DISTRIBUTED ALGORITHMS FOR SPARSE RECONSTRUCTION

João Mota

Instituto Superior Técnico
Lisbon, Portugal

ABSTRACT

Many applications require the knowledge of a sparse linear combination of elementary signals that can explain a given signal. This problem is known as "sparse approximation problem" and arises in many fields of electrical engineering and applied mathematics. The great difficulty when dealing with sparse approximation problems is the lack of convexity or the non-differentiability inherent to the sparsity measures.

This paper proposes and analyzes some distributed algorithms that solve a sparse convex approximation problem known as the "basis pursuit" problem. The interest in solving this problem distributedly concerns applications such as communications, computing and sensor networks. All the proposed algorithms assume that the matrix which contains the description of elementary signals is partitioned either horizontally or vertically among the several processors available. Nevertheless, each algorithm is based on a particular architecture for the links between the processors.

We also state results of convergence for all the proposed algorithms. This requires extending the well-known results of convergence of the Diagonal Quadratic Approximation and the Nonlinear Gauss-Seidel methods to cover a new class of non-differentiable functions, which we call "rigid functions".

Index Terms— Basis Pursuit, Distributed Algorithms, Nonlinear Gauss–Seidel, Diagonal Quadratic Approximation, Subgradient Method, Rigid Functions.

1. INTRODUCTION

Over the last half century, society has benefited a lot from the significant advances in signal processing. Ranging from image and audio processing to genetics and bioengineering, its applications are numerous. So, it is no surprise that developments on these application areas can be observed almost everyday. Nevertheless, the field of signal processing itself is far from being static. The topic of sparse representation of signals is a quite recent and promising one [11, 12].

At the same time, the computing architecture paradigm is changing towards more distributed environments, where computational resources are not just concentrated on a single place but over many. Sensor networks are just an example that, by itself, have several applications.

In this paper, we are interested in two important problems that arise in the context of sparse representation: the *basis pursuit*

$$\begin{array}{l}\min \quad \|x\|_1 \\ Ax = b \\ \text{var} : x \in \mathbb{R}^n\end{array} \tag{BP}$$

and the *basis pursuit denoising*

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|^2 + \beta\|x\|_1, \tag{BPDN}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $\beta \in \mathbb{R}$, $\|\cdot\|_1$ is the ℓ_1-norm and $\|\cdot\|$ is the ℓ_2-norm. The variable is $x \in \mathbb{R}^n$ for both problems. We assume that $m < n$ (usually, A is a "fat" matrix) and, without loss of generalization, that the rows of A are linearly independent. This way, we don't need to worry about the feasibility of (BP), since the matrix A can always span the vector b.

Although problems (BP) and (BPDN) are convex, it can be shown in that (BP) is a convex relaxation of a combinatorial problem involving the ℓ_0-quasi norm; and (BPDN) is a more robust way of solving (BP). Using standard techniques in optimization [5], one can easily see that (BP) is equivalent to a linear program (LP) and that (BPDN) is equivalent to a quadratic program (QP).

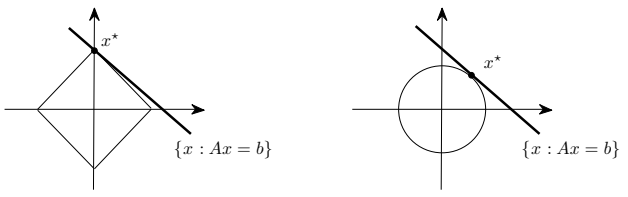
1.1. Preliminaries

This subsection provides some background, some analysis and some recent theoretical results about the BP and the BPDN.

1.1.1. Why is sparsity always associated with the ℓ_1-norm?

There is more than one answer for this question. Here, we will present perhaps the more intuitive one, comparing the preferred ℓ_1-norm to the widely used ℓ_2-norm. Fig. 1 compares the solutions that we get when we use the ℓ_1-norm and the ℓ_2-norm in the objective of the (BP). The optimal solution in both cases is the first point of the corresponding norm ball to "touch" the set $\{x : Ax = b\}$, as if we were swelling up

The author would like to thank the *Fundação para a Ciência e Tecnologia* for partial funding this project.



(a) ℓ_1-norm (b) ℓ_2-norm

Fig. 1. Graphical explanation of the sparsity induction property of the ℓ_1-norm.

the ball. It happens that the ℓ_1-norm ball is larger on the coordinate axis directions than in the other directions, while the ℓ_2-norm ball is isotropic.

So, if we seek a solution with lots of small (near–zero) components, the ℓ_1-norm is a good choice.

The authors of [7, 8, 6] have established rigorous results stating when it is possible to find a unique s-sparse vector[1] by solving the (BP). To understand that, we need the following definition:

Definition 1 (Restricted isometry constant [7])**.** *For each $s = 1, 2, \ldots, n$, the restricted isometry constant $\delta_s(A)$ of a matrix $A \in \mathbb{R}^{m\times n}$ is the smallest number such that*

$$(1 - \delta_s(A))\|x\|^2 \leq \|Ax\|^2 \leq (1 + \delta_s(A))\|x\|^2,$$

holds for all s-sparse vectors.

It is straightforward to prove that if the matrix A has a restricted isometry constant $\delta_{2s}(A) < 1$, then the linear system $Ax = b$ has a unique s-sparse solution. Further, if $\delta_{2s}(A) < \sqrt{2} - 1$ and if the vector b was generated by a vector $x \in \mathbb{R}^n$, *i.e.*, $b = Ax$, then the solution x^* of (BP) obeys $\|x^\star - x\| \leq C_0 s^{-1/2}\|x - x_s\|_1$, where x_s is the vector which is equal to x only at the largest s entries and zero elsewhere, and C_0 is a small constant. This claim is proved in [6]. Note that if x is s-sparse, then the error is zero.

The same authors also refer which kind of matrices have "good" restricted isometry constants, namely Gaussian random matrices with i.i.d. entries, Fourier ensembles and general orthogonal measurement ensembles.

Concerning the practical implications of this theory, [7, 1] state that we can code without loss (with an overwhelming probability) all the information of a s-sparse vector x of size n into a vector b of size m, with $m = \mathcal{O}(s\log(n/s))$, just by doing $b = Ax$, where A is a random matrix of the kind of the ones mentioned above.

