IRWIN AND JOAN JACOBS
CENTER FOR COMMUNICATION AND INFORMATION TECHNOLOGIES

Distributed Sparse Signal Recovery for Sensor Networks

Stacy Patterson, Yonina C. Eldar, and Idit Keidar

CCIT Report #821
November 2012

Electronics
Computers
Communications

DEPARTMENT OF ELECTRICAL ENGINEERING
TECHNION - ISRAEL INSTITUTE OF TECHNOLOGY, HAIFA 32000, ISRAEL

CCIT Report #821 November 2012

DISTRIBUTED SPARSE SIGNAL RECOVERY FOR SENSOR NETWORKS

Stacy Patterson, Yonina C. Eldar, and Idit Keidar

Department of Electrical Engineering
Technion - Israel Institute of Technology, Haifa, Israel
{stacyp,yonina,idish}@ee.technion.ac.il

ABSTRACT

We propose a distributed algorithm for sparse signal recovery in sensor networks based on Iterative Hard Thresholding (IHT). Every agent has a set of measurements of a signal x, and the objective is for the agents to recover x from their collective measurements at a minimal communication cost and with low computational complexity. A naïve distributed implementation of IHT would require global communication of every agent's full state in each iteration. We find that we can dramatically reduce this communication cost by leveraging solutions to the distributed top-K problem in the database literature. Evaluations show that our algorithm requires up to three orders of magnitude less total bandwidth than the best-known distributed basis pursuit method.

Index Terms— compressed sensing, distributed algorithm, iterative hard thresholding, top-K

1. INTRODUCTION

In compressed sensing, a sparse signal $x \in \mathbb{R}^N$ is sampled and compressed into a set of M measurements, where M is typically much smaller than N. If these measurements are taken appropriately, then it is possible to recover x from this small set of measurements [1].

Compressed sensing is an appealing approach for sensor networks, where measurement capabilities may be limited due to both coverage and energy constraints. Recent works have demonstrated that compressed sensing is applicable to a variety of sensor networks problems including event detection [2], urban environment monitoring [3] and traffic estimation [4]. In these applications, measurements of the signal are taken by sensors that are distributed throughout a region. The measurements are then collected at a single fusion center where signal recovery is performed. Due to limits in bandwidth, storage, and computation capabilities, it may be more efficient, and sometimes even necessary, to perform signal recovery in the network in a distributed fashion.

The work of Y. Eldar is supported in part by the Israel Science Foundation under Grant no. 170/10, and in part by the Ollendorf Foundation. This work was funded in part by the Arlene & Arnold Goldstein Center at the Technion Autonomous Systems Program, a Technion fellowship, and an Andrew and Erna Finci Viterbi Fellowship.

Distributed solutions for compressed sensing have begun to receive attention lately. For example, one work proposes a distributed subspace pursuit recovery algorithm for a mixed-support set model [5]. This work assumes that every agent knows the sensing matrix of every other agent. The need for global knowledge of these matrices presents a scalability bottleneck as individual sensors do not have the capacity to store and process a large number of these matrices. Several works have proposed distributed basis pursuit algorithms for sparse signal recovery in sensor networks where the measurement matrices are not globally known [6, 7, 8]. In these algorithms, agents collaborate to solve a convex relaxation of the original recovery problem. Each agent stores its own estimate of the signal x, and, in each iteration, it updates this estimate based on communication with its neighbors in the network. This approach requires that every agent solve a local convex optimization problem in each iteration. While these algorithms use only local communication, each agent must send its entire estimate vector to every neighbor in every iteration. This vector is not necessarily sparse until the algorithm converges, and therefore, the messages can be quite large. As a result, these algorithms have a large total bandwidth cost. Furthermore, simulations show that this bandwidth cost increases dramatically as the network connectivity increases.

