BASIS PURSUIT IN SENSOR NETWORKS

João F. C. Mota[1,2], *João M. F. Xavier*[2], *Pedro M. Q. Aguiar*[2], *and Markus Püschel*[3]

[1] Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, USA
[2] Institute of Systems and Robotics, Instituto Superior Técnico, Lisbon, Portugal
[3] Department of Computer Science, ETH Zurich, Switzerland

ABSTRACT

Basis Pursuit (BP) finds a minimum ℓ_1-norm vector z that satisfies the underdetermined linear system $Mz = b$, where the matrix M and vector b are given. Lately, BP has attracted attention because of its application in compressed sensing, where it is used to reconstruct signals by finding the sparsest solutions of linear systems. In this paper, we propose a distributed algorithm to solve BP. This means no central node is used for the processing and no node has access to all the data: the rows of M and the vector b are distributed over a set of interconnected compute nodes. A typical scenario is a sensor network. The novelty of our method is in using an optimal first-order method to solve an augmented Lagrangian-based reformulation of BP. We implemented our algorithm in a computer cluster, and show that it can solve problems that are too large to be stored in and processed by a single node.

***Index Terms*—** Convex optimization, basis pursuit, distributed algorithm, sensor network, compressed sensing

1. INTRODUCTION

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

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

where the optimization variable is $z \in \mathbb{R}^q$, $\|z\|_1 = |z_1| + \cdots + |z_q|$ is the ℓ_1 norm of the vector z, and $M \in \mathbb{R}^{m\times q}$ is a matrix with more columns than rows: $m < q$. Since the underdetermined linear system $Mz = b$ has potentially many solutions, BP finds the smallest one in the ℓ_1 norm sense. The following assumption guarantees that $Mz = b$, and thus (BP), is always solvable.

Assumption 1. *M is full rank.*

There is a large body of literature exploring BP as the relaxation of the nonconvex problem obtained by replacing the ℓ_1 norm in (BP) by the ℓ_0 pseudo norm $\|z\|_0$, which is the number of nonzero elements of z. The conditions that ensure an *exact* relaxation underlie the success of compressed sensing (CS) [2]. BP usually arises in CS with a matrix M whose entries are drawn independently from some probability distribution. In this context, Assumption 1 holds with probability one whenever that distribution is non-degenerate.

Related work. Very recently, several algorithms have been proposed to solve BP such as [3]. These approaches, however, cannot be easily adapted to distributed scenarios. On the other hand, since BP can be recast as a linear program (LP) [1], any algorithm solving LPs also solves BP. However, solving an LP in a distributed way is also nontrivial. For example, [4] proposes a distributed method for solving LPs, using a simplex-based approach. However, that method requires the network to be a complete graph, i.e., there is a link between any pair of two nodes. The algorithm we introduce in this paper only requires the graph to be connected.

A problem closely related to BP is Basis Pursuit Denoising (BPDN) [1]. BPDN was solved in a distributed way in [5], using well-known optimization techniques similar to the ones we apply in this paper. Specifically, [5] uses a first-order method known as the alternate direction minimization to solve a suitable reformulation of BPDN. In our case, we also use a first-order method, but the nice properties of our dual function enable us to use an optimal first-order method: Nesterov's first method [6]. The application of Nesterov's method to our problem is possible because we assume the network does not vary along time, i.e., there are no link failures during the execution of the algorithm. The case where the network topology may vary along time has been studied before, e.g., in [7], which proposes an algorithm to solve general optimization problems (and thus also BP). However, the method is based on subgradient algorithms, which are known to be much slower than Nesterov's method.

$$M = \begin{bmatrix} M_1 \\ \vdots \\ M_P \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ \vdots \\ b_P \end{bmatrix}$$

Fig. 1. Partition of the matrix $M \in \mathbb{R}^{m\times q}$ and vector $b \in \mathbb{R}^m$.

