On Fast Decoding of High-Dimensional Signals from One-Bit Measurements*

Vasileios Nakos†

Harvard University, Cambridge, MA, USA
vasileiosnakos@g.harvard.edu

Abstract

In the problem of one-bit compressed sensing, the goal is to find a δ-close estimation of a k-sparse vector $x \in \mathbb{R}^n$ given the signs of the entries of $y = \Phi x$, where Φ is called the measurement matrix. For the one-bit compressed sensing problem, previous work [32, 19] achieved $\Theta(\delta^{-2} k \log(n/k))$ and $\tilde{\mathcal{O}}(\frac{1}{\delta} k \log(n/k))$ measurements, respectively, but the decoding time was $\Omega(nk \log(n/k))$. In this paper, using tools and techniques developed in the context of two-stage group testing and streaming algorithms, we contribute towards the direction of sub-linear decoding time. We give a variety of schemes for the different versions of one-bit compressed sensing, such as the for-each and for-all versions, and for support recovery; all these have at most a $\log k$ overhead in the number of measurements and $\text{poly}(k, \log n)$ decoding time, which is an exponential improvement over previous work, in terms of the dependence on n.

1998 ACM Subject Classification F.2.0 [Anaysis of Algorithms and Problem Complexity] General

Keywords and phrases one-bit compressed sensing, sparse recovery, heavy hitters, dyadic trick, combinatorial group testing

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.61

1 Introduction

1.1 Standard Compressed Sensing

The compressed sensing framework describes how to reconstruct a vector (signal) $x \in \mathbb{R}^n$ given the linear measurements $y = \Phi x$ where $\Phi \in \mathbb{R}^{m \times n}$ for some $m \ll n$. This is an under-determined system with n variables and m equations. In many applications, however, such as images, we know that the vector x can be approximated by a k-sparse vector in some known basis. In this case, the matrix Φ contains a sufficient amount of information to roughly recover x if m is large enough; in particular, as shown in [4, 5], the signal can be approximately reconstructed from $\Theta(k \log(n/k))$ measurements when Φ is a Gaussian matrix. In order to do this, however, one has to solve the non-convex program

$$\min \|x\|_0 \text{ s.t. } y = \Phi x$$

However, [9, 4] show it is possible to avoid the non-convex formulation and, alternatively, we can use Basis Pursuit (BP), which changes the objective to $\min \|x\|_1$, and still recover a decent approximation of x. This can be solved using linear programming.

Compressed sensing, and sparse recovery, have appeared to be very useful tools in many areas such as analog-to-digital conversion [25], threshold group testing [1], discrete

* A full version of the paper is available at https://arxiv.org/abs/1603.08585.
† Supported in part by ONR grant N00014-15-1-2388.

44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 61; pp. 61:1–61:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

signal processing [12], streaming algorithms [29] and bioinformatics [26]. Depending on the application, one cares about optimizing different parameters of interest (measurements, decoding time, encoding time, failure probability).

Often we distinguish between the for-all model (or uniform recovery) and the for-each model (non-uniform recovery). In the for-all model, a single matrix is picked, which allows reconstruction of all k-sparse vectors. Whereas in the for-each model, the measurements are chosen at random such that, for some error probability p, they will contain sufficient information to reconstruct a single vector x with probability at least $1 - p$. We note that all aforementioned papers refer to the for-all model.

Moreover, it is desirable to achieve *sublinear* decoding time. The state of the art for the for-each model is [15]. The authors there achieve $k \cdot \text{poly}(\log n)$ decoding time with $\Theta(k \log(n/k))$ measurements. The failure probability was improved later in [17] using a much more complicated scheme. In the for-all model, Gilbert et.al. [18] give the first algorithm that runs in sublinear time with a number of measurements $\mathcal{O}(k\text{poly}(\log n))$. Porat and Strauss devised a scheme under a slightly weaker recovery guarantee with $\mathcal{O}(k \log(n/k))$ measurements accompanied with the first sublinear decoding procedure running in time $\mathcal{O}(k^{1-\alpha} n^{\alpha})$, for any constant α [33]. Subsequent work [16], the authors manage to bring the dependence of the approximation ϵ fact down to the right order of ϵ^{-1} and achieve runtime $\text{poly}(k, \log n)$, when $\epsilon \leq (\frac{\log k}{\log n})^{\gamma}$, for any constant γ.