We propose an alternative approach to distributed sparse signal recovery in sensor networks that is based on *Iterative Hard Thresholding* (IHT) [9]. In our distributed implementation of IHT, which we call D-IHT, all agents store identical copies of the estimate of x. In each iteration, every agent first performs a *local computation* to derive an intermediate vector. The agents then perform a *global computation* on their intermediate vectors to derive the next iterate. A naïve distributed implementation of IHT would require global communication of the intermediate vector of each agent in every iteration. We find that we can dramatically reduced the communication cost of this global computation by leveraging solutions to the distributed top-K problem in the database literature [10, 11, 12]. Our evaluations show that D-IHT requires up to three orders of magnitude less total bandwidth than the best-known distributed basis pursuit method. D-IHT is also computationally simpler since it does not require that agents solve local convex optimization problems. While, in this work, we present

our distributed recovery algorithm for compressed sensing, we note that our solution easily generalizes to sparse signal recovery from nonlinear measurements [13].

The remainder of the paper is organized as follows. In Section 2, we detail our problem setting and formulation and provide a brief description of IHT. The D-IHT algorithm is presented in Section 3. Section 4 gives numerical results on the performance of D-IHT.

2. PRELIMINARIES

2.1. Problem Formulation

We consider a set of P agents that form a connected, undirected static network topology with E edges. The agents may be the sensors themselves or they may be fusion nodes that collect measurements from several nearby sensors. Every agent knows the number of agents P, and we assume there is a unique agent identified as agent 1. If the uniquely identified agent is not defined a priori, one can be chosen using a variety of well-known distributed algorithms (see [14]). Agents communicate with their neighbors in the network using fixed size messages. Messaging is reliable but asynchronous, meaning that every message that is sent is eventually delivered, but the delay between sending and delivery may be arbitrarily long.

There is a K-sparse signal $x \in \mathbb{R}^N$ that the agents seek to estimate. Each agent $p = 1 \dots P$ has $M_p > 0$ (possibly noisy) measurements of x that have been taken using the agent's sensing matrix $A^p \in \mathbb{R}^{M_p \times N}$. There are $M = M_1 + \ldots + M_P$ measurements in total. The measurement vector of agent p, denoted b^p, is given by $b^p = A^p x + \epsilon^p$, where $\epsilon^p \in \mathbb{R}^{M_p}$ is the measurement error for agent p. Agents do not know the sensing matrices or measurement vectors of other agents.

Our goal is for every agent to recover the same signal x from their collective measurements at a minimal communication cost. Let b be the vector of all measurements, and let A be the sensing matrix for the entire system:

$$b := \left[\begin{array}{c} b^1 \\ \hline \vdots \\ \hline b^P \end{array}\right], \qquad A := \left[\begin{array}{c} A^1 \\ \hline \vdots \\ \hline A^P \end{array}\right].$$

To recover x from A and b, the agents must solve the following optimization problem,

$$\hat{x} = \arg\min_{x \in \mathbb{R}^N} \|Ax - b\|_2^2 \;\text{ subject to }\; \|x\|_0 \leq K, \qquad (1)$$

where $\|\cdot\|_0$ denotes the l_0 norm, i.e, the number of non-zero components.

This problem is known to be NP-Hard in general [15]. However, for suitable A matrices, efficient centralized algorithms to recover $\hat{x}$ exist. Our distributed solution is based on IHT [16, 9], which we describe next.

2.2. Iterative Hard Thresholding Algorithm

IHT is a gradient-like, iterative algorithm for finding a K-sparse vector $\hat{x}$ in a centralized setting where A and b are known. Let $\mathcal{T}_K(v)$ be the thresholding operator which returns a vector where all but the K entries of v with the largest magnitude are set to 0 (with ties broken arbitrarily). The IHT algorithm begins with an initial, arbitrary K-sparse vector x_0. In each iteration, a gradient-step is performed, followed by application of the thresholding operator:

$$x_{t+1} = \mathcal{T}_K\left(x_t - \alpha A^\mathsf{T}(b - Ax_t)\right). \qquad (2)$$

It has been shown that, for $\alpha < 1/\left(2\lambda_{max}(A^\mathsf{T}A)\right)$, IHT converges to a local minimum of (1) [9, 13].

