of the concave dual function L at the point λ^k. This algorithm is globally convergent, under an appropriate choice for the stepsizes, see [5]. Using subgradient calculus [6, 5] it can be shown that, for any given λ, the supergradient $g(\lambda)$ can be chosen as $g(\lambda) = b - \sum_{p=1}^{P} A_p x_p^\star(\lambda)$ where the ith component of $x_p^\star(\lambda)$ is $\phi(r_i)$; here, r_i denotes the inner-product between λ and the ith column of A_p and

$$\phi(t) = \begin{cases} -R, & \text{if} \quad t \leq -1 \\ 0, & \text{if} \quad -1 < t < 1 \\ R, & \text{if} \quad t \geq 1 \end{cases} .$$

Network topology. We propose to implement the subgradient method in a network where the P nodes are connected to a central node, see figure 1(a). The central node implements the recursion (1) as follows: (i) λ^k is broadcasted to the P nodes, (ii) the pth node computes $x_p^\star(\lambda)$ and sends $A_p x_p^\star(\lambda)$ to the central node — this is done in parallel for all P nodes —, (iii) the central node adds the received data to obtain $g(\lambda^k)$ and moves to λ^{k+1}. It is assumed that b is available at the central node. Note that the computational complexity associated with each of the P nodes is very low.

Solving the primal. Let $\lambda^\star$ be a solution of (DBBP). Through the KKT optimality conditions of (BBP), we can infer valuable information about the solutions of (BBP) (hence of (BP) since they are equivalent). More precisely, let the support of a vector be the indexes of its nonzero entries, *i.e.*, $\operatorname{supp} x = \{i \ : \ x_i \neq 0\}$. Then, it can be seen through the KKT system that, for any solution $x^\star$ of (BBP), we have

$$\operatorname{supp} x^\star \subset \mathcal{I}^\star := \{i \ : \ |a_i^\top \lambda^\star| \geq 1\} \qquad (2)$$

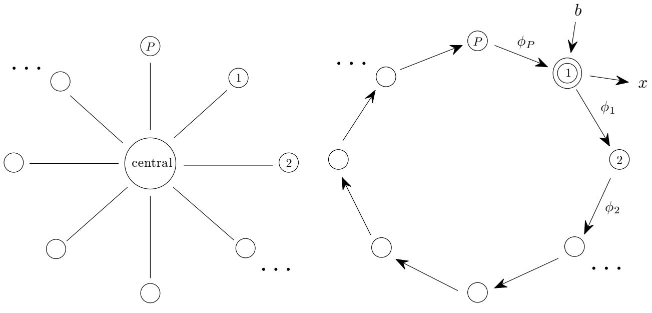
where a_i is the ith column of A (actually, $|a_i^\top \lambda^\star| > 1$ cannot occur due to our assumption on R, but we omit this technical discussion). In other words, once the central node knows $\lambda^\star$ it has access to an overestimate of the support of a solution of (BBP). In particular, for $i \notin \mathcal{I}^\star$, we have $x_i^\star = 0$. Thus, at this point, the central node could solve a *reduced* basis pursuit problem, *i.e.*, a smaller (BP) where the columns of A outside $\mathcal{I}^\star$ are removed. Since the solutions of (BP) are expected to be highly sparse, this could represent a significant reduction in the problem dimensionality.

The aforementioned strategy relies on the knowledge of $\lambda^\star$. In practice, the subgradient method is stopped after a finite number of iterations at the central node and only an estimate $\hat{\lambda}$ of a solution $\lambda^\star$ is available there. We adapt the previous scheme as follows. Let $\hat{\mathcal{I}} = \{i \ : \ |a_i^\top \hat{\lambda}| \geq 1 - \xi\}$ where $\xi > 0$ denotes a user-defined tolerance. Let $A_{\hat{\mathcal{I}}}$ be the columns of A indexed by $\hat{\mathcal{I}}$ (the notation $x_{\hat{\mathcal{I}}}$ follows a similar definition). Two cases are now possible, based on the feasibility of the linear system $A_{\hat{\mathcal{I}}} x_{\hat{\mathcal{I}}} = b$. Case 1: system is feasible – we let $x_{\hat{\mathcal{I}}}^\star$ be a solution of a reduced (BP) with A replaced by $A_{\hat{\mathcal{I}}}$. Case 2: system is unfeasible – we let $x_{\hat{\mathcal{I}}}^\star$ be a solution of the least-squares problem

$$\underset{x_{\hat{\mathcal{I}}}}{\text{minimize}} \quad \|A_{\hat{\mathcal{I}}} x_{\hat{\mathcal{I}}} - b\|^2 \ .$$

In either cases, we set $x_i^\star = 0$ for $i \notin \hat{\mathcal{I}}$. Case 2 is usually a consequence of a premature stop of the subgradient method.