1.1.2. The Basis Pursuit Denoising

If we know that the linear system $Ax = b$ is impossible to solve due to any perturbation, it makes more sense to solve, instead of (BP), the problem

$$\begin{array}{c} \min \quad \|x\|_1. \\ \|Ax - b\| \leq \epsilon \\ \text{var} : x \in \mathbb{R}^n \end{array} \tag{1}$$

Let x be s-sparse. There is a theorem in [6] that states that if the vector b was generated by $b = Ax + z$, where z is any perturbation bounded by ϵ, *i.e.* $\|z\| \leq \epsilon$, and if $\delta_{2s}(A) < \sqrt{2} - 1$, then the solution x^* of (1) obeys $\|x^\star - x\| \leq C_1\epsilon$, for some small constant C_1.

By analyzing the KKT-system [5] for (1), we can conclude that a solution of this problem is either $x^* = 0$ or also a solution of the (BPDN), for some $\beta > 0$. The usual procedure when we want to find a s-sparse vector from the resolution of (BPDN) is to solve it for several values of β and choose the solution with the desired sparsity.

1.1.3. The Dual of The BP

The main algorithms in this paper are based on the resolution of dual problems. When the problems are convex and verify some constraint qualification, *e.g. Slater's condition* [5], strong duality holds, meaning that the optimal value of the objectives of the primal and dual problems are equal. In this case, the knowledge of the dual optimal variable can provide useful information about the primal optimal variable and, in some cases, even allow its direct calculation.

The dual problem of (BP) can be shown to be

$$\begin{array}{c} \max \quad \lambda^T b. \\ \|A^T\lambda\|_\infty \leq 1 \\ \text{var} : \lambda \in \mathbb{R}^m \end{array} \tag{2}$$

It is a linear program, where we maximize the inner product $b^T\lambda$ over the polyhedron $\mathcal{P} = \left\{\lambda : \|A^T\lambda\|_\infty \leq 1\right\}$. Usually, the matrix A is "fat", *i.e.*, it has much more columns than rows: $m \ll n$. When this happens, the polyhedron $\mathcal{P}$ is characterized by many constraints. Then, any optimization method that relies on projections on the constraining set of an optimization problem becomes cumbersome (for example, gradient and subgradient projection methods [3, 4]).

2. DISTRIBUTED BP WITH HORIZONTAL PARTITION

In this section, we will address the problem of solving the BP in the case where matrix A is partitioned horizontally into P block matrices $A_p \in \mathbb{R}^{m\times n_p}$, for $p = 1, \ldots, n$ and with $n_1 + \ldots + n_P = n$.

$$A = \underbrace{\left[\begin{array}{ccccc} A_1 & \cdots & A_p & \cdots & A_P \end{array}\right]}_{n} \Big\updownarrow m$$

[1] A vector is s-sparse if it has at most s non–zero entries.

This kind of partition makes sense, for example, when we want to operate each family of functions (represented by a column of A) on different computers, for instance to adapt each computer architecture to a particular family of functions.

We assume, then, that no processor knows the entire matrix A. Nevertheless, all the P processors have to cooperate in order to solve the BP.

In the sequence, we will use dual problems to solve the BP but, in order to avoid the polyhedral constraints present in (2), we will use two "tricks":

- Guaranteeing that the primal constraint set is compact, and making use of Weierstrass's theorem.
- Transforming the dual Lagrangian function into a coercive function[2] with respect to the primal variable. Recall that a continuous coercive function has always a finite infimum (this method will be approached in subsection 2.2).

2.1. Subgradient Method

Here, we will assume that we know a sufficiently large R such that any optimal solution x^* of (BP) is contained in the interior of the ℓ_∞ ball: $B_\infty(0,R)$. Assuming this, problem

$$\begin{array}{c} \min \quad \|x\|_1, \\ Ax = b \\ x \in B_\infty(0,R) \\ \text{var} : x \in \mathbb{R}^n \end{array} \qquad \text{(BBP)}$$

which we will call *bounded basis pursuit* (BBP), is equivalent to (BP). If we know a vector $\hat{x} \in \mathbb{R}^n$ that satisfies the linear system $A\hat{x} = b$, then an R equal to $n\|\hat{x}\|_\infty + \epsilon$, for some small $\epsilon > 0$, guarantees that any optimal solution x^* of (BP) is in the interior of $B_\infty(0,R)$.

2.1.1. Theory Behind The Algorithm

By partitioning the variable x into P variables, $x = (x_1, x_2, \ldots, x_P)$, (BBP) is equivalent to

$$\begin{array}{c} \min \quad \sum_{p=1}^{P} \|x_p\|_1. \\ \sum_{p=1}^{P} A_p x_p = b \\ \|x_p\|_\infty \leq R,\ p = 1,2,\ldots,P \end{array} \qquad (3)$$

If we only dualize the equality constraints of (3), we get a problem equivalent to $\min_{\lambda \in \mathbb{R}^m} H(\lambda)$, where

$$H(\lambda) = -\lambda^T b - \sum_{p=1}^{P} \left[\inf_{\|x_p\|_\infty \leq R} \left(\|x_p\|_1 - \lambda^T A_p x_p \right) \right]. \quad (4)$$

The function $H(\lambda)$ can be written as the pointwise supremum of convex functions, thus it is a subdifferentiable function. As so, we can use an algorithm named *subgradient method* [4] in order to minimize it over $\mathbb{R}^m$. This method only requires the availability of one subgradient of $H(\lambda)$ at each point. It can be proved that the vector $g = A \cdot (\sum_{p=1}^{P} \tilde{x}_p(\lambda)) - b$ is a subgradient of $H(\lambda)$, where each component of $\tilde{x}_p(\lambda)$ is given by

$$\tilde{x}_p^j(\lambda) = \begin{cases} 0 & \text{,if} \quad |a_p^{j\,T}\lambda| < 1 \\ R \cdot \text{sign}\left(a_p^{j\,T}\lambda\right) & \text{,if} \quad |a_p^{j\,T}\lambda| \geq 1 \end{cases}, \quad (5)$$

for $j = 1, \ldots, n_p$, and for $p = 1, \ldots, P$.