We note that, even if A satisfies the properties necessary to enable recovery using IHT, it is not necessary and, in fact, not likely that each A^p satisfies these properties. Therefore it is not possible for any single agent to recover $\hat{x}$ on its own; agents must exchange information with one another to perform the recovery. In the next section, we present D-IHT, our distributed implementation of IHT.

3. DISTRIBUTED ITERATIVE HARD THRESHOLDING

In D-IHT, every agent stores an identical copy of x_t, which is initially 0. In each iteration t, each agent first performs a *local computation* to derive an intermediate vector $z_t^p \in \mathbb{R}^N$. The agents then perform a *global computation* on their intermediate vectors to derive the next iterate x_{t+1}, which is, again, identical at every agent. We now define these local and global computations.

Local computation. Each agent p computes a local residual vector, $y_t^p := b^p - A^p x_t$. The intermediate vector for agent p, denoted z_t^p, is then computed as follows,

$$z_t^p = \begin{cases} x_t - \alpha\left(A^p\right)^\mathsf{T} z_t^p & \text{if } p = 1, \\ -\alpha\left(A^p\right)^\mathsf{T} z_t^p & \text{otherwise.} \end{cases} \qquad (3)$$

Note that each agent can compute z_t^p using its local information.

Global computation. In the global computation step, all agents must compute a function G that depends on all of their intermediate vectors. This function is defined as follows,

$$x_{t+1} = G\left(z_t^1, \ldots, z_t^P\right) := \mathcal{T}_K\left(\sum_{p=1}^{P} z_t^p\right). \qquad (4)$$

We note that the combination of the local computation step (3) and the global computation step (4) are equivalent to (2).

A naïve implementation of G is for all agents to collaborate to compute all N sums, one for each component of the intermediate vectors. Then, each agent can independently determine the values with the K largest magnitudes. This approach

Agent 1		Agent 2		Agent 3	
z_t^1	L^1	z_t^2	L^2	z_t^3	L^3
21	$(1, 21)$	28	$(4, 45)$	2	$(3, 30)$
14	$(2, 14)$	3	$(1, 28)$	5	$(9, 29)$
11	$(4, 13)$	26	$(3, 26)$	30	$(7, 27)$
13	$(9, 12)$	45	$(10, 22)$	14	$(6, 15)$
2	$(3, 11)$	20	$(5, 20)$	6	$(4, 14)$
4	$(7, 10)$	10	$(9, 18)$	15	$(10, 7)$
10	$(8, 6)$	1	$(8, 13)$	27	$(5, 6)$
6	$(6, 4)$	13	$(6, 10)$	1	$(2, 5)$
12	$(5, 2)$	18	$(2, 3)$	29	$(1, 2)$
1	$(10, 1)$	22	$(7, 1)$	7	$(8, 1)$

(a) The vector z_t^p and the resulting sorted list L^p, at three agents.

step	agent	object	sum	τ_1	τ_2	τ_3	τ	top-2 set
1	1	1	51	21	?	?	?	$\{(1, 51)\}$
2	2	4	72	-	45	?	?	$\{(4, 72), (1, 51)\}$
3	3	3	67	-	-	30	96	$\{(4, 72), (3, 67)\}$
4	1	2	22	14	-	-	89	$\{(4, 72), (3, 67)\}$
5	2	10	30	-	22	-	66	$\{(4, 72), (3, 67)\}$

(b) Steps of TA to find top two objects. After five steps, the threshold τ is 65, and objects 3 and 4 both have sums greater than τ. No remaining objects can have a sum greater than τ. Therefore, the top two objects have been found, and the algorithm terminates.