Contribution: BP in sensor networks. Our goal in this paper is to devise an algorithm that solves (BP) when the given data, M and b, are distributed across a network of P nodes. More specifically, we partition the matrix M, as shown in Fig. 1, into P blocks where each block $M_p \in \mathbb{R}^{m_p\times q}$ contains m_p rows of M, with $m_1 + \cdots + m_P = m$. Vector b is partitioned similarly into P "small" vectors b_p in $\mathbb{R}^{m_p}$. Further, the problem is also solved distributed, i.e., there is no notion of central node for the processing. This scenario can be associated with a network of P nodes: the pth node of the network only has access to M_p and to b_p, and all nodes participate in computing the solution. For the network, we only assume the following.

Assumption 2. *The given network is connected and its topology does not vary along time.*

In summary, we obtain the following problem statement: *Given*

Work supported by the grants SIPM PTDC/EEA-ACR/73749/2006 and SFRH/BD/33520/2008 (Carnegie Mellon/Portugal Program) from FCT, by FCT (ISR/IST plurianual funding) through the PIDDAC Program funds, and by NSF through award 0634967.

a connected network topology of P nodes, each node storing M_p and b_p, solve BP in a distributed way.

The work presented here is a major improvement over our previous work [8], which used a very different reformulation and an algorithm that required a much larger number of iterations.

The main contribution of this paper is two-fold. First, we present a novel distributed algorithm for (BP) by reformulating it to make Nesterov's efficient first method applicable. Second, for proper evaluation, we implemented the algorithm in C/MPI to run, validate, and benchmark it on a distributed computer cluster. As we show, due to its design, the algorithm can solve problems that are too large to be stored in and processed by a single node. An extended version of this paper is available at [9].

Applications. Distributed basis pursuit has several potential applications in sensor network scenarios. For example, several distributed approaches to CS have been proposed, e.g., [10, 11], but in all of them the reconstruction problem, of which BP is an example, has always been solved at a central or fusion node. Using our method, distributed reconstruction becomes possible.

As a concrete application, consider a connected network of P nodes. The goal is to capture an ultra-wide band but spectrally sparse signal $z \in \mathbb{R}^q$, as described in [2]. For simplicity, we assume $m = P$, i.e., each node only stores one row of M. Using a random demodulator [2], each node senses z and obtains a number b_p, for $p = 1, \ldots, P$. This b_p can be represented as the inner product $r_p^\top z$, where $r_p^\top$ is the pth row of M (stored in node p). Applying CS theory, it is possible to recover z from the measurements b_p by solving BP; and doing it in a distributed way falls into our problem statement. Other possible applications include sparse event detection [11] and distributed target localization [12].

2. ALGORITHM

In this section we derive an algorithm to solve our stated problem. We start by suitably reformulating (BP) and then use a well known algorithm to solve the dual problem.

Problem reformulation. BP can be recast as an LP [1]:

$$\begin{array}{ll} \text{minimize} & 1_n^\top x \\ \text{subject to} & Ax = b\,,\ x \geq 0\,, \end{array} \tag{1}$$

where $n = 2q$, $A = [M\ -M] \in \mathbb{R}^{m \times n}$, and 1_n is the vector of ones in $\mathbb{R}^n$. Notice that the knowledge of the $m_p \times q$ block M_p translates into the knowledge of the $m_p \times n$ block A_p. In other words, the pth node has access to the block A_p (we partition A similarly to M) whenever it knows M_p. By making P copies of x (node p stores the copy $x_p \in \mathbb{R}^n$), and by making the partition of A explicit, (1) can be written as

$$\begin{array}{ll} \text{minimize} & \frac{1}{P}\sum_{p=1}^P 1_n^\top x_p \\ \text{subject to} & A_p x_p = b_p\,,\ p = 1, \ldots, P \\ & x_p \geq 0\,,\ p = 1, \ldots, P \\ & x_i = x_j\,,\ (i,j) \in \mathcal{E}\,, \end{array} \tag{2}$$

with $(x_1, \ldots, x_P) \in (\mathbb{R}^n)^P$ as the optimization variable, and $\mathcal{E} := \{(i,j) :$ there exists a link between nodes i and j and $i < j\}$ as the set of edges of the network. Assumption 2 ensures that problems (1) and (2) are equivalent.