1.2 One-Bit Compressed Sensing

Often in applications compressed sensing measurements must be quantized, since the requirement of infinite precision is not realistic: any measurement must be mapped to a small finite value in some universe [3]. Thus, one-bit compressed sensing emphasizes the compressed aspect of compressed sensing; many algorithms for standard compressed sensing rely on infinite precision in their real-valued inputs, relying on more precision than real sensors are capable of returning, an assumption which is unrealistic for real-world applications. Moreover, in hardware implementations, for example, where quantizers are implemented using comparators to zero [3], there is need of quantization to one-bit measurements. Comparators are indeed fast, but they are expensive, so minimizing their usage is really important. Moreover, dynamic range issues are a smaller problem in the case of one-bit quantizers.

It is clear that quantization increases the complexity of the decoding procedure and, additionally, is irreversible: given $y = \text{sign}(\Phi x)$ it is impossible to get the exact vector back. Previous results inquired the case in which the quantization maps each coordinate to $\{-1, +1\}$, which means that we learn only the sign of each coordinate. First, it is not obvious whether there is sufficient information to reconstruct a signal given its one-bit measurements. Of course, since we cannot know the length of the signal, nor the exact signal (even if its length were given), we can ask the following question: can we find its direction?

The problem was first studied in the work of Boufounos and Baraniuk [3], where the authors suggest recovering the signal x by solving the optimization problem

$$\min \|x\|_1 \text{ s.t.: } y \odot Ax \geq 0, \|x\|_2 = 1,$$

where $\odot$ stands for the element-wise product between two vectors. The goal is to find a vector y on the unit sphere such that $\|y - \frac{x}{\|x\|_2}\|_2^2 \leq \delta$. It is clear that this relaxation requires solving a non-convex program, an obstacle which Laska et al. [28] tried to remedy by giving an optimization algorithm that finds a stationary point of the aforementioned program; both papers, however, do not provide provable guarantees for the number of measurements needed. An alternative formulation was studied in [24], which showed that the number

of measurements could be brought down to $\mathcal{O}(\delta^{-1}k\log n)$, but the main obstacle of the non-convex formulation remained. In [31] Vershyin and Plan gave the first computationally tractable algorithm for the problem of one-bit compressed sensing by designing a compressed sensing scheme that approximately recovers a k-sparse vector from $\mathcal{O}(\delta^{-5}k\log^2(\frac{n}{k}))$ one-bit measurements via a linear programming relaxation. Their techniques were based on random hyperplane tessellations; the main geometric lemma they needed was that $\mathcal{O}(k\log(n/k))$ random hyperplanes partition the set of k-sparse vectors with unit norm into cells, each one having small diameter. In [32] the same authors improved the number of measurements to $\mathcal{O}(\delta^{-2}k\log(\frac{n}{k}))$ by analyzing a simple convex program. Their results can also be generalized to other sparsity structures, where the crucial quantity that determines the number of measurements is the gaussian mean-width of the set of all unit vectors having a specific sparsity pattern. Last but not least, they manage to handle gaussian noise and, most importantly, adversarial bit flips, though with a small worsening in the dependence on δ in their number of measurements. In [19] a two-stage algorithm with $\tilde{\mathcal{O}}(\frac{1}{\delta}k\log(n/k))$ measurements and $\mathcal{O}(nk\log(n/k)+\frac{1}{\delta^5}(k\log(n/k))^5)$ decoding time was proposed. Apart from recovering the vector, other algorithms that recover only the support of the signal have been proposed; see for example [19, 22].

In [20] it is conjectured that even if the support of the vector is known, the dependence of the number of measurements on δ must be at least $\frac{1}{\delta}$. In order to circumvent this, alternative quantization schemes were proposed, with the most common being Sigma-Delta quantization [21, 27]. In [2] Baraniuk, Foucart, Needell, Plan and Wootters manage to bring the dependence on δ down to $\log(\frac{1}{\delta})$ if the quantizer is allowed to be adaptive and the measurements take a special form of threshold signs.