The KKT-system for (BBP) allows us to conclude that the knowledge of the dual optimal variable λ^* gives information about the sign of each entry of the optimal primal variable x^* as follows: if $a_i^T\lambda^* = 1$, we have $x_i^* > 0$; if $a_i^T\lambda^* = -1$, then $x_i^* < 0$. And when $|a_i^T\lambda^*| < 1$, the column a_i of the matrix A isn't activated by the optimal solution, *i.e.*, $x_i^* = 0$. Note that this is valid for any $i = 1, \ldots, n$, that is, for every column a_i of the matrix A. This knowledge permits to discard the columns of A that aren't activated by the optimal primal solution and solve the (BP) in a lower dimension:

$$\min_{Mu=b} \|u\|_1. \qquad \text{(RBP)}$$

We call this problem a *reduced basis pursuit*. Note that the variable is the vector u. The matrix M is the matrix formed by the columns of A that we keep when we know an approximation $\hat{\lambda}$ of the optimal dual variable. To do that, we need to establish a small threshold $\xi > 0$ (the need of such threshold might not be evident at this point, but the fact that it isn't possible to know the "optimal" dual variable with total accuracy together with a look at Fig. 3 might suggest that ξ is really needed). This way, we define the set $\Omega = \{i : |a_i^T\hat{\lambda}| > 1 - \xi\}$, where $\hat{\lambda}$ is the dual variable returned the subgradient method. Note that we have, in general, $\lambda^* \neq \hat{\lambda}$. Consequently, matrix M is defined by $M = A|_\Omega$. We also define the set Ω^* as being $\{i : x_i^* \neq 0\}$, where x^* is any optimal solution of (BP). Actually, we can only find an optimal solution of (BP), using the method that we are developing, if there exists an optimal set Ω^* such that $\Omega^* \subset \Omega$. When this condition doesn't hold, neither we can find an optimal solution from (RBP), nor we can guarantee that the constraint set of (RBP) is feasible. On the other hand, when we have $\Omega^* \subset \Omega$ for any optimal set, the linear system $Mu = b$ might even have a unique solution.

We assume, from now on, that an integer L such that

$$n' \leq L \quad \Longrightarrow \quad \delta_{n'}(A) < 1$$

is previously known, where n' is the cardinality of the set Ω, *i.e.*, $n' = |\Omega|$. Notice that when no information about the matrix A is available, we can always set $L = 1$.

Now, we are in conditions to define a procedure to try to find an optimal solution of (BP). It takes the set Ω, the matrix M, the vector b and the integer L as inputs.

[2] We say that a function $f : \mathbb{R}^n \to \mathbb{R}$ is *coercive* if $\lim_{\|x\| \to +\infty} f(x) = +\infty$.

Procedure 1 (Calculate $\hat{x}$)**.**

Step 1 *Solve the least–squares problem*

$$q = \min_{u} \|Mu - b\|,$$

and designate the found solution by u'.

Step 2 *If* $q > 0$ *or* $n' \leq L$*, make* $\hat{u} = u'$ *and go to step* 4.

Step 3 *Else* ($q = 0$)*, solve the RBP*

$$\min_{Mu=b} \|u\|_1,$$

and designate the found solution by $\hat{u}$.

Step 4 *Form* $\hat{x} \in \mathbb{R}^n$ *by making* $\hat{x}|_\Omega = \hat{u}$ *and equating all the other entries to zero.*

We enunciate an important theorem about this procedure.

Theorem 1. *Procedure 1 returns* $\hat{x}$ *equal to an optimal solution* x^* *if and only if an optimal set* Ω^* *is contained in* Ω, i.e.

$$\exists_{\Omega^*} : \Omega^* \subset \Omega \qquad \Longleftrightarrow \qquad \hat{x} = x^*,$$

where Ω^* *is any optimal set that doesn't necessarily correspond to* x^*.

Note that steps 1 and 2 of procedure 1 allowed us to overcome the questions of the infeasibility of the linear system $Mu = b$ and the uniqueness of this same linear system (if that linear system has a unique solution, we don't need to solve the RBP).

2.1.2. Resulting Algorithm And Simulations

From what we have seen above, it is clear that the algorithm has two phases: first, we calculate a dual variable $\hat{\lambda}$, that we expect to be close to the optimal one. Then, after selecting the "important" columns of A, we run procedure 1. In the first phase, in order to calculate a subgradient g^k, at the iteration k of the subgradient method, we need to gather the products $A_p\tilde{x}_p(\lambda^k)$ from all the processors. This suggests the need of a central processor. The architecture for the links required to execute the following algorithm is plotted in Fig. 2

Algorithm 1 (SMBBP)**.**

- Procedure (for central processor):
  - *Receive* b *(from outside);*
  - *Iterate on* k *[subgradient method cycle]*
    1. §1 *Send* λ^k *to each processor* p*;*
    2. §1 *Receive* $A_p\tilde{x}_p(\lambda^k)$ *from each processor;*
    3. *Compute* $g^k = \sum_{p=1}^{P} A_p\tilde{x}_p(\lambda^k) - b$*;*
    4. $\lambda^{k+1} = \lambda^k - \alpha^k g^k$*, for a known step size* α^k*;*
  - $\hat{\lambda} = \lambda^k$*;*
  - §2 *Send* $\hat{\lambda}$ *and* ξ *to each processor* $p = 1, \ldots, P$.
  - §2 *Receive* Ω_p *and* $A_p|_{\Omega_p}$ *from each processor.*
  - *Form the matrix* $M = \left[A_1|_{\Omega_1} \cdots A_P|_{\Omega_P}\right]$*, and run* ***procedure 1****.*
- Procedure (for each processor $p = 1, \ldots, P$)
  There are two modes (referenced above as §1 and §2):
  - *Mode 1:*
    1. *Receive* λ^k *and compute* $c = A_p^T\lambda^k$*;*
    2. *Set* $x^j = 0$ *if* $|c^j| < 1$*, and* $x^j = R \cdot \mathsf{sign}(c^j)$ *if* $|c^j| \geq 1$*, for* $j = 1, \ldots, n_p$*;*
    3. *Return* $A_p\tilde{x}_p(\lambda^k)$*, where* $\tilde{x}_p(\lambda^k) = \left(x^1, \ldots, x^{n_p}\right)$.
  - *Mode 2:*
    1. *Receive* $\hat{\lambda}$ *and* ξ*; and compute* $c = A_p^T\hat{\lambda}$*;*
    2. *Form the set* $\Omega_p = \{j : |a_p^{j\,T}\lambda^k| > 1 - \xi\}$ *and return* Ω_p *and* $A_p|_{\Omega_p}$.