Dual problem. We propose to solve a dual problem of (2). The dualization strategy we adopt is to dualize each constraint $x_i = x_j$ in the augmented Lagrangian sense. This means that we associate the Lagrange multiplier λ_{ij} to this constraint and take the augmented Lagrangian

$$\begin{aligned} L(x_1, \ldots, x_P; \lambda) = \frac{1}{P}\sum_{p=1}^P 1_n^\top x_p + \sum_{(i,j)\in\mathcal{E}} \lambda_{ij}^\top (x_i - x_j) \\ + \sum_{(i,j)\in\mathcal{E}} \frac{\rho}{2}\|x_i - x_j\|^2\,, \end{aligned} \tag{3}$$

where $\rho > 0$ is a pre-chosen constant, and $\lambda \in (\mathbb{R}^n)^P$ is the vector collecting all the Lagrange multipliers: $\lambda = (\{\lambda_{ij}\})$. The dual problem is then

$$\underset{\lambda}{\text{maximize}} \quad L(\lambda)\,, \tag{4}$$

where the dual function is given by

$$L(\lambda) = \inf\{L(x_1, \ldots, x_P; \lambda) : A_p x_p = b_p,\ x_p \geq 0 \text{ for all } p\}\,. \tag{5}$$

Given a fixed λ, we represent by $x(\lambda) := (x_1(\lambda), \ldots, x_P(\lambda)) \in (\mathbb{R}^n)^P$ the vector that achieves the infimum in (5). It can be shown that this infimum is always finite due to our Assumption 1.

Next, we address the problem of solving the dual problem (4) in a distributed manner; and then we will see how one can obtain a primal solution (i.e., a solution of (2) and thus of (BP)) from an optimal dual solution $\lambda^\star$.

Solving the dual. The dual function L, defined in (5), satisfies the following properties.

Lemma 1 ([9]). *Let L be the dual function in* (5). *Then:*

1. *L is concave and finite-valued everywhere;*
2. *L is continuously differentiable everywhere and the gradient of L at the point $\lambda = (\ldots, \lambda_{ij}, \ldots)$ is*
$$\nabla L(\lambda) = (\ldots, x_i(\lambda) - x_j(\lambda), \ldots)\,; \tag{6}$$
3. *L has a Lipschitz gradient with constant $1/\rho$.*

Property 1 ensures that λ has no implicit constraints in (4). Hence, (4) is indeed unconstrained. Due to these properties of L, first Nesterov's method [6, §2.2.1] can be used to solve the dual problem (4). Using Nesterov's method the error in the objective function decreases with $O(1/k^2)$, where k denotes the iteration number, whereas if we used the standard gradient method, it would decrease with $O(1/k)$ [6].

In every iteration of Nesterov's method we are required to compute (6), the gradient of L. This implies finding $x(\lambda)$, the solution of the optimization problem in (5). Let λ be fixed. This means we have to solve

$$\begin{array}{ll} \text{minimize} & L(x_1, \ldots, x_P, \lambda) \\ \text{subject to} & A_p x_p = b_p\,,\ x_p \geq 0\,, \\ & p = 1, \ldots, P\,, \end{array} \tag{7}$$

where the variable is $(x_1, \ldots, x_P)$. The following properties of $L(x_1, \ldots, x_P, \lambda)$, when λ is fixed, enable us to use Nesterov's (projected) method again.

Lemma 2 ([9]). *Let $g_\lambda(x_1, \ldots, x_P) := L(x_1, \ldots, x_P; \lambda)$, i.e., g_λ is obtained from L by fixing λ. Then:*

1. *g_λ is convex, continuously differentiable everywhere and the*

gradient of g_λ with respect to x_p at $(x_1, \ldots, x_P)$ is

$$\nabla_{x_p} g_\lambda(x_1, \ldots, x_P) = \frac{1}{P} 1_n + \gamma_p + \rho D_p x_p - \rho \sum_{j \in \mathcal{N}_p} x_j \, , \tag{8}$$

where $\mathcal{N}_p$ is the set of neighbors of node p, $D_p = |\mathcal{N}_p|$, and $\gamma_p = \sum_{j \in \mathcal{N}_p} \text{sign}(j-p) \lambda_{pj}$;

2. *The largest eigenvalue of the Hessian matrix of g_λ is $\rho\lambda_{max}(\mathcal{L})$, where $\mathcal{L}$ is the Laplacian [13] of the network and $\lambda_{max}(\mathcal{L})$ its largest eigenvalue.*