1.3 Group Testing

In the group testing problem, we have a large population, which consists of "items", with a known number of defectives. The goal is to find the defectives using as few tests as possible, where a test is just a query whether a certain subset of items contains at least one defective. The group testing problem was first studied by Dorfman in [13]. There are two types of algorithms for this problem, namely adaptive and non-adaptive. In the first case, the outcome of previous tests can be used to determine future tests, whereas in non-adaptive algorithms all tests are performed at the same time. Group testing has many applications in DNA library screening and detection of patterns in data; more can be found in [6], [8].

Any solution for the group testing problem corresponds to a binary matrix, where the number of rows equals the number of tests and the number of columns equals the cardinality of the population. Given such a matrix M and a vector x indicating the positions of the defectives, we should be able to identify x from Mx, where the addition here corresponds to the OR operation of Boolean algebra. Since decoding time is important, the brute-force algorithm that iterates over each possible subset in order to recognise the defective set does not suffice. However, one can design matrices such that the naive decoding algorithm, which eliminates items belonging to negative tests and returns all the other items, correctly identifies all defective items [14]. In the literature these matrices are known as k-disjunct matrices.

In this paper, we are also interested in the so-called two-stage group testing problem, where two stages are allowed: the first stage recognises a superset of the defectives, and the second stage, which is performed after seeing the results of the first stage, recognizes the exact set of the defectives by querying separately for each one. We refer to (k,l) two-stage group testing as the case when there are k defectives and the superset is allowed to have up to $k+l$ elements. In fact, this is equivalent to the existence of a matrix M such that given

Mx one can find a set S with $k+l$ elements such that all defectives are included in S. The same naive algorithm, which eliminates all items that belong to a negative test and returns all other items, will be used here. A matrix is called (k,l) list-disjunct if this algorithm finds a superset of the support with at most $k+l$ elements. The term 'list-disjunct' appeared in [23], although it was also studied before in [11] under the name of super-imposed codes, and in [34] under the name of list-decoding super-imposed codes. In [30] Ngo, Porat and Rudra give efficient and strongly explicit constructions of matrices that allow two-stage group testing, which are also error-tolerant, in the sense that they can correct e_0 false positives and e_1 false negatives in sub-linear time and additional $\Theta(e_0 + k \cdot e_1)$ tests. They also prove matching lower bounds for several cases, including the case that $k = \Theta(l)$.

A group testing scheme is a tuple (M, R), where M is a matrix in $\{0,1\}^{m\times n}$ and R a procedure that takes as input Mx and outputs a vector y. Depending on the guaratee we want, we will either refer to it as two-stage group testing or (one-stage) group testing. The worst-case running time of the procedure R corresponds to the decoding time of the scheme.

1.4 Our Results

The main goal of our work is to make algorithms for one-bit compressed sensing not only "data-efficient" but also computationally efficient. Thus, we try to understand under which conditions and which number of measurements sublinear decoding time is possible. In the for-each version of noisy one-bit compressed sensing we give a scheme with almost-optimal measurements and sublinear decoding time. We also focus on decoding noiseless signals and presents several results towards this direction. We first give a scheme with sublinear decoding time with a small overhead in the number of measurements, by connecting the problem with Combinatorial Group Testing. Second, we try to understand whether it is possible to achieve a for-all guarantee for one-bit compressed sensing, while still keeping sublinear decoding time. We answer this question in the affirmative if we are allowed to use $\mathcal{O}(k^2 \log n)$ measurements. Our techniques also give a scheme for support recovery that outperforms the one in [19] by being exponentially faster than it; one additional aspect of our scheme is that the measurement matrix is explicit, which means that we can compute it in polynomial time.

For the case of general vectors, we define the δ-ℓ_2/ℓ_2 guarantee: For a unit vector $x \in \mathbb{R}^n$ we say that a scheme satisfies the δ-ℓ_2/ℓ_2 guarantee for one-bit compressed sensing if the output satisfies

$$\|\hat{x} - x\|_2^2 \leq c\|x_{\mathrm{tail}(k)}\|_2^2 + \delta,$$

while $x_{\mathrm{tail}(k)}$ is the vector that occurs after zeroing out the biggest k coordinates of x in magnitude and c is some absolute constant.