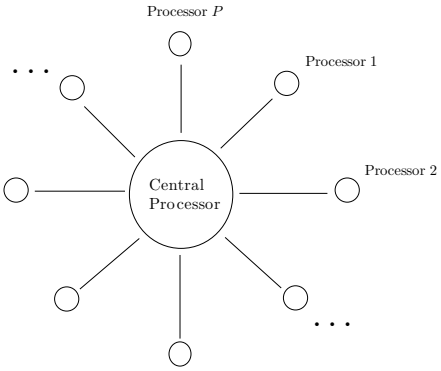


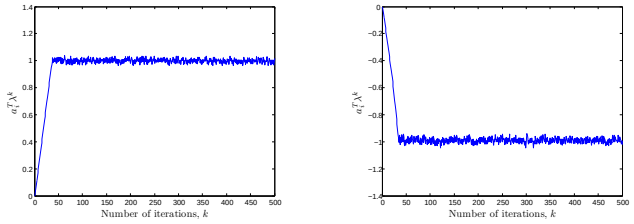
Fig. 2. Minimal architecture of the links between the processors for the implementation of the SMBBP.

Note that theorem 1 guarantees that algorithm 1 returns an optimal solution x^* whenever we have $\Omega^* \subset \Omega$. However, this condition isn't easy to verify. Fig. 3 shows the typical behavior of the inner products $a_i^T\lambda^k$ over the iterations of the subgradient method cycle. The ripple found there is a consequence of the *bang–bang* solution we adopted for $\tilde{x}_p^j(\lambda)$ in (5). That ripple justifies partially the need of a parameter ξ.

However, a moderate value for ξ isn't enough to guarantee that $\Omega^* \subset \Omega$. Indeed, there are some columns a_i for which $i \in \Omega^*$ and for which the inner product $a_i^T\lambda^k$ takes many iterations to converge to ± 1. Figure 4 shows an example of such column.

2.2. Multiplier Methods

In order to avoid the polyhedral constraints that we have talked about, we will transform the dual Lagrangian of (BP)



(a) $x_i^* > 0$. (b) $x_i^* < 0$.

Fig. 3. Typical behavior of the inner product $a_i^T \lambda^k$ along 500 iterations of algorithm 1, in which a_i is a column of A associated with a non-zero coefficient of x^*.

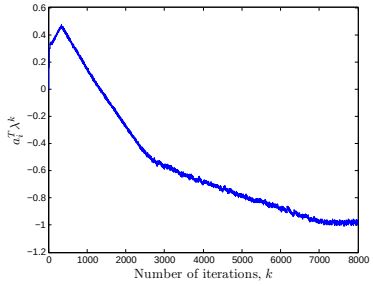


Fig. 4. Inner product $a_i^T \lambda^k$ along the iterations of the SMBBP, where a_i a column activated by an optimal solution x^*, and for which many iterations are needed to converge to -1. Note the difference in the number of iterations between this plot and Fig. 3.

into a coercive function. First, note that (BP) is equivalent to

$$\min_{Ax=b} \|x\|_1 + \frac{\rho}{2}\|b - Ax\|^2, \tag{6}$$

where ρ is a positive scalar parameter. If we calculate the Lagrangian of (6) we get, by definition, the *augmented Lagrangian* [3, 2] of the original problem, (BP). Considering again the decomposition of the variable x into $(x_1, \ldots, x_P)$, the augmented Lagrangian of (BP) is

$$L_a(x_1, \ldots, x_P, \lambda) = \lambda^T b + \sum_{p=1}^{P} (\|x_p\|_1 - \lambda^T A_p x_p) + \frac{\rho}{2}\|b - \sum_{p=1}^{P} A_p x_p\|^2. \tag{7}$$

Note that (7) is now coercive with respect to the primal variable x. There is a method, known as the *method of multpliers* [3, 2], that permits us to find simultaneously the optimal primal and dual variables, only from successive minimizations of the augmented Lagrangian. That algorithm operates in the following way: starting with an arbitrary $\lambda^0 \in \mathbb{R}^m$ and $\rho^0 \in \mathbb{R}$, for each iteration k, we fix λ^k and find an x^k that minimizes $L_a(x, \lambda^k)$ with respect to (w.r.t.) the primal variable x. Then we update the dual variable $\lambda^{k+1} = \lambda^k + \rho^k(b - Ax^k)$ and the parameter of the augmented Lagrangian: $\rho^{k+1} = c\rho^k$, where c is a real number greater or equal to 1.

This method is well–defined for coercive augmented Lagrangians, since there is always a minimizer of $L_a(x, \lambda^k)$ w.r.t. x, for any λ^k.

However, we still haven't figured out how to find the optimal primal variable x^* using the P available processors. In fact, they are used in the minimization of $L_a(x, \lambda^k)$. Yet, there is still one problem: the augmented Lagrangian $L_a(x, \lambda^k)$ (7) isn't separable into P independent function due to a quadratic term that binds all the variables x_p one to each other. We are going to see two methods that can overcome this problem.

2.2.1. Diagonal Quadratic Approximation (DQA)

Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be a convex and coercive function. Decompose a variable $x \in \mathbb{R}^n$ into P subvariables in $\mathbb{R}^{n_p}$, with $n_1 + \ldots + n_P = n$. Define, for each $p = 1, 2, \ldots, P$, the function

$$f_p(x, y_p) = f(x_1, \ldots, x_{p-1}, y_p, x_{p+1}, \ldots, x_P).$$

Then, the DQA algorithm consists on:

Algorithm 2 (DQA)**.**

Step 1 *Find*

$$y_p^t \in \arg\min_{y_p \in \mathbb{R}^{n_p}} f_p(x^t, y_p),$$

for all $p = 1, \ldots, P$.

Step 2 *Form the vector* $y^t = (y_1^t, y_2^t, \ldots, y_P^t)$.

Step 3 *Update* x^t,

$$x^{t+1} = x^t + \tau(y^t - x^t). \tag{8}$$

Step 4 $t \leftarrow t + 1$*; and return to Step 1.*

In the algorithm, $x^0 \in \mathbb{R}^n$ is an arbitrary vector, and the variable τ in (8) is any fixed real number between 0 (excluded) and P (included).

The DQA is known to converge if applied to differentiable functions [13]. However, if we want to apply it to the minimization of (7), the literature doesn't contain, to the best of our knowledge, any result that guarantees the convergence of the DQA for this function. Nonetheless, it is possible to extend the proof of convergence of the DQA for differentiable functions to cover the case of *rigid functions*, which we define next.