Recall that A is stored in the P nodes, not in the central node. However, $A_{\hat{\mathcal{I}}}$ is needed there to address either case 1 or 2. This information can be obtained as follows: after stopping the subgradient method, the central node sends $\hat{\lambda}$ to the P nodes which, in an obvious and parallel way, can compute their corresponding fragments of $\hat{\mathcal{I}}$ and $A_{\hat{\mathcal{I}}}$ and report them back to the central node.



(a) Algorithm 1 (b) Algorithm 2

Fig. 1. Minimal architecture for the links between the nodes.

Overall algorithm. We gather our results in the form of an overall algorithm. Clearly, we have two phases. In the first phase, the central node runs the subgradient method (1) to obtain $\hat{\lambda}$. This is done by exchanging several messages with the P nodes, which work in parallel. In the second phase, $\hat{\lambda}$ is sent to the P nodes which report back their parts of $\hat{\mathcal{I}}$ and $A_{\hat{\mathcal{I}}}$. Then, the central nodes goes into case 1 or 2, and builds $x^\star$.

Algorithm 1. *Input:* b at the central node (and matrices A_p stored at each node). *Output:* $x^\star$ at the central node

Computation at the central node:

1. Iterate (1) until some stopping criterion is met

   (a) Send λ^k to each node

   (b) For each node $p = 1, \ldots, P$ do
   Computation at node p:

   - Compute $x_p^\star(\lambda^k)$

[2] Although this terminology is usually used for the minimization of a convex function, we decided to keep it in our case: the maximization of a concave function.

[3] The vector s is a supergradient (resp. subgradient) of a concave (resp. convex) function f at the point x if $f(y) \leq f(x) + s^\top(y - x)$ (resp. $f(y) \geq f(x) + s^\top(y - x)$) holds for all y.

```
            • Return $A_p x_p^\star(\lambda^k)$ to the central node
        (c) Compute $g(\lambda^k) = b - \sum_{p=1}^{P} A_p x_p^\star(\lambda^k)$
        (d) $\lambda^{k+1} = \lambda^k + \alpha^k g(\lambda^k)$
    2. $\hat{\lambda} = \lambda^{k+1}$
    3. Send $\hat{\lambda}$ to each node
    4. For each node $p = 1, \ldots, P$ do
       Computation at node p:
         • Compute the $p$th fragment of $\mathcal{I}$, say $\hat{\mathcal{I}}_p$
         • Return $\hat{\mathcal{I}}_p$ and $(A_p)_{\hat{\mathcal{I}}_p}$ to the central node
    5. Receive $\hat{\mathcal{I}}_p$ and $(A_p)_{\hat{\mathcal{I}}_p}$ from each node
    6. Concatenate them to obtain $\hat{\mathcal{I}}$ and $A_{\hat{\mathcal{I}}}$
    7. Run case 1 or 2 to obtain $x^\star$
```

2.2. Augmented BP

In this subsection, we need an assumption whose utility will became clear soon. We assume that in the horizontal partition $A = [\, A_1 \cdots A_p \cdots A_P \,]$ there holds

$$\operatorname{rank} A_p = n_p, \tag{3}$$

that is, each A_p has full column rank. Note that this automatically imposes that $n_p \leq m$ for all p.

The starting point for our new approach is based on a reformulation of (BP). Namely, for any given $\rho > 0$, we note that (BP) is equivalent to

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

since we just augmented the cost function with a term which evaluates to zero (because the constraints $Ax = b$ are in force).