In the support recovery problem, one wants to construct a matrix Φ, such that for all k-sparse x, one is able to recover the support of the vector x, given measurements $y = \mathrm{sign}(\Phi x)$.

We present the results that we have in greater detail in Tables 1 and 2. We note that the decoding time of each scheme is $\mathrm{poly}(k, \log n)$.

- δ-ℓ_2/ℓ_2 for-each one-bit Compressed Sensing from $\mathcal{O}(k \log(n/k) \cdot (\log k + \log\log(n/k)) + \delta^{-2}k)$ measurements.
- For-each one-bit Compressed Sensing (noiseless signals) from $\mathcal{O}(k \log n + \log_k n \cdot \log\log_k n + \delta^{-2}k)$ measurements. This extends the result of [32], as it manages to also decrease the number of measurements for the for-each version of the problem for a wide range of parameters.

Table 1 Comparison of recovery schemes for one-bit compressed sensing.

Algorithm	Measurements	Decoding-Time	Model	Noise
[32]	$\delta^{-6}k\log(\frac{n}{k})$	$\text{poly}(n)$	For-all	Type 1
[32]	$\delta^{-2}k\log(\frac{n}{k})$	$\text{poly}(n)$	For-each	Type 2
[19]	$\tilde{\mathcal{O}}(\delta^{-1}k\log(\frac{n}{k}))$	$\mathcal{O}(nk\log n)+\text{poly}(k,\log n)$	For-all	No
This paper	$\delta^{-2}k+k\log(\frac{n}{k})(\log k+\log\log(\frac{n}{k}))$	$\text{poly}(k,\log n)$	For-each	Type 3
This paper	$k\log n+\delta^{-2}k+\log_k n\cdot\log\log_k n$	$\text{poly}(k,\log n)$	For-each	No
This paper	$k^2\log n\log\log_k n+\delta^{-6}k\log(\frac{n}{k})$	$\text{poly}(k,\log n)$	For-all	No

Table 2 Comparison of schemes for support recovery.

Algorithm	Measurements	Decoding time	Model
[19]	$k^3\log n$	$nk\log n$	For-all
This paper (Theorem 18)	$k^3\log n$	$k^3\text{poly}(\log n)$	For-all

- For-all one-bit Compressed Sensing (noiseless signals) in $\mathcal{O}(k^2\log n\log\log_k n+\delta^{-2}k\log n)$ measurements. This is the first scheme that allows sublinear decoding time in the for-all model of one-bit compressed sensing, although the dependence on k is k^2.
- Support recovery from one-bit measurement (noiseless signals) in $\mathcal{O}(k^3\log n)$ measurennts. This scheme is not only exponentially faster than the one presented in [19], but also explicit, in the sense that the matrix can be computed in polynomial time in n.

An interesting aspect of our results is that in the for-each model, the δ factor does not need to multiply the $k\log n$ factor, in contrast to the for-all version. We explain the three types of noise handled by the schemes in Table 1:

- Type 1 stands for adversarial bit flips. This means that after receiving $y=\text{sign}(\Phi x)$, an adversary can flip some of the entries of y, and then give it to the decoder. Here, we assume that x is exactly k-sparse. The result of [32] can tolerate up to $c\delta$ fraction of adversarial bit flips, where c is some absolute constant smaller than 1.
- Type 2 noise stands for gaussian random noise that is added to the k-sparse vector x after the matrix Φ has been applied to it. This means that $y=\text{sign}(Ax+u)$, where $u\sim c\mathcal{N}(0,I)$, where c is some absolute constant.
- Type 3 noise refers to general noise and is handled by the δ-ℓ_2/ℓ_2 guarantee. This means that $y=\text{sign}(\Phi(x_{\text{head}(k)}+x_{\text{tail}(k)}))$, where we can view the term $x_{\text{tail}(k)}$ as pre-measurement adversarial noise.