Definition 2 (Rigid Function)**.** *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be a subdifferentiable and continuous function over* $\mathbb{R}^n$*. We say that* f *is a* rigid function *if, for any point* $x^\star \in \mathbb{R}^n$*, we have*

$$0 \in \arg\min_h f(x^\star + he_i),\ \forall_{i=1,\ldots,n} \quad \Longrightarrow \quad 0 \in \partial f(x^\star),$$

where e_i *is a canonical direction in* $\mathbb{R}^n$.

Note that the differentiable functions are all rigid, but there are subdifferentiable functions that are non–rigid [2]. In any case, the following lemma is valid.

Lemma 1. *Every function $f : \mathbb{R}^n \to \mathbb{R}$ of the type $f(x) = g(x)+\|x\|_1$, where $g : \mathbb{R}^n \to \mathbb{R}$ is a convex and differentiable function, is a rigid function.*

This lemma guarantees that (7) is a rigid function. Furthermore, the convergence of the DQA algorithm applied to (7) is guaranteed by

Theorem 2 (Convergence of DQA)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex, rigid and coercive function over $\mathbb{R}^n$. Admit the decomposition of a variable $x \in \mathbb{R}^n$ into $(x_1, x_2, \ldots, x_P)$, where $x_p \in \mathbb{R}^{n_p}$, for $p = 1, 2, \ldots, P$, with $n = n_1 + n_2 + \ldots + n_P$. Then,*

1. *Every accumulation point of the sequence $\{x^t\}$ generated by the DQA algorithm solves $\min_{x\in\mathbb{R}^n} f(x)$;*
2. *The sequence of objective values converges to the optimum,* i.e.*, $f(x^t) \downarrow f^* := \min_{x\in\mathbb{R}^n} f(x)$.*

It can be shown that if we fix all the block variables x_j^t, for $j \neq p$ at a fixed iteration t of the DQA, and we minimize the augmented Lagrangian (7) w.r.t. x_p, we get a problem equivalent to

$$\min_{y_p} \frac{1}{2}\|A_p y_p - \gamma_p^{t,k}\|^2 + \frac{1}{\rho^k}\|y_p\|_1, \tag{9}$$

where $\gamma_p^{t,k} = b - \sum_{j\neq p} A_j x_j^t + \lambda^k/\rho^k$. In fact, (9) is a BPDN problem. This is the main problem that each processor p has to solve in each iteration of the DQA. Efficient algorithms that solve a BPDN rapidly are described in [9, 10].

Chaining the method of multipliers with the DQA, we get the following algorithm, which uses the architecture for the links depicted in Fig. 2.

Algorithm 3 (M/DQA)**.**

- *Procedure (for central processor):*
  - *Receive b (from outside);*
  - *Iterate on k* [Method of multipliers]*:*
    - *Iterate on t* [DQA]*:*
      1. §1 *Send x_p^t to each processor;*
      2. §1 *Receive $A_p x_p^t$ from each processor;*
      3. §2 *Send $\gamma_p^{t,k} = b - \sum_{j\neq p} A_p x_p^t + \lambda^k/\rho^k$ to each processor;*
      4. §2 *Collect y_p from each processor and form $y = (y_1, y_2, \ldots, y_P)$;*
      5. *$x^{t+1} = x^t + \tau\,(y - x^t)$; and $t \leftarrow t + 1$.*
    - *Make $x^k = x^t$ for the last t;*
    - *Evaluate the sum $Ax^k = \sum_{p=1}^P A_p x_p^k$;*
    - *$\lambda^{k+1} = \lambda^k + \rho^k\left(b - Ax^k\right)$; $\rho^{k+1} = c\rho^k$; and $k \leftarrow k + 1$.*
- *Procedure (for each processor $p = 1, \ldots, P$): There are two modes (referenced above as §1 and §2):*
  - *Mode 1:*
    1. *Receive x_p^t and return the product $A_p x_p^t$.*
  - *Mode 2:*
    1. *Receive $\gamma_p^{t,k}$;*
    2. *Solve $y_p = \arg\min_{w_p}(1/2)\|A_p w_p - \gamma_p^{t,k}\|^2 + (1/\rho^k)\|w_p\|_1$, and return y_p.*

In fact, a central node is needed because the variables $\gamma_p^{t,k}$ have to be calculated for all $p = 1, \ldots, P$ at the same iteration t of the DQA cycle.

2.2.2. *Nonlinear Gauss–Seidel (NGS)*

The NGS is a more efficient variation of a kind of algorithms that include the DQA (Nonlinear Jacobi algorithms). In the NGS, the calculation of the each new variable x_p^{t+1} is done in a sequential order (which we assume here to be a circular one: $1, 2, \ldots, P, 1 \ldots$), as opposed to what happens in the DQA (all the new variables are calculated at the same time). This fact allows us to use the link architecture described in Fig. 5.

However, to use the NGS here, we need to make a restriction on the partition of the matrix A: we have to assume that the partition is such that each block matrix A_p has full column–rank.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex and coercive function over $\mathbb{R}^n$. Then, the NGS consists on:

Algorithm 4 (NGS)**.**

Step 1 *For all $p = 1, 2, \ldots, P$, find*

$$x_p^{t+1} \in \arg\min_{x_p\in\mathbb{R}^{n_p}} f(x_1^{t+1}, \ldots, x_{p-1}^{t+1}, x_p, x_{p+1}^t, \ldots, x_P^t).$$

Step 2 *$t \leftarrow t + 1$; and return to Step 1.*

As happens with the DQA, the known results about the convergence of the NGS only apply to differentiable functions. However, we claim that those results remain valid for rigid functions.

Theorem 3 (Convergence of the NGS)**.** *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is a convex, rigid and coercive function over $\mathbb{R}^n$. Further, assume that f has unique block coordinate minimizers, that is, for any $x = (x_1, x_2, \ldots, x_P) \in \mathbb{R}^n$, the optimization problem*

$$\min_{w_p\in\mathbb{R}^{n_p}} f(x_1, \ldots, x_{p-1}, w_p, x_{p+1}, \ldots, x_P)$$

has a unique solution. Then, every limit point of the sequence $\{x^t\}$ generated by the NGS solves $\min_{x\in\mathbb{R}^n} f(x)$.

For the theorem 3 be applicable to (7), we need to ensure that L_a has unique block coordinate minimizers for any fixed λ. Indeed, this can be proved if the assumption that each A_p has full column–rank holds.