Fig. 1: Example execution of the TA algorithm for $K = 2$.

requires communication of all components of all intermediate vectors and is thus very costly with respect to bandwidth. We derive a more communication efficient approach by leveraging work in the database literature on the distributed top-K problem. We describe this problem and a popular solution in Section 3.1. We then present our distributed algorithm for computing G in Section 3.2.

3.1. The Distributed Top-K Problem

In the distributed top-K problem, each agent p has a list L^p of pairs $(o, val^p(o))$, where o is an object ID and $val^p(o)$ is the value of object o at agent p. In our recovery problem, this list is generated from the vector z_t^p; o is the index into the vector z_t^p and $val^p(o)$ is the value of z_t^p at index o. Each object has a score; in our case, the score is the magnitude of the sum of object's values at all agents. The objective is to find the objects with the K largest scores. Clearly, it is possible to solve this problem by computing the sum for every object and then selecting the objects with the K largest magnitude sums, but it is often not necessary to compute all sums in order to find the top-K objects.

The *Threshold Algorithm* (TA) is a solution for the distributed top-K problem that is instance optimal, i.e., TA makes the minimum number of sum computations necessary for a given input of lists [10, 11, 12]. As it was originally proposed, TA requires that all values be non-negative[1]. Here, we present the algorithm, assuming that this is the case. In Section 3.2, we explain how we modify TA to support both negative and non-negative values.

In TA, each agent first sorts its list by object value, in descending order. One agent acts as the leader, requesting information from the other agents, computing sums for objects, and distributing the final top-K list to all agents. The algorithm proceeds as follows. The leader requests an object/score pair from each agent's list in sorted order, one pair from one agent in any given step. When the leader receives a pair containing an object it has not yet seen, it requests the value for that object from all other agents and computes the object's sum. The leader always stores the objects with the K largest sums it has seen so far. In addition, the leader stores the value of the last object seen from each agent p under the sorted access. This value is denoted τ_p. It computes the threshold value $\tau = \tau_1 + ...\tau_P$ in each step. As soon as the leader has seen K objects that each have a score of at least τ, the algorithm terminates. The leader then disseminates the list of top-K objects and their scores.

An example execution of TA is given in Figure 1. Note that, while each list has 10 objects, the algorithm only requires five sum computations to find the top two objects.

[1] More precisely, TA requires that that an object's score is given by a function that is monotonic in the values.

Algorithm 1: Pseudocode for D-IHT algorithm.

```
1 initialize
2     x_0^p ← 0
3     t ← 0
4 while TRUE do
5     z_t^p ← value from equation (3)        Local computation.
6     L^p ← Sort(z_t^p)                      Create sorted list from z_t^p.
7     x_{t+1}^p ← DATA(L^p)                  Global computation.
8     t ← t + 1
```

3.2. Distributed Computation of G

The computation of G is equivalent to solving a top-K problem over the vectors z_t^p, $p = 1 \dots P$. In D-IHT, we solve this top-K problem using a modified version of TA that is not leader-based and that accommodates both negative and non-negative values. We describe our modifications and the resulting algorithm below.

Support for non-negative values. As it was originally proposed, TA is applicable only to score functions like sum that are monotonic in the object values. To compute G, we need to find the top-K *magnitude* sums, which means that the score function is not monotonic unless all values are non-negative. A simple way to address this limitation is to run two instances of TA to find the top-K largest sums and the top-K smallest sums (since a sum with a negative value may have a large

magnitude). The top-K magnitude objects can then be found from this set of $2K$ objects. We implement a more efficient algorithm in which the agents find the top-K magnitude sums in a single algorithm instance by processing the sorted lists from both the top and the bottom.