The ideas that are used in this paper to obtain sublinear decoding time are based on ideas that appeared in [30], as well as the dyadic trick, which has appeared in the streaming literature in the context of the Count-Min Sketch [10]. As far as we know, our work is the first that looks at sublinear decoding time in the one-bit compressed sensing framework and even achieves less measurements in some cases. Last but not least, we believe that a strong point of our schemes is their simplicity.

2 Preliminaries

We define the sign function as $\text{sign}(z)=+1$, for $z\geq 0$ and $\text{sign}(z)=-1$ for $z<0$. For a vector x, we define $\text{sign}(x)_i=\text{sign}(x_i)$, for all $i\in[n]$.

Any one-bit compressed sensing scheme is defined by a pair $(\mathcal{D}, \mathrm{Dec})$ where $\mathcal{D}$ is a distribution over $\mathbb{R}^{m\times n}$ and Dec is an algorithm that takes input $\mathrm{sign}(\Phi x)$ for some $x \in \mathbb{R}^n$ and gives back a vector $\hat{x}$. We will refer to Dec either as the "decoder" or "decoding procedure". We may also slightly abuse notation and refer to the pair (A, Φ), where A is a matrix coming from some distribution $\mathcal{D}$. We use m to denote the number of measurements, and *decoding time* refers to the running time of *Dec*. We also define $\Sigma_k = \{x : \|x\|_0 \leq k, \|x\|_2 \leq 1\}$ to be the set of all k-sparse vectors contained in the unit ℓ_2 ball, and $\Sigma_k^1 = \{x : \|x\|_2 = 1, \|x\|_0 \leq k\}$ the set of unit norm vectors with at most k non-zero coordinates.

For $x \in \mathbb{R}^n$ we denote its support set by $\mathrm{supp}(x)$. For a vector x, $\mathrm{head}(k)$ denotes the set of its k largest coordinates in magnitude, while $\mathrm{tail}(k)$ denotes the set of its $n-k$ smallest coordinates in magnitude. For a coordinate i, we say that i is a $\frac{1}{k}$-heavy hitter if, for some absolute constant C_h it holds that $x_i^2 \geq \frac{1}{C_h k}\|x_{\mathrm{tail}(k)}\|_2^2$. The constant C_h will be chosen later. For any x which we are sensing using a one-bit compressed sensing scheme, we assume that it $\|x\|_2 = 1$.

For each $S \subset n$, let $x_S \in \mathbb{R}^{|S|}$ denote the signal x restricted to coordinates in S. Similarly, for a matrix $M \in \mathbb{R}^{r\times n}$ and each $S \subset n$ let $M_S \in \mathbb{R}^{r\times|S|}$ be the matrix M restricted to columns in S.

▶ **Definition 1.** A scheme $(\mathcal{D}, \mathrm{Dec})$ satisfies the δ-ℓ_2/ℓ_2 guarantee for one-bit compressed sensing with failure probability p if for each $x \in \mathcal{S}^{n-1}$, it estimates a vector $\hat{x}$ such that

$$\forall x, P_{\Phi\sim\mathcal{D}}[\hat{x} = \mathrm{Dec}(\Phi x) : \|x - \hat{x}\|_2^2 \leq C\|x_{\mathrm{tail}(k)}\|_2^2 + \delta] \geq 1 - p,$$

where C is an absolute constant.

For function $f : \mathbb{R}^n \to \mathbb{R}^m$ and a vector $x \in \mathbb{R}^n$ we say that $y = f(x)$ is a sketch of x. In our case, f will always be of the form $f(x) = \mathrm{sign}(Ax)$, where $A \in \mathbb{R}^{m\times n}$.

We also give the definition of the tensor product of two matrices. We note that this is not the standard tensor product (or Kronecker product, as usually known) appearing in the literature.

▶ **Definition 2.** Let $A \in \{0,1\}^m \times \{0,1\}^N$ and $B \in \{0,1\}^{m'} \times \{0,1\}^N$. The tensor product $A \otimes B$ is an $mm' \times N$ binary matrix with rows indexed by the elements of $[m] \times [m']$ such that for $i \in [m]$ and $j \in [m']$, the row of $A \otimes B$ indexed by (i, j) is the coordinate-wise product of the i-th row of A and j- th row of B.