If B is any bound for $\lambda_{\max}(\mathcal{L})$, then ρB will be a Lipschitz constant for ∇g_λ. The works [13, 14] propose bounds for $\lambda_{\max}(\mathcal{L})$ that are easily computed in a distributed way. In the sequel, given a fixed network, we will use B to denote the minimum of those bounds. Now, in every iteration of Nesterov's projected method we have to project the gradient (8) onto the the constraint set of (7), i.e., $\{y : A_p y = b_p, \, y \geq 0\}$. This projection, which we denote with $[\cdot]^+_{A_p, b_p}$, corresponds to solving an optimization problem which we do using a very efficient Barzilai-Borwein algorithm [15]. See [9] for details.

Once the dual problem (4) is solved, the solution of (2) is immediately available. This is a well known feature of augmented Lagrangian duality and is formally captured in the following lemma.

Lemma 3 ([9])**.** *If $\lambda^\star$ solves* (4)*, then $x(\lambda^\star)$ solves* (2)*.*

Since each node holds a copy of the BP variable, this means that it will immediately know an optimal solution to BP after (4) is solved.

The resulting algorithm. We can implement the method described above in a truly distributed manner. To see how, note that the relevant variables are the λ_{ij}'s, each of which is associated to the edge connecting nodes i and j, and the $x_p(\lambda)$'s, each of which is associated to node p. We assume that a copy of λ_{ij} is held in both nodes i and j. The only information node p will exchange with its neighbors is x_p (and it will also receive the respective x_p's from its neighbors). This is done in every iteration of the inner Nesterov loop. Note that operating this way every node has the required information to compute (8). Once this inner loop has converged, not only $x_p(\lambda^k)$ will be available at node p, but also $x_j(\lambda^k)$, where j is any neighbor of p. Here, λ^k denotes the value of λ at the kth iteration. Therefore, node p is able to compute the required parts of (6) for updating λ_{pj} or λ_{jp}.

Algorithm 1 gives a detailed description of our method. Lines 4 and 5 implement the outer Nesterov loop, i.e., the one that updates the λ_{ij}'s. The variables η_{ij} are auxiliary variables intrinsic to Nesterov's method. Note that $\nabla L_{\lambda_{ij}}(\eta^{(k-1)})$ in line 4 was computed in line 7 in the previous iteration by calling the function ComputeGL (Algorithm 2). This function implements Nesterov's inner loop in lines 3 and 4. Note that now the auxiliary variables are the y_p's.

Algorithms 1 and 2 also contain explicit stopping criteria for both loops (which are based on the value of the norm of the respective gradients), and a "safety communication" protocol that ensures that, whenever a node converges either in the inner or in the outer loops, its neighbors know that in the next iteration the respective node will not be active. This protocol is implemented using the variables StopIn and StopOut for the inner and outer loops, respectively.

3. EXPERIMENTAL RESULTS

We implemented Algorithms 1 and 2 in C and MPI (message passing interface) for the communication and used icc Version 11.0 with

Algorithm 1 Row partition: algorithm for node p

```
Require: $A_p \in \mathbb{R}^{m_p \times n}$ stored at the $p$th node and each node knows
    $\rho$ (predefined constant), the bound $B$, $\epsilon_{\text{in}}$, and $\epsilon_{\text{out}}$

Input: $b_p \in \mathbb{R}^{m_p}$
Output: $x^\star$ solving (1) at node $p$

Initialization: $k = 1$, $y_p^{(0)} = 0$ and $\lambda_{ij}^{(0)} = \eta_{ij}^{(0)} = 0$ for all $(i,j)$'s
    such that $i = p$ and $j \in \mathcal{N}_p$ or $j = p$ and $i \in \mathcal{N}_p$; StopOut $= 0$

 1: $\{\nabla L_{\lambda_{ij}}(\eta^{(0)})\} = \text{ComputeGL}(\eta^{(0)})$
 2: loop
 3:     for all $(i,j) : i = p, j \in \mathcal{N}_p$ or $j = p, i \in \mathcal{N}_p$ do
 4:         $\lambda_{ij}^{(k)} = \eta_{ij}^{(k-1)} + \rho \nabla L_{\lambda_{ij}}(\eta^{(k-1)})$
 5:         $\eta_{ij}^{(k)} = \lambda_{ij}^{(k)} + \frac{k-1}{k+2}\left(\lambda_{ij}^{(k)} - \lambda_{ij}^{(k-1)}\right)$
 6:     end for
 7:     $\{\nabla L_{\lambda_{ij}}(\eta^{(k)})\} = \text{ComputeGL}(\eta^{(k)})$
 8:     if $\|\nabla L_{\lambda_{ij}}(\eta^{(k)})\|/\sqrt{n} \leq \epsilon_{\text{out}}$ for all $(i,j)$, or all neighbors
    have stopped computing then
 9:         StopOut $= 1$
10:     end if
11:     Communicate StopOut to the neighbors
12:     if StopOut = 1 then
13:         break loop
14:     end if
15:     $k \leftarrow k + 1$
16: end loop
```