Decentralized list processing. We speed up the execution of TA by having each agent independently processes its own list in sorted order. Each agent initiates a *group sum computation* for each new object it encounters, so that P group sum computations are executed in parallel for each iteration of the algorithm. In a group sum computation, every agent learns the sum of the values for the specified object. There are many distributed algorithms for group sum computation (e.g. [17, 18]). We use an algorithm based on the well-known *broadcast-convergecast* paradigm (see [19]). A request message is propagated down a broadcast tree that is rooted at the initiating agent. Each agent collects values from its children (if it has any), sums up these values with its own, and sends the result to its parent. For a network with E edges, the algorithm requires a preprocessing phase of less than $2E$ messages to create a broadcast tree. Each group sum computation requires $3(P-1)$ messages, and each agent p sends at most $3D_p$ messages, where D_p is the node degree.

The algorithm. We call our modified version of TA the Distributed Absolute Threshold Algorithm (DATA). We briefly summarize the algorithm below. The pseudocode is given in the appendix.

In DATA, Each agent stores two variables for global thresholds, one for sorted access from the top of the lists, denoted $\overline{\tau}^p$, and one for sorted access from the bottom of the lists, denoted $\underline{\tau}^p$. Both values are initially ∞. Each agent also stores a top-K list that is initially empty. The agents each execute the following steps.

1. If $\overline{\tau}^p > \underline{\tau}^p$, select next object from top of list for which p has not received a sum in a previous iteration. Else, select next object from bottom of list for which p has not received a sum in a previous iteration

2. Initiate group sum computation for selected object. Agent 1 also initiates a group sum computation for the global threshold, either $\overline{\tau}^p$ or $\underline{\tau}^p$, depending on whether it accessed its object from the top or bottom of its list. When participating in the group sum computation for $\overline{\tau}^p$, an agent uses the most recent value seen in sorted access from the top of its list, and for $\underline{\tau}^p$, it uses the most recent value seen in sorted access from the bottom.

3. On receipt of sums (from group computation) from all agents, update top-K list with objects that have sums with the K largest magnitudes seen so far.

4. On receipt of threshold (from group computation), update appropriate global threshold variable, $\overline{\tau}^p$ or $\underline{\tau}^p$. If the top-K list contains K objects with magnitudes at least $\max(|\overline{\tau}^p|, |\underline{\tau}^p|)$, return the list. Else, go to Step 1.

Table 1: Recovery problem parameters.

Problem	N	M	P	K	α
Random	1000	250	50	20	1
Sparco 7	2560	600	40	20	0.99
Sparco 11	1024	256	64	32	0.0025
Sparco 902	1000	200	50	3	0.99

We note that in a single iteration, multiple agents may initiate group sum computations for the same object. While, in theory, this introduces unnecessary message overhead, in practice the redundant sums do not add significantly to the total message cost.

3.3. The D-IHT Algorithm and Analysis

We combine the local computation step with DATA to arrive at the full D-IHT algorithm. The pseudocode is given in Algorithm 1. We note that while agent 1 plays a unique role in the local and global computations, it performs the same number and types of computations and sends the same number of messages as any other agent.

Storage complexity. Both D-IHT and distributed basis pursuit [6, 7, 8] require $O(N)$ storage per agent.

Message complexity. Let T_1 be the number of iterations of D-IHT required to achieve a certain accuracy. Let S_j be the number of group sum computations for iteration j (including the group threshold computations). Each sum computation requires $3(P-1)$ messages. Therefore, the total number of messages is $3(P-1)\sum_{j=1}^{T_1} S_j$. We compare this to distributed basis pursuit where, in every iteration, each agent sends its estimate of x to all of its neighbors. Assuming that the message size is limited to a single value, N messages are required to send a single estimate. Let T_2 be the number of iterations of distributed basis pursuit required to achieve the same accuracy as T_1 iterations of D-IHT. The total number of messages sent in distributed basis pursuit is $2NET_2$. For a connected network, $E \geq P-1$, and therefore, the total number of messages is at least $2NT_2(P-1)$.