Dual problem. Although (BP) and (ABP) are equivalent, their duals are distinct. In particular, the dual of (ABP) is unconstrained in the sense that it corresponds to

$$\underset{\lambda}{\text{maximize}} \quad L(\lambda) \tag{DABP}$$

and $L(\lambda)$ is finite everywhere, *i.e.*, $L(\lambda) > -\infty$ for all $\lambda \in \mathbb{R}^m$. Here, $L(\lambda) = \inf\{L(x, \lambda) \,:\, x \in \mathbb{R}^n\}$ and

$$L(x, \lambda) = \|x\|_1 - \lambda^\top(Ax - b) + \frac{\rho}{2}\|Ax - b\|^2 . \tag{4}$$

Solving the dual. We propose to solve (DABP) through the iterations

$$\lambda^{k+1} = \lambda^k + \rho g(\lambda^k) \tag{5}$$

where $g(\lambda^k)$ is a supergradient of the concave dual function $L(\lambda)$ at the point λ^k: it can be computed as $g(\lambda^k) = b - Ax^\star(\lambda^k)$ where $x^\star(\lambda)$ is a solution of

$$\underset{x}{\text{minimize}} \quad L(x, \lambda) \ . \tag{6}$$

The iterative scheme (5) is nothing else but the method of multipliers [5] applied to (BP). Also, $L(x, \lambda)$ in (4) is known as the augmented Lagrangian of (BP). The method of multipliers enjoys an important property: any cluster point of the sequence $\{x^\star(\lambda^k) \,:\, k = 1, 2, \ldots\}$ solves (BP), see [5].

Network topology. We propose to implement the method of multipliers (5) in a ring-shaped network, see figure 1(b). Note the absence of a central node. We arrive at this network configuration by reasoning as follows.

Let λ be fixed and focus on solving (6) with respect to $x = (x_1, \ldots, x_p, \ldots, x_P)$, $x_p \in \mathbb{R}^{n_p}$. As we have P nodes available, we would like to distribute this task over them. Note that $L(\cdot, \lambda)$ is not separable with respect to the x_p's, a well known drawback of the augmented Lagrangian approach. That is, a straightforward decoupling into P independent subproblems is not possible. We tackle the minimization of $L(\cdot, \lambda)$ by the nonlinear Gauss-Seidel method [5, page 267] (another viable method would be the diagonal quadratic approximation method [7], whose convergence properties can be established for this scenario, see [8]). This consists in minimizing $L(\cdot, \lambda)$ through sweeps. Each sweep updates the x_p's by minimizing $L(\cdot, \lambda)$ with respect to each x_p while keeping fixed x_q, $q \neq p$. A more precise description follows:

Nonlinear Gauss-Seidel method. *Input:* λ and initial point $x^0 = (x_1^0, \ldots, x_p^0, \ldots, x_P^0)$. *Output:* $x^\star(\lambda)$ solving (6)

```
    1. Initialize $t = 0$
    2. Iterate on $t$ until some stopping criterion is met:
       For all $p = 1, \ldots, P$ find
         $x_p^{t+1} = \arg\min_{x_p} L(x_1^{t+1}, \ldots, x_{p-1}^{t+1}, x_p, x_{p+1}^t, \ldots, x_P^t, \lambda)$      (7)
    3. $x^\star(\lambda) = (x_1^t, \ldots, x_p^t, \ldots, x_P^t)$
```

It is because assumption (3) holds that each minimization in (7) is well-posed, *i.e.*, there is an unique minimizer. Regarding the convergence of the nonlinear Gauss-Seidel method in our context, the optimality of any cluster point of the sequence $\{(x_1^t, \ldots, x_p^t, \ldots, x_P^t) \,:\, t = 1, 2, \ldots\}$ is guaranteed despite the non-smoothness of the cost function in (7), see [9].