Algorithm 2 ComputeGL

```
Input: $\eta$
Output: Set of gradients $\{\nabla L_{\lambda_{ij}}(\eta)\}$

Initialization: Set $y_j^{(0)}$, for $j \in \mathcal{N}_p \cup \{p\}$, with the values from the
    last iteration; $t = 1$, $L_g = \rho B$, StopIn $= 0$

 1: Compute $\nabla_{y_p} g(y_p^{(0)}) = \frac{1}{P} 1_n + \gamma_p + \rho D_p y_p^{(0)} - \rho \sum_{j \in \mathcal{N}_p} y_j^{(0)}$
 2: loop
 3:     $x_p^{(t)} = \left[y_p^{(t-1)} - \frac{1}{L_g} \nabla_{y_p} g(y_p^{(t-1)})\right]^+_{A_p, b_p}$
 4:     $y_p^{(t)} = x_p^{(t)} + \frac{t-1}{t+2}(x_p^{(t)} - x_p^{(t-1)})$
 5:     if $\|\nabla_{y_p} g(y_p^{(t-1)})\|/\sqrt{n} \leq \epsilon_{\text{in}}$ or any of the neighbors has
    stopped then
 6:         StopIn $= 1$
 7:     end if
 8:     Send $y_p^{(t)}$ and StopIn to all neighbors and receive $y_j^{(t)}$ from
    all of them
 9:     Compute $\nabla_{y_p} g(y_p^{(t+1)}) = \frac{1}{P} 1_n + \gamma_p + \rho D_p y_p^{(t)} -$
    $\rho \sum_{j \in \mathcal{N}_p} y_j^{(t)}$
10:     if StopIn $= 1$ then
11:         break loop
12:     end if
13:     $t \leftarrow t + 1$
14: end loop
15: Send $x_p^{(t)}$ to all neighbors and receive $x_j^{(t)}$ from all of them
16: Compute $\nabla_{\lambda_{ij}} L(\eta) = \text{sign}(j - i)(x_i^{(t)} - x_j^{(t)})$ for all pairs of
    neighbors
```

Table 1. Comparison between Algorithm 1 (per node results) and spgl1. Algorithm 1 was executed in a cluster with 1 Gbit/s links.

			Algorithm 1 (distributed)			spgl1 (centralized)		
m	q	m_p	$\|Mx-b\|/\|b\|$	Sum of inner iterations	Time [s]	$\|Mx-b\|/\|b\|$	Time [s]	$\|x-x_{\text{spgl1}}\|/\|x_{\text{spgl1}}\|$
513	1000	57	3.75×10^{-7}	244	6	1.19×10^{-5}	0.1	3.01×10^{-5}
2574	5000	286	2.14×10^{-7}	214	122	6.09×10^{-6}	2.6	3.40×10^{-5}
5148	10000	572	1.59×10^{-7}	281	438	4.15×10^{-6}	14.9	4.78×10^{-5}
10296	20000	1144	1.31×10^{-7}	342	1400	————	——	————
20601	40000	2289	1.22×10^{-7}	336	5239	————	——	————
30897	60000	3433	1.10×10^{-7}	427	15500	————	——	————

flags -Wall -ansi -O3 -xWPT -mcpu=pentium4. As platform we use a computer cluster with $P = 9$ nodes. Each node has an Intel Xeon processor (six X3220, two E5430, and one E5440) with 2.4–2.8 GHz; the bandwidth of the links is 1 Gbit/s. We use the library MKL for the linear algebra operations, and generate the input data using Matlab.