The preprocessing phase of D-IHT requires at most 2EP messages, which is less than the number of messages required for one iteration of distributed basis pursuit. Therefore, if $T_1 < T_2$, then D-IHT requires fewer messages than distributed basis pursuit so long as less than $\frac{2}{3}N$ sums are computed per iteration of D-IHT, on average. In our evaluations, T_1 is always at least one order of magnitude smaller than T_2, and in most cases, the average number of sums computed per iteration of D-IHT is far fewer than $\frac{2}{3}N$.

4. NUMERICAL RESULTS

In this section, we present an experimental comparison of D-IHT and distributed basis pursuit. As a representative ex-

Table 2: Evaluation results for D-IHT and D-ADMM.

(a) ER graph with connection probability of 0.25.

Problem	Total Messages		Clock Ticks	
	D-IHT	D-ADMM	D-IHT	D-ADMM
Random	1.06×10^6	1.43×10^8	5.13×10^3	1.60×10^6
Sparco 7	2.23×10^6	1.11×10^8	2.59×10^4	2.02×10^6
Sparco 11	3.28×10^6	5.25×10^8	1.24×10^4	4.29×10^6
Sparco 902	1.48×10^6	5.58×10^7	9.09×10^3	7.20×10^5

(b) ER graph with connection probability of 0.75.

Problem	Total Messages		Clock Ticks	
	D-IHT	D-ADMM	D-IHT	D-ADMM
Random	1.13×10^6	1.21×10^9	3.75×10^3	1.18×10^7
Sparco 7	2.27×10^6	7.33×10^8	9.66×10^3	9.33×10^6
Sparco 11	3.41×10^6	4.88×10^9	8.06×10^3	3.41×10^7
Sparco 902	1.54×10^6	2.67×10^8	5.78×10^3	2.65×10^6

(c) Geometric graph with d=0.5.

Problem	Total Messages		Clock Ticks	
	D-IHT	D-ADMM	D-IHT	D-ADMM
Random	1.05×10^6	7.22×10^7	2.46×10^4	1.68×10^6
Sparco 7	2.23×10^6	6.06×10^7	7.60×10^4	1.99×10^6
Sparco 11	3.26×10^6	2.37×10^8	6.24×10^4	3.66×10^6
Sparco 902	1.46×10^6	2.94×10^7	5.17×10^4	6.88×10^5

ample, we select D-ADMM, a distributed implementation of the alternating direction method of multipliers that has been shown to outperform other distributed basis pursuit algorithms in terms of the number of communications in similar experiments [8]. In each iteration of D-ADMM, each agent exchanges its estimate with its neighbors and generates a new estimate by solving a local optimization problem involving its estimate and the estimates of some of its neighbors. We have implemented D-IHT and D-ADMM in Matlab, using CVX [20] to solve the local optimization problems in D-ADMM. D-ADMM requires a graph coloring, which we generate using the heuristic from the Matgraph toolbox [21], as is done in [8]. We include the preprocessing phase in our results for D-IHT, but we do not include graph coloring pre-processing in our results for D-ADMM.

Recovery problems. We evaluate the performance of D-IHT and D-ADMM on four reconstruction problems, similar to those in [8]. For the first problem, we generate the A matrix with i.i.d Gaussian entries with zero mean and variance of $1/m$. The remaining three problems are from the Sparco toolbox [22]. The parameters for each problem are given in Table 1. For each problem, we divide the A matrix evenly among the agents so that each agent has M/P rows. For the randomly generated problem, we find the optimal sparse solution $\hat{x}$ using CVX. For the Sparco problems, we use the provided optimal sparse solution.

Performance measures. For each algorithm, we measure the total number of messages sent in order for $\|x_t^p - \hat{x}\|/\|\hat{x}\| \leq 10^{-2}$ for all agents. To standardize the bandwidth comparison between the algorithms, we assume that only one value is sent per message. Therefore, in D-ADMM, when an agent sends its N-vector to its neighbor, this requires N messages. D-IHT is designed so that only one value is sent per message. We also measure the time required for convergence in a synchronous network where each message is delivered in one clock tick. For both algorithms, we only allow one message to be sent on a link in each direction per clock tick.