Consider now the minimization problem in (7). It is natural to assign this task to node p. In the following, we show that it can be so. We start by noting that (7) boils down to

$$\underset{x_p}{\text{minimize}} \quad \tfrac{1}{2}\|A_p x_p - \gamma_p^t\|^2 + \tfrac{1}{\rho}\|x_p\|_1 \ , \tag{8}$$

where

$$\gamma_p^t := b - \sum_{i<p} A_i x_i^{t+1} - \sum_{i>p} A_i x_i^t + \frac{1}{\rho}\lambda.$$

Since node p has access to A_p, problem (8) can be tackled there if γ_p^t is available. Now, note the identity

$$\gamma_p^t = \underbrace{\left(\gamma_{p-1}^t - A_{p-1} x_{p-1}^{t+1}\right)}_{\phi_{p-1}} + A_p x_p^t. \tag{9}$$

Here, ϕ_{p-1} is available at the $(p-1)$th node while $A_p x_p^t$ is available at the pth node (from the previous iteration). Thus, at the pth node, it suffices to receive ϕ_{p-1} from the $(p-1)$th node to build γ_p^t and tackle (8), *e.g.*, through the method in [10]. This flow of information can be implemented in a circular network like the one in figure 1(b).

So far, only problem (6) has been discussed. However, the whole method of multipliers (5) can be implemented in the proposed network by letting node 1 do the bookkeeping of λ^k. Indeed, for a given λ^k, suppose T sweeps around the network solve (6) for $\lambda =$

λ^k (one sweep corresponds to one t-iteration in the nonlinear Gauss-Seidel method). After the T sweeps, we have $x_p^t \simeq x_p^{t+1}$ (since we assume that the nonlinear Gauss-Seidel method has converged), and thus node 1 can calculate

$$g(\lambda^k) = \gamma_1^T - A_1 x_1^T - \frac{1}{\rho}\lambda^k.$$

This enables node 1 to update λ according to (5). Further, this node can trigger a new set of sweeps to minimize (6) for $\lambda = \lambda^{k+1}$ because it can incorporate this knowledge into γ_1 by making

$$\gamma_1^0 = \gamma_1^T - \frac{1}{\rho}(\lambda^k - \lambda^{k+1}).$$

Solving the primal. After the dual problem (DABP) is solved, it suffices to do a cycle through the network to collect $x(\lambda^\star)$ which corresponds to the last solutions of (7) stored in each node.

Overall algorithm. Leaving out the details of the (already seen) updates of λ, we summarize our results in the following algorithm.

Algorithm 2. *(Procedure for nodes $p = 1, \ldots, P$) Input:* b at node 1 (and matrices A_p stored at each node). *Output:* $x^\star$ at node 1

```
1. Receive $\phi_{p-1}$ from node $p-1$
2. if $p = 1$, $t \leftarrow t+1$, unless some stopping criterion is met (in this case $t \leftarrow 0$)
3. Set $\gamma_p^t = \phi_{p-1} + A_p x_p^t$
4. Find $x_p^{t+1} = \arg\min_{x_p} \frac{1}{2}\|A_p x_p - \gamma_p^t\|^2 + \frac{1}{\rho}\|x_p\|_1$
5. Send $\phi_p = \gamma_p^t - A_p x_p^{t+1}$ to node $p+1$
```

In practice, it is better to also update ρ in each iteration of the method of multipliers as $\rho^{k+1} = c\rho^k$ where $c \geq 1$, see [5]. This can be easily incorporated in our algorithm (details are skipped). We used this latter version in our simulations.

Finally, notice that, due to the fact that in this algorithm only one node "works" at each iteration, it is possible to solve P simultaneous problems (with the same matrix A), using the architecture of figure 1(b).

3. VERTICAL PARTITION

In this section, we consider that the rows of A are distributed among the P nodes. The pth node stores the pth submatrix in the vertical partition

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

where $A_p \in \mathbb{R}^{m_p \times n}$, $m_1 + \cdots + m_P = m$. This scenario arises when a network of interconnected sensors, where each sensor only measures $b_p := A_p x \in \mathbb{R}^{m_p}$, must reconstruct the sparse vector x without the nodes exchanging information about their sensing devices (rows of matrix A).

Network topology. Contrary to the approaches in section 2, we will not impose any particular network structure. In other words, we let the network topology be given to us. We just assume that the given network of P nodes is connected. Let $\mathcal{E} = \{(i,j)\}$ denote the set of edges of the given network. We adopt the convention that $i < j$ for each edge $(i,j) \in \mathcal{E}$. See figure 2 for an example.