In order to proceed, we have to explain the difference between the for-all and the for-each model. Let P_x be the predicate that the sparse recovery scheme returns a vector $\hat{x}$ such that $\|x - \hat{x}\|_2^2 > \delta$, when the matrix Φ is chosen from the distribution $\mathcal{D}$. Let p be some target probability. In the for-each model the guarantee is that $\forall x \in \Sigma_k^1, \mathbb{P}[P_x] \leq p$. In the for-all model the guarantee is that $\mathbb{P}[\exists x \in \Sigma_k^1 : P_x] \leq p$. The randomness of the scheme is over the distribution $\mathcal{D}$.

The following result, appearing in [32] is a crucial component of most of our algorithms. This theorem is a special case of Theorem 1.1 from that paper, and it is also discussed in subsection 3.1 of the same paper (check "random noise before quantization" discussion).

▶ **Theorem 3.** *Let A be a random $m \times n$ matrix, with each entry being a standard gaussian, and all entries are independent. Let $x \in \Sigma_k^1$ and $y = \mathrm{sign}(Ax + v)$, where $v \sim \mathcal{N}(0, I)$. Then, the convex program*

$$z = \mathrm{argmax}\,\langle y, Az\rangle \;\; s.t. \;\; \|z\|_2 \leq 1, \|z\|_1 \leq \sqrt{k},$$

outputs a $\hat{x}$ such that $\|\hat{x} - x\|_2^2 \leq \delta$ with probability $1 - e^{-\Omega(k\log(n/k))}$, as long as $m = \Omega(\delta^{-2}k\log(n/k))$.

Algorithm 1 Naive Decoding Algorithm.

$S \leftarrow \emptyset$
for $i \in [n]$ **do**
  if exists no negative test where i participates in **then**
    $S \leftarrow S \cup \{i\}$
  end if
end for
Output S.

We should review some folklore definitions from Combinatorial Group Testing theory. For their proofs one can check [30].

▶ **Definition 4.** A $t \times n$ matrix M is k-disjunct if for every set $S \subset [n]$ with $|S| \leq k, \forall j \notin S, \exists i$ such that $M_{i,j} = 1$ but $\forall k \in S, M_{i,k} = 0$. In other words, $\text{supp}(M_j) - \cup_{l \in S}\text{supp}(M_l) \neq \emptyset$.

▶ **Definition 5.** A $t \times n$ matrix M is (k, l)-disjunct if for every two disjoint sets $S, T \subset [n]$ with $|S| \leq k, |T| = l$, there exists a row i such that $\forall j \in S, M_{i,j} = 0$, but $\exists j \in T, M_{i,j} = 1$.

In the noiseless case, we will make extensive use of the following two lemmas:

▶ **Lemma 6.** *Let M be a k-disjunct matrix. Then, given $y = Mx$, the naive decoding algorithm returns a set S such that $S = \text{supp}(x)$, i.e. the naive decoding algorithm correctly finds the support of x.*

▶ **Lemma 7.** *Let M be a (k, l)-disjunct matrix. Then, given $y = Mx$, the naive decoding algorithm returns a set S such that supp$(x) \subset S$ and $|S| \leq |\text{supp}(x)| + l$, i.e. the naive decoding algorithm finds a superset of the support of x with additional l elements.*

Of course, the two different definitions solve a different problem; the latter one solving a more relaxed version of Group Testing than the former. We will refer to the second version as two-stage group testing, whereas we will refer to the first version just as group testing.

2.1 Formal Statement of Results

▶ **Theorem 8.** *There exists a randomized construction of a scheme* $(\Phi,$ OneBitCS$())$ *which with probability* $1 - \mathcal{O}(\frac{1}{k \log(n/k)} + e^{-k})$ *satisfies the* δ-ℓ_2/ℓ_2 *guarantee. Moreover,* Φ *has* $\mathcal{O}(k \log(n/k)(\log k + \log\log(n/k)) + \delta^{-2}k)$ *rows and* OneBitCS$()$ *runs in time* $\text{poly}(k \log n)$.