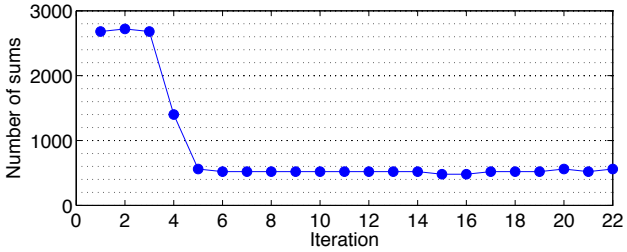

Fig. 2: Number of sums computed per iteration by D-IHT to solve Sparco problem 7 (with $N = 2560$) in a 40 node ER graph with connection probability of 0.25.

Results. Table 2 shows the results of our evaluations in three different network topologies. The first is an Erdös-Rényi (ER) graph [23] where each pair of vertices is connected with probability 0.25 (Figure 2a), and the second is an ER graph where each pair of vertices is connected with probability 0.75 (Figure 2b). The third network topology is a geometric graph [24] with vertices placed uniformly at random in a unit square, and two vertices are connected if they are within a distance of 0.5 of each other (Figure 2c).

These results show that, for every recovery problem, D-IHT requires far fewer total messages than D-ADMM to achieve the same recovery accuracy, between one and two orders of magnitude in most cases. D-IHT also requires less total time to perform the recovery than does D-ADMM. We note that, as network connectivity increases, in D-ADMM the total message count and total time increase (Table 2a vs. Table 2b). In D-IHT, sums can be computed more quickly in networks that are more connected. Therefore, in D-IHT, the recovery time decreases as network connectivity increases.

A key to the good performance of D-IHT is that, after just a few iterations, the algorithm finds the correct support set (the non-sparse components of the signal). The magnitudes of the values in the support set quickly dominate the other values in the intermediate vectors. As a result, in DATA, the sums for these objects are computed first, and the top-K objects are identified after a minimal number of sum computations (on the order of PK). This behavior is illustrated in Figure 2, where we show the total number of sums computed for each iteration of D-IHT for a single experiment. This figure shows a dramatic drop in the number of sum computations after just four iterations.

5. REFERENCES

[1] M.F. Duarte and Y.C. Eldar, "Structured compressed sensing: From theory to applications," *IEEE Trans. Sig. Proc.*, vol. 59, no. 9, pp. 4053–4085, Sep 2011.

[2] J. Meng, H. Li, , and Z. Han, "Sparse event detection in wireless sensor networks using compressive sensing," in *Proc 43rd Ann. Conf. Information Sciences and Systems*, 2009.

[3] Z. Li, Y. Zhu, H. Zhu, and M. Li, "Compressive sensing approach to urban traffic sensing," in *Proc. 31st Int. Conf. Distributed Computing Systems*, 2011, pp. 889–898.

[4] X. Yu, H. Zhao, L. Zhang, S. Wu, B. Krishnamachari, and V. O. K. Li, "Cooperative sensing and compression in vehicular sensor networks for urban monitoring," in *2010 IEEE Int. Conf. on Communications*, 2010, pp. 1–5.

[5] D. Sundman, S. Chatterjee, and M. Skoglund, "A greedy pursuit algorithm for distributed compressed sensing," in *Proc. IEEE Int. Conf. on Acoust., Speech, and Sig. Proc. (ICASSP)*, 2012, pp. 2729–2732.

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

[7] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "Basis pursuit in sensor networks," in *Proc. IEEE Int. Conf. on Acoust., Speech, and Sig. Proc. (ICASSP)*, 2011, pp. 2916–2919.

[8] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "Distributed basis pursuit," *IEEE Trans. Sig. Proc.*, vol. 60, no. 4, pp. 1942–1956, Apr 2012.