Table 1 shows the results of computer experiments that were conducted as follows. For each problem dimension, we generated 10 times a random matrix $M \in \mathbb{R}^{m\times q}$, where $M_{ij} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1/m)$, and a vector $b \in \mathbb{R}^m$, created by multiplying M by a k-sparse vector, where k is the closest integer to $0.04q$. Each time we generated M and b, we also created a connected network with a random topology using the Erdös-Rényi model, where each edge is present with a probability of 25%. With this data and network we then ran Algorithm 1 in the cluster. On the same data we ran the centralized spgl1 algorithm [3] in Matlab and on a single computer (Intel Xeon X3220 processor with 2.40GHz). Every number in Table 1 represents the average value of a set of 10 experiments.

The problem size is specified in the first two columns of Table 1: the number of rows and the number of columns of M. The next column contains $m_p = m/P$, the number of rows of M stored in each node. Note that when $q \geq 20000$ we could not declare M due to Matlab memory limitations. Consequently, we were not able to execute spgl1 for those scenarios. For both algorithms we show the infeasibility $\|Mx - b\|/\|b\|$, which is a measure of the results' accuracy, and the execution time. For our algorithm, we also show the cumulative number of inner iterations, i.e., the sum of inner iterations for all outer loops. Finally, the last column contains the relative error between both algorithms' output, which was always less than 0.005%. This indicates that Algorithm 1 and spgl1 produce solutions of the same quality.

The entries for spgl1 that are missing are due to too large problem sizes: the matrix M could not be stored in the RAM of the machine used. Algorithm 1, due to its distributed nature, could still solve these problems.

For small problem sizes, those that spgl1 can solve, the distributed Algorithm 1 is 30–60x slower than spgl1. This is due to the distributed processing which is inherently computationally more expensive. The real benefit is in larger problem scenarios that cannot be stored within a single node and for which spgl1 is not applicable anymore. A computer cluster is no perfect model for a sensor network, but captures the distributed memory nature and the need for explicit communication.

4. CONCLUDING REMARKS

We proposed a truly distributed algorithm that solves BP in a connected network of arbitrary topology. This means the algorithm has no notion of a special or central node, and no node has access to all the data at any time. The distributed nature makes the algorithm computationally more expensive than standard algorithms but, in turn, enables the processing of problems that do not fit into the memory of a single node. Further, our algorithm can be used as a starting point to develop more advanced algorithms that are robust to link or node failures or that can dynamically incorporate new data.

5. REFERENCES

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

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

[3] 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.

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

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

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

[7] I. Lobel, A. Ozdaglar, and D. Feijer, “Distributed multi-agent optimization with state-dependent communication,” April 2010, submitted.

[8] 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*, April 2009.

[9] ——, “Distributed basis pursuit,” August 2010, arxiv link: http://arxiv.org/abs/1009.1128.

[10] J. Haupt and R. Nowak, “Signal reconstruction from noisy random projections,” *IEEE Trans. Inf. Th.*, vol. 52, no. 9, pp. 4036–4048, 2006.

[11] J. Meng, L. Husheng, and Z. Han, “Sparse event detection in wireless sensor networks using compressive sensing,” in *43rd CISS Conf.*, 2009.

[12] V. Cevher, M. Duarte, and R. Baraniuk, “Distributed target localization via spatial sparsity,” in *16th Eur. Sig. Proc. Conf., Eusipco2008*, 2008.

[13] J. Heuvel and Pejić, “Using laplacian eigenvalues and eigenvectors in the analysis of frequency assignment problems,” Department of Mathematics, London School of Economics, Tech. Rep., December 2000.

[14] J. Li and Y. Pan, “Upper bounds for the laplacian graph eigenvalues,” *Acta Mathematica Sinica*, vol. 20, no. 5, pp. 803–806, 2004.

[15] M. Raydan, “The barzilai and borwein gradient method for the large scale unconstrained minimization problem,” *SIAM J. Optimization*, vol. 7, no. 1, pp. 26–33, February 1997.