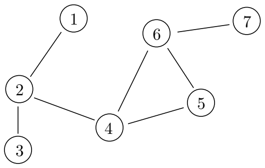


Fig. 2. Example of a network in which algorithm 3 can run. The set of edges is $\mathcal{E} = \{(1,2),(2,3),(2,4),(4,5),(4,6),(5,6),(6,7)\}$.

Cloned BP. To design a distributed algorithm for this environment, we start by reformulating (BP) by cloning the optimization variable. More precisely, problem (BP) is equivalent to

$$\begin{array}{ll} \text{minimize} & \sum_{p=1}^P \frac{1}{P}\|x_p\|_1 + \frac{\rho}{2}\sum_{(i,j)\in\mathcal{E}} \|x_i - x_j\|^2 \\ \text{subject to} & A_p x_p = b_p, \quad p = 1, \ldots, P \\ & x_i = x_j, \quad (i,j) \in \mathcal{E} \end{array} \quad , \qquad \text{(CBP)}$$

where the variable to optimize is $x = (x_1, \ldots, x_P) \in (\mathbb{R}^n)^P$ and $\rho > 0$ is a constant. Problem (CBP) is equivalent to (BP) because the graph is connected: thus, $x_1 = x_2 = \cdots = x_P$ (we just duplicated the variable).

Dual problem. Dualizing only the constraints $x_i = x_j$ in (CBP) leads to the unconstrained problem

$$\underset{\{\lambda_{(i,j)} \,:\, (i,j)\in\mathcal{E}\}}{\text{maximize}} \quad L(\{\lambda_{(i,j)}\}) \quad , \qquad \text{(DCBP)}$$

where

$$L(\{\lambda_{(i,j)}\}) = \inf\left\{L(\{x_p\},\{\lambda_{(i,j)}\}) \,:\, A_p x_p = b_p, \text{ for all p}\right\}$$

and

$$L(\{x_p\},\{\lambda_{(i,j)}\}) = \sum_{p=1}^P \frac{1}{P}\|x_p\|_1 + \sum_{(i,j)\in\mathcal{E}} \Big[\lambda_{(i,j)}^\top (x_i - x_j) + \frac{\rho}{2}\|x_i - x_j\|^2\Big] . \qquad (10)$$

Solving the dual. The dual problem (DCBP) can be solved by the method of multipliers, see a related discussion in subsection 2.2. In the current setup, this corresponds to the iterations

$$\lambda_{(i,j)}^{k+1} = \lambda_{(i,j)}^k + \rho\left(x_i^\star\left(\{\lambda_{(i,j)}^k\}\right) - x_j^\star\left(\{\lambda_{(i,j)}^k\}\right)\right) \qquad (11)$$

where $\{x_p^\star(\{\lambda_{(i,j)}\})\}$ minimize (10) with respect to $\{x_p\}$ subject to $A_p x_p = b_p$, for $\lambda_{(i,j)}$ fixed at $\lambda_{(i,j)}^k$.

We employ the nonlinear Gauss-Seidel method to find

$$x^\star\left(\{\lambda_{(i,j)}\}\right) = \left(x_1^\star(\{\lambda_{(i,j)}\}), \ldots, x_P^\star(\{\lambda_{(i,j)}\})\right).$$

After straightforward calculations, it can be shown that updating the variable $x_p \in \mathbb{R}^n$ inside a nonlinear Gauss-Seidel sweep amounts to solving at the pth node the program

$$\begin{array}{ll} \text{minimize} & \frac{1}{P}\|x_p\|_1 + \sum_{j\in\mathcal{N}_p}(\text{sign}(j-p)\,\lambda_{(p,j)} - \rho x_j)^\top x_p \\ & \qquad + \frac{N_p\rho^k}{2}\|x_p\|^2 \\ \text{subject to} & A_p x_p = b_p \end{array} \qquad (12)$$

with respect to $x_p \in \mathbb{R}^n$, where $\mathcal{N}_p$ is the set of neighbors of node p and N_p its cardinality. According to the canonical script of the nonlinear Gauss-Seidel method [5], the minimizations are to be performed in the sequential order $p = 1, 2, \ldots, P$; thus, x_j in (12) stands for x_j^{t+1} if $j < p$ and for x_j^t if $j > p$. Actually, the order of the minimizations can be arbitrary as long as there is an integer M such that each x_p is updated at least once in every group of M contiguous iterations [9]. Hence, we don't need to worry about the sequence of our minimizations: we can let that sequence be determined by the topology of the graph. This also means that the algorithm is robust to instantaneous or even permanent link failures, provided the resulting graph remains connected.