Consider now any processor p. Fix all the variables x_i, for $i < p$, at the iteration $t+1$ of the NGS and fix all the variables i, for $i > p$, at the iteration t of the NGS. Then, if we minimize the augmented Lagrangian (7) w.r.t. x_p we get a problem equivalent to

$$\min_{x_p \in \mathbb{R}^{n_p}} \frac{1}{2}\|A_p x_p - \gamma_p^{t,k}\|^2 + \frac{1}{\rho^k}\|x_p\|_1,$$

where $\gamma_p^{t,k} = b - \sum_{i<p} A_i x_i^{t+1} - \sum_{i>p} A_i x_i^t + \lambda^k/\rho^k$. This variable, being different from the one defined in subsection 2.2.1, can be updated from processor $p-1$ to processor p as follows: if the processor $p-1$ sends the vector $\phi_{p-1} = \gamma_{p-1}^{t,k} - A_{p-1}x_{p-1}^{t+1}$ to the processor p, this processor can correctly find $\gamma_p^{t,k}$ by doing $\gamma_p^{t,k} = \phi_{p-1} + A_p x_p^t$, where x_p^t had been calculated in the previous iteration.

The following algorithm results from chaining the method of multipliers with NGS, and operates upon the architecture described in Fig. 5.

Algorithm 5 (M/GS)**.**

- *Iterate on* k [Method of Multipliers]*:*

  Procedure (for processor $p = 1, \ldots, P$**):**

  1. *Receive* ρ^k *and* ϕ_{p-1} *from processor* $p-1$*;*
  2. *Set* $\gamma_p^{t,k} = \phi_{p-1} + A_p x_p^t$*;*
  3. *Solve* $x_p^{t+1} = \arg\min_{x_p} \frac{1}{2}\|A_p x_p - \gamma_p^{t,k}\|^2 + \frac{1}{\rho^k}\|x_p\|_1$*;*
  4. *Keep* x_p^{t+1} *in memory;*
  5. *Send* ρ^k *and* $\phi_p = \gamma_p^{t,k} - A_p x_p^{t+1}$ *to processor* $p+1$.

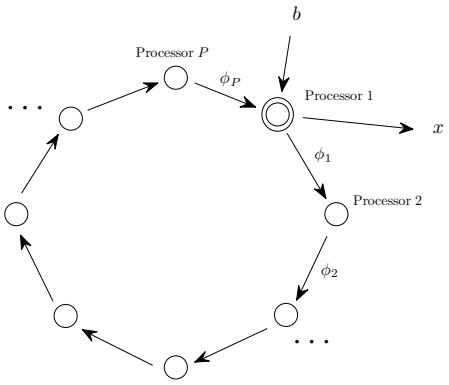


Fig. 5. Required architecture for the implementation of the algorithm M/GS.

2.3. Comparison Of The Methods: Horizontal Partition

Table 1 presents the main theoretical differences, that weren't emphasized during the text, between the algorithms that solve the (BP) for an horizontal partition of the matrix A. We now explain the concepts in that table:

Degree of parallelism Let φ be the number of distributed processors that execute the same task at the same time (in a typical iteration: no stopping criterion has been achieved), for a given algorithm which requires distributed processors that are equal and that take no decisions (no "if" clauses). Then, we define

$$\text{Degree of parallelism} = \lim_{P \to +\infty} \frac{\varphi}{P}.$$

This measure translates the gain in time we would have if we increased the number of processors P.

Complexity of a processor We say that a processor is complex (and represent it by $+$) if it has to do any kind of internal minimizations, implying an internal cycle. Whereas a simple processor ($-$) is characterized by doing only simple operations (sums, multiplications,...).

Robustness to instantaneous link failures Table 1 includes the comparison of the algorithms in this aspect, because the M/GS is the only algorithm that has this feature. However, that isn't evident just from algorithm 5. That fact follows from a generalization that can be done to theorem 3. This theorem remains valid if the minimizations are carried in an arbitrary order, as long as there exists an M such that at the end of every M iterations all the minimizations (in each block variable) have been done at least once. As so, if the communications between the processors are done randomly, then the M/GS becomes robust to instantaneous link failures.

Size of the vectors exchanged by iteration This row tells the size of the vectors exchanged between the processors for each (inner) iteration of an algorithm as a function of the dimensions of the matrix $A \in \mathbb{R}^{m \times n}$.

Table 1. Theoretical features of the algorithms that use an horizontal partition.

	SMBBP	M/DQA	M/GS
Degree of parallelism	1	1	0
Complexity of distributed processors	$-$	$+$	$+$
Complexity of central processor	$+$	$-$	
Robustness to instantaneous link failures	No	No	Yes
Size of vectors exchanged by iteration	$2(m+1)$	$2(m+n)$	$m+1$

Table 2 shows experimental results for all algorithms that we have presented. The first two rows show the average error in the variable, x, and in the cost function, $f(x)$, respectively. We considered that the solution given by the linear program

formulation of (BP) is the correct one. As the algorithms M/DQA and M/GS have the same "internal structure", we put the same stopping criteria for both. After solving them, we solved the SMBBP making its subgradient method cycle stop when the dual cost function, given by $-H(\lambda)$ (see (4)), hits 99% of the value of the cost function that M/DQA and M/GS returned.

We must say that in 50% of the experiences the SMBBP returned the exact optimal solution (which means $\Omega^* \subset \Omega$), while the other algorithms didn't return it with zero error once.

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 can be done in parallel (note that these simulations were done in a single computer), and the estimated time for the execution of the algorithms if we really had P processors available. Let t_T and t_p designate, respectively, the total time of an algorithm and the time spent in parallelizable tasks. Then, the estimated time can be calculated by $t_{\text{est}} = t_T + ((1 - P)/P)t_p$. Note that this estimation only applies to SMBBP and M/DQA, since the M/GS has a null degree of parallelism.

The rows that show the inner and the external iterations represent the average values of these quantities for each algorithm (note that the SMBBP only has one cycle, thus has no internal iterations).

The percentage of used columns is the average of the number of the used columns that served to form the matrix M, before running procedure 1.

Table 2. Experimental results from the simulation of the SMBBP, the M/DQA and M/GS. The numbers presented are the average of 10 random experiments in a simulation of a distributed environment with $P = 10$ processors, each with 25 columns of a matrix $A \in \mathbb{R}^{50\times 250}$.