[9] T. Blumensath and M. E. Davies, "Iterative hard thresholding for compressed sensing," *Applied and Computational Harmonic Analysis*, vol. 27, no. 3, pp. 265–274, 2009.

[10] S. Nepal and M.V. Ramakrishna, "Query processing issues in image (multimedia) databases," in *Proc. 15th Int. Conf. Data Engineering*, 1999, pp. 22–29.

[11] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *J. Computer and System Sciences*, vol. 66, no. 4, pp. 614–656, 2003.

[12] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid, "Supporting top-k join queries in relational databases," *The VLDB Journal*, vol. 13, no. 3, pp. 207–221, Sep 2004.

[13] A. Beck and Y.C. Eldar, "Sparsity constrained nonlinear optimization: Optimality conditions and algorithms," *CoRR*, vol. abs/1203.4580, 2012.

[14] N. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers, Inc., USA, 1996.

[15] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM J. Comput.*, vol. 24, no. 2, pp. 227–234, Apr 1995.

[16] T. Blumensath and M. E. Davies, "Iterative thresholding for sparse approximations," *J. Fourier Analysis and Applications*, vol. 14, no. 5, pp. 629–654, Dec 2008.

[17] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.

[18] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. 44th Ann. IEEE Sym. Foundations of Computer Science*. IEEE, 2003, pp. 482–491.

[19] A. Segall, "Distributed network protocols," *IEEE Trans. Inf. Theory*, vol. 29, no. 1, pp. 23–35, 1983.

[20] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 1.21," `http://cvxr.com/cvx`, Apr 2011.

[21] E. R. Scheinerman, "Matgraph: A matlab toolbox for graph theory," Online: `http://www.ams.jhu.edu/~ers/matgraph/matgraph.pdf`, 2012.

[22] E. van den Berg, M. P. Friedlander, G. Hennenfent, F. Herrmann, R. Saab, and Ö. Yılmaz, "Sparco: A testing framework for sparse reconstruction," Tech. Rep. TR-2007-20, Dept. Computer Science, University of British Columbia, Vancouver, October 2007.

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

[24] M. Penrose, *Random Geometric Graphs*, Oxford University Press, Oxford, U.K., 2004.

A. PSEUDOCODE FOR DATA

Algorithm 2: Distributed Absolute Threshold Algorithm, as it is executed by each node p.

function DATA($L^p = \{(ndx, val)\}_{i=1}^{N}$)
  $topKList \leftarrow \emptyset$, $top \leftarrow 1$, $bottom \leftarrow N$
  $\overline{\tau}^p \leftarrow \infty$, $\underline{\tau}^p \leftarrow \infty$
  $done \leftarrow$ FALSE
  while $done =$ FALSE **do**
    if $\overline{\tau}^p > \underline{\tau}^p$ **then**
      $oid \leftarrow$ new object id from top of list
    else
      $oid \leftarrow$ new object id from bottom of list
    GroupComputeSum(oid)
    if $p = 1$ **then**
      GroupComputeThreshold(oid)
    Receive (ndx^q, sum^q) for $q = 1 \ldots P$ **and** receive $threshold$
    if $\overline{\tau}^p > \underline{\tau}^p$ **then**
      $\overline{\tau}^p \leftarrow threshold$
    else
      $\underline{\tau}^p \leftarrow threshold$
    for $q = 1$ **to** P **do**
      if $|topKList| < K$ **then**
        Add (ndx^q, sum^q) to $topKList$
      else if $sum^q >$ *min abs. sum in* $topKList$ **then**
        Replace smallest magnitude element with (ndx^q, sum^q)
    if *min. abs. sum in* $topKList \geq \max(\overline{\tau}^p, \underline{\tau}^p)$ **then**
      $done \leftarrow$ TRUE
  $x \leftarrow$ GenerateVectorFromList($topKList$)
  return x