Looking at (12), we see that the pth node only needs the variables x_j^{t+1} and x_j^t that belong to its neighbors. Further, this knowledge is also sufficient to update, after each iteration k of the method of multipliers (which correspond to several sweeps of the nonlinear Gauss-Seidel method) the relevant subset of the dual variables $\{\lambda_{(i,j)}\}$. In conclusion, this algorithm is totally distributed and can run on any connected graph.

Solving the primal. After the dual problem (DCBP) is solved, a solution to (BP) is available at any node: it corresponds to $x^\star\left(\{\lambda^\star_{(i,j)}\}\right)$.

Overall algorithm. We summarize our findings.

Algorithm 3. *(Procedure for nodes* $p = 1, \ldots, P$*) Input:* $b_p \in \mathbb{R}^{m_p}$ at each node (and matrices A_p stored at each node). *Output:* $x^\star$ at each node

Iterate on k (method of multipliers)

1. If it receives x_j^{t+1} or x_j^t from all its neighbors,
   (a) Form $\tau_p = \sum_{j \in \mathcal{N}_p} \left(\operatorname{sign}(j-p)\,\lambda_{(p,j)} - \rho x_j\right)$
   (b) Find x_p^{t+1} that solves (12) (using τ_p)
   (c) Send x_p^{t+1} to all neighbors ($\mathcal{N}_p$)
2. else, in the idle times of the links, exchange convergence information with the neighbors; and, whenever the nonlinear Gauss-Seidel sweeps stop, update the dual variables $\lambda_{(p,j)}$, for all $j \in \mathcal{N}_p$.

4. EXPERIMENTAL RESULTS

To test the behavior of the distributed algorithms in practice, we simulated them in an ordinary (single) processor, using Matlab. The results presented in table 1 validate the fact that all the algorithms converge, in general, to a point close to an optimal solution, as predicted by the theory. However, as each algorithm has more than one parameter to tune, we cannot draw strict conclusions with respect to the running times: we would need real distributed simulations, varying exhaustively the parameters. Since the paradigms for the horizontal and vertical partitions are different, we cannot directly compare the results between the respective algorithms.

The first two rows of table 1 show the average relative error in the variable x and in the cost function $f(x)$, respectively. The correct solutions, $x^\star$ and $f(x^\star)$, were considered to be the output of a Matlab linear program solver that solved (BP) in its linear program format. The numbers of the rows that contain time measurements are the average of the total time that the algorithms took to solve each problem, the respective time spent in operations that could be done in parallel (note that these simulations were performed on a single processor), and the estimated time for the execution of the algorithms if we really had P nodes available. Note that the timings are for illustration only as they do not take into account communication time or other effects of a real distributed environment.

Table 1. Results from the simulations. The numbers shown are an average of 15 random experiments.

	Horizontal partition		Vertical partition
Feature	Algorithm 1	Algorithm 2	Algorithm 3
Error in x [%]	2.26	3.69×10^{-2}	6.07×10^{-1}
Error in $f(x)$ [%]	6.96×10^{-1}	1.96×10^{-2}	7.07×10^{-2}
Total Time [s]	6.90×10^{-1}	3.66	7.59×10^{1}
Time in Parallel [s]	5.84×10^{-1}		2.17×10^{1}
Estimated Time [s]	1.64×10^{-1}	3.66	5.73×10^{1}

Horizontal partition. For the horizontal partition, each node ($P = 10$) stored 25 columns of a matrix $A \in \mathbb{R}^{50 \times 250}$, where each entry of the matrix was drawn independently from a Gaussian distribution. Obviously, the simulated environments were those of figure 1. We first ran algorithm 2 and, after that, we ran algorithm 1 stopping it when the dual function achieved 99% of the value returned by algorithm 2. While the errors in x and $f(x)$ for the algorithm 2 are representative of what happened in all experiments (never achieving exactly the correct solution), the same errors for the algorithm 1 are not. In fact, in more than half of the experiments, algorithm 1 returned the exact optimal solution. Concerning time estimates, the estimated running time for algorithm 1 was calculated using $t_{\text{est}} = t_T + ((1-P)/P)t_p$, where t_T and t_p designate, respectively, the total and parallel times. As in algorithm 2 only one node can "work" at each iteration, the estimated time coincides with the total time. However, we must be aware that it is possible to solve P simultaneous problems using algorithm 2, so the time to solve each problem is, in average, less than the one we obtained (by a factor of P).