	SMBBP	M/DQA	M/GS
Error in x	3.11×10^{-2}	5.54×10^{-3}	8.58×10^{-3}
Error in $f(x)$	1.40×10^{-2}	1.72×10^{-3}	1.07×10^{-3}
Total Time [s]	2.28	1.03×10^{2}	6.58
Time in Parallel [s]	1.72	1.02×10^{2}	
Estimated Time [s]	7.32×10^{-1}	1.20×10^{1}	6.58
Inner Iterations		3.52×10^{2}	4.71×10^{2}
External Iterations	2.47×10^{3}	9.00	7.56
Percentage Of Used Columns	15.1		

At this point, it is worth to say that the SMBBP is the algorithm more adequate to applications where accuracy is more important than the time spent in solving the BP. Though this is not evident from table 2, we know that if we increase the number of the iterations of the subgradient method in SMBBP or even just increase the parameter ξ, we will have, with great probability, the condition $\Omega^* \subset \Omega$. This means that the SMBBP will return the exact solution of the BP (recall theorem 1).

On the other hand, the M/GS is the more adequate to applications where the time is more important than accuracy. The trade–off between these two quantities lies in the stopping criteria of the algorithm.

3. DISTRIBUTED BP WITH VERTICAL PARTITION

In this section, we will propose an algorithm that solves the (BP), but adapted to a vertical partition of the matrix $A \in \mathbb{R}^{m\times n}$:

$$A = \underbrace{\left[\begin{array}{c} A_1 \\ \vdots \\ A_p \\ \vdots \\ A_P \end{array}\right]}_{n} \Bigg\} m \quad ,$$

where each submatrix $A_p \in \mathbb{R}^{m_p} \times \mathbb{R}^n$ is stored in the processor p, for $p = 1, 2, \ldots, P$ and with $m = m_1 + m_2 + \ldots + m_P$. Also, we use the same partition for the vector $b \in \mathbb{R}^m$:

$$b = \left[\begin{array}{c} b_1 \\ \vdots \\ b_p \\ \vdots \\ b_P \end{array}\right],$$

where $b_p \in \mathbb{R}^{m_p}$ for $p = 1, 2, \ldots, P$. Without loss of generality, we can assume that each submatrix A_p is full–rank, *i.e.*, its rows are linearly independent.

This kind of partition makes sense when, for example, we are measuring a phenomenon x using a network of sensors: each measure $b_i \in \mathbb{R}$ is done by a sensor, and they all have to cooperate in order to find x, without exchanging information about their sensing devices (each row of the matrix A).

3.1. Theory Behind The Algorithm

Using the above definitions, we can write (BP) as

$$\begin{array}{c} \min \\ A_p x = b_p, \quad p = 1, \ldots, P \\ \text{var} : x \in \mathbb{R}^n \end{array} \quad \sum_{p=1}^{P} \frac{1}{P}\|x\|_1,$$

and cloning the variable x into $x_1, x_2, \ldots, x_P$, this problem is equivalent to

$$\begin{array}{c} \min \\ A_p x_p = b_p, \quad p = 1, \ldots, P \\ x_p = x_{p+1}, \quad p = 1, \ldots, P-1 \end{array} \quad \sum_{p=1}^{P} \frac{1}{P}\|x_p\|_1, \qquad (10)$$

where the variable is now $(x_1, \ldots, x_P)$, belonging each subvariable x_p to $\mathbb{R}^n$. We will solve problem (10) through its

dual, by dualizing only the last $P-1$ equalities of its constraints, and using the augmented Lagrangian (to transform the objective of (10) into a coercive function).

Designating the dual variables by λ_p for $p = 1, \ldots, P-1$, the augmented Lagrangian L_a of (10) is

$$L_a(x_1, \ldots, x_P, \lambda_1, \ldots, \lambda_{P-1}) = \sum_{p=1}^{P} \left[\frac{1}{P}\|x_p\|_1 + (\lambda_p - \lambda_{p-1})^T x_p \right] + \rho\|x_1 - x_2\|^2 + \ldots + \rho\|x_{P-1} - x_P\|^2, \quad (11)$$

where, by definition, $\lambda_0 = \lambda_P = 0_n$ (0_n is the zero vector in $\mathbb{R}^n$).

From the coercivity and convexity of the ℓ_1-norm, it follows that the method of multipliers is well–defined for problem (10) (and proved to converge [3, 2]). Note that the application of the method of multipliers to this problem leads to the minimization of (11) w.r.t. $(x_1, \ldots, x_P)$ for fixed dual variables $(\lambda_1^k, \ldots, \lambda_{P-1}^k)$, at each iteration. We will designate the minimizers of this problem by $(x_1^k, \ldots, x_P^k)$. Also, the dual variables are updated as follows: $\lambda_p^{k+1} = \lambda_p^k + \rho^k(x_p^k - x_{p+1}^k)$, for $p = 1, \ldots, P-1$.

However, we return to the same problem that we had in subsection 2.2: the minimization of (11) w.r.t. $(x_1, \ldots, x_P)$ isn't separable into P independent minimization problems. This is due to the existence of the quadratic coupling terms in (11).

It can be proved that (11) is convex, coercive and rigid on the primal variables, when the dual variables λ_p^k are fixed, for $p = 1, \ldots, P-1$. Furthermore, (11) has unique block coordinate minimizers, as it is strictly convex on each block variable x_p, for $p = 1, \ldots, P$. Consequently, theorems 2 and 3 hold for the minimization of (11).

In order to minimize (11), we can apply both the DQA or the NGS. Though, we will just explore the use of the latter for two reasons: the NGS is known to have better convergence properties than the DQA, namely the speed of convergence is higher; the minimum requirements for the architecture of the links, opposed to what happened in subsection 2.2, is the same, whatever it is the algorithm that we use.

3.1.1. NGS Approach

Indexing each iteration of the NGS (algorithm 4) with the letter t, it can be shown that the application of this algorithm to the minimization of (11) leads to a problem equivalent to

$$\min_{A_p y_p = b_p} \frac{1}{P}\|y_p\|_1 + \left[\lambda_p^k - \lambda_{p-1}^k - 2\rho^k(x_{p-1}^{t+1} + x_{p+1}^t)\right]^T y_p + 2\rho^k\|y_p\|^2, \quad (12)$$

for each processor $p = 1, \ldots, P$, where the variable is $y_p \in \mathbb{R}^n$. Note that (12) can be recast as a quadratic program.