Vertical partition. The simulations of algorithm 3 were done using the network of figure 2, where each node stored one row of a matrix $A \in \mathbb{R}^{7 \times 30}$ (each entry of which was randomly obtained from a Gaussian distribution). To understand how the time estimates were calculated, we must first understand the dynamics of the calculation of new variables. Suppose that, at some iteration t of the nonlinear Gauss-Seidel sweep, node 2 of figure 2 computes x_2^t. The next step consists in sending this variable to the neighbors 1, 3 and 4 which, at iteration $t+1$, compute their new variables. After sending them to node 2, this node is again able to compute a new variable, x_2^{t+1}. Hence, each node computes a new variable every two iterations. This also means that, in average, half of the nodes of a network are "working" (computing new variables) at each iteration. So, the time in parallel can be computed as $t_p = (2/P)t_T$, where t_T is the total time, whereas the estimated running time is given by the same expression we used for algorithm 1.

General comments. Independently of the complexity of the operations that a node has to perform at each iteration, all the three algorithms require many communications between the nodes before their convergence. Although this depends on several parameters and on the choice of the stopping criteria, we can say that algorithm 1 needs around 700 subgradient iterations, thus $1400P$ total communications. In the simulations, algorithm 2 required around 3500 total communications between the nodes, and algorithm 3 needed around 8000. In all algorithms, we can increase (resp. reduce) the communication overhead by increasing (resp. decreasing) the number of nodes. Additionally, in algorithm 3 we can explore the connectivity of the network: by increasing (resp. decreasing) it, we reduce (resp. increase) significantly the number of iterations of the nonlinear Gauss-Seidel method.

5. CONCLUSIONS

We propose three algorithms capable of solving the basis pursuit problem (BP) in a distributed environment. That distributivity is characterized by the fact that parts of the matrix A are stored in different nodes, being the nodes able to communicate among themselves.

Two of the algorithms are to be applied in the case where A is partitioned horizontally. The fundamental difference between those two algorithms is in the network topology that they require: one needs a central node and has the advantage that the other (distributed) nodes perform very simple calculations, while the other runs in networks where the nodes are connected circularly. In the latter, the calculations at each node are more complex, but it has the potential to solve P similar problems at the same time.

We also propose an algorithm that is able to solve (BP) when the partition of A is vertical. This algorithm can run in any connected network. This makes this algorithm attractive for sensor networks applications. However, the major disadvantage we found in it was the high number of communications (and thus the time consumed) required to converge, showing that there is margin for improvement.

6. REFERENCES

[1] 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.

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

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

[4] G. Yarmish, *A distributed implementation of the simplex method*, Ph.D. thesis, Polytechnic University, 2001.

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

[6] S. Boyd and L. Vandenberghe, "Subgradients, notes for ee364b," Stanford University, 10 January 2007.

[7] A. Zakarian, *Nonlinear Jacobi and ϵ-relaxation methods for parallel network optimization*, Ph.D. thesis, University of Wisconsin, Madison, 1995.

[8] J. Mota, "Distributed algorithms for sparse approximation," M.S. thesis, Instituto Superior Técnico, Portugal, 2008, `http://users.isr.ist.utl.pt/~aguiar/jmota_thesis.pdf`.

[9] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *Journal of Optimization Theory and Applications*, vol. 109, pp. 475–494, June 2001.

[10] M. Figueiredo, R. Nowak, and S. Wright, "Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems," *IEEE Journal of Selected Topics in Signal Processing: Special Issue on Convex Optimization Methods for Signal Processing*, vol. 1, no. 4, pp. 586–598, 2007.