Due to the nature of the variables involved in each problem (12), we can see that the processor p only needs, in order to calculate a new variable x_p^{t+1}, the variables x_{p-1}^{t+1} and x_{p+1}^t. Furthermore, in the possession of these variables, processor p can update, not only the dual variable λ_p^k, but also λ_{p-1}^k, whenever each subproblem (NGS) converges. As so, this algorithm can operate upon the architecture defined in Fig. 6.

3.2. Resulting Algorithm And Simulations

The following algorithm summarizes what we have seen in this section.

Algorithm 6 (M/GS for VP)**.**

- *Iterate on* k [Method of Multipliers]*:*

  Procedure (for the processor $p = 1, \ldots, P$**):**

  - *If it receives* $x_{p-1}^{t_p+1}$ *and* $x_{p+1}^{t_p}$*,*
    1. *Form the vector* $\tau_p = \lambda_p^k - \lambda_{p-1}^k - 2\rho^k(x_{p-1}^{t_p+1} + x_{p+1}^{t_p})$*;*
    2. *Solve* $x_p^{t_p+1} = \arg\min_{A_p x_p = b_p}(1/P)\|x_p\|_1 + \tau_p^T x_p + 2\rho^k\|x_p\|^2$*;*
    3. *Send* $x_1^{t_p+1}$ *to processors* $p-1$ *and* $p+1$*;*
    4. *Make* $t_p \leftarrow t_p + 1$.
  - *else, exchange information about its convergence (both of the NGS and the method of multipliers) with the neighbors, in the idle times of the links.*

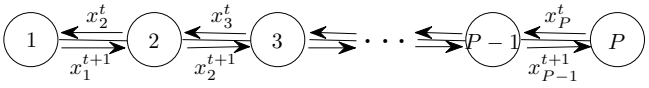


Fig. 6. Architecture needed for the implementation of algorithm 6; and illustration of the flow of information in its inner cycle (NGS step).

Although algorithm 6 is written for a generic processor p, we must take into account that for the extremal processors, 1 and P, all kind of interaction of these processors with the processors $p-1$ and $p+1$, respectively, should be ignored.

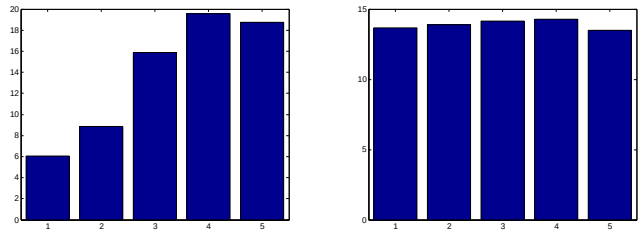
We need to say that neither the stopping criteria, nor the actions that each processor takes when a stopping criterion is verified, are included in algorithm 6. For example, when the NGS converges, each processor must update the variables λ_p^k and λ_{p-1}^k, as well as ρ^k.

A detailed analysis of algorithm 6 shows that the links between the processors are occupied only 50% of the time (after $P-1$ iterations). As a consequence, the other 50% can be used by the processors to exchange information about the achieved convergence of either the method of multipliers or the NGS. The characteristics of the algorithm 6 are in table 3.

Table 3. Theoretical features of algorithm 6.

Degree of parallelism	$1/2$
Complexity of distributed processors	$+$
Robustness to instantaneous link failures	Yes
Size of vectors exchanged by iteration	$2n$

A curious aspect of this algorithm, though, is that before the processors update their dual variables λ_p^k, they need to get to a "small consensus" first, independently of the value of the variable ρ^k. This is evident from Fig. 7, where the variables x_p^0, for five processors, were initialized with very different vectors (Fig. 7(a)) and only after seven iterations of the inner cycle of algorithm 6 they were almost equal, although very different from the optimal solution (Fig. 7(b)). We can see then, that the inner cycle is responsible for not keeping the variables very different from each other, and the external cycle is responsible for "pushing" the variables of each processor to near the optimal solution.



(a) Initial iteration. (b) 7th iteration of the NGS cycle.

Fig. 7. Histograms of the errors of each variable x_p, for $p = 1, \ldots, 5$, during the execution of algorithm 6. The variables were initialized in the following way: to x_1^0 was assigned a random vector of norm equal to 10; then, we made $x_5^0 = 2x_4^0 = 2x_3^0 = 2x_2^0 = 2x_1^0$. The histograms show the distance of each variable to the optimal one, x^*.

3.3. Application To The BPDN

If, instead of the (BP), we had the (BPDN) problem, we could also apply algorithm 6 with minor modifications to solve it. Also, the guarantees of convergence for the method of multipliers, for the DQA and for the NGS would be maintained.

4. REFERENCES

[1] R. Baraniuk, J. Romberg, and M. Wakin. Tutorial on compressive sampling. February 2008. Available at the Rice University Webpage.

[2] D. Bertesekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997. Available at `http://web.mit.edu/dimitrib/www/pdc.html`.

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

[4] S. Boyd and A. Mutapic. Subgradient methods, notes for ee364b. Stanford University, available at `http://www.stanford.edu/class/ee364b/notes/subgrad_method_notes.pdf`, 23 January 2007.

[5] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. Also available at `http://www.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf`.

[6] E.J. Candès. The restricted isometry property and its implications for compressed sensing. Preprint submitted to the Académie des sciences, February 2008.

[7] E.J. Candès and T. Tao. Near–optimal signal recovery from random projections and universal encoding strategies. *submitted to IEEE Trans. Inform. Theory*, November 2004. Available on the ArXiV preprint server: `math.CA/0410542`.

[8] E.J. Candès and T. Tao. Decoding by linear programming. *IEEE Trans. Inform. Theory*, 51(12):4203–4215, December 2005.

[9] M.A.T. Figueiredo, R.D. Nowak, and S.J. Wright. Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. To Appear in the IEEE Journal of Selected Topics in Signal Processing, 2007.

[10] S.J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large scale $l1$-regularized least squares. To Appear in IEEE Journal on Selected Topics in Signal Processing.

[11] J.A. Tropp. Just relax: Convex programming methods for identifying sparse signals. *IEEE Transactions on information theory*, 51(3):1030–1051, March 2006.

[12] Rice university webpage. Compressive sensing resources. `http://www.dsp.ece.rice.edu/cs/`.

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