▶ **Theorem 9.** *There exists a randomized construction of a scheme* $(\Phi,$ OneBitCS$())$ *such that*

$$\forall x \in \Sigma_k^1, \mathbb{P}[\hat{x} = \text{OneBitCS}(\Phi x), \|x - \hat{x}\|_2^2 > \delta] \leq e^{-k}.$$

The number of rows of Φ *is* $\mathcal{O}(k \log n + \log_k n \log\log_k n + \delta^{-2}k)$ *and the running time of* OneBitCS$()$ *is* $\text{poly}(k, \log n)$.

▶ **Theorem 10.** *There exists a randomized construction of a scheme* $(\Phi,$OneBitCS$())$ *such that*

$$\mathbb{P}[\exists x \in \Sigma_k^1 : \hat{x} = \text{OneBitCS}(\Phi x), \|x - \hat{x}\|_2^2 > \delta] \leq e^{-k \log(n/k)}.$$

The matrix Φ *has* $\mathcal{O}(k^2 \log(n/k) \log\log_k n + \delta^{-6}k \log(n/k))$ *rows and* OneBitCS$()$ *runs in poly*$(k, \log n)$ *time.*

▶ **Theorem 11.** *There exists a deterministic construction of a scheme* $(\Phi, \text{Support}())$*, where* $\text{Support}(\Phi x)$ *outputs a set* L *of at most* k *elements, and* $\forall x \in \Sigma_k^1, \text{Support}(\Phi x) = \text{supp}(x)$*. The matrix* Φ *has* $\mathcal{O}(k^3 \log n)$ *rows and* $\text{Support}()$ *runs in* $\text{poly}(k, \log n)$ *time.*

Complete proofs of these theorems are deferred to the full version.

2.2 Overview of techniques

All of our algorithms find the superset of the support of the vector x (noiseless case) or a small set containing the largest $\mathcal{O}(k)$ in magnitude coordinates (noisy case). This set contains the crucial information needed to approximate x. Then, by restricting to the set obtained, we show how the algorithm from [32] can give us the desired guarantees.

We first treat the noisy case. We sketch our approach to find a set of size $O(k)$ that contains all $\frac{1}{k}$-heavy hitters of our vector x. Let us first discuss the Count-Sketch [7] for finding heavy hitters in data streams (where we also have magnitude information). The Count-Sketch consists of $\log n$ different iterations: in each iteration r we hash every element to $\mathcal{O}(k)$ buckets using a 2-wise independent hash function $h_r : [n] \to [\mathcal{O}(k)]$, combined with random signs. This means that for each pair (r, b) the (r, b)-th measurement is $\sum_{i:h_r(i)=b} \sigma_{i,r} x_i$, where $\sigma_{i,r}$ are pairwise independent random signs. For each coordinate i and iteration r, we read the value of the bucket $h_r(i)$ multiplied by $\sigma_{i,r}$ to get an estimate for x_i. The median of all $\log n$ different estimates is our final estimate for x_i. This approach essentially solves a harder problem called ℓ_∞/ℓ_2 sparse recovery, but at this point we are interested in finding only the heavy hitters of x, not approximating their values; we will use the algorithm of [32] for that later. So, when we only have access to one-bit measurements an immediate approach is the following. Using the same hashing scheme as Count-Sketch, the decoding algorithm for every coordinate i and every iteration r checks if $\sigma_{i,r}$ agrees or disagrees with the sign of the value of the bucket $h_r(i)$, let this be $C_{r,h_{r(i)}}$. If this happens more than $\frac{3}{4}$ of the time we classify i as a heavy hitter. This happens because if i is a heavy hitter, with constant probability x_i will dominate the value of $h_r(i)$. Unfortunately, this does not suffice to give us sublinear decoding time.

What we need is a technique called the dyadic trick, which was introduced in [10], for the ℓ_1 case when all x_i are non-negative. In this case, we form a tree of size depth $\log n$, where level l corresponds to the decomposition of $[n]$ into 2^l equal-sized and disjoint intervals. The algorithm at each step keeps a list of size $\mathcal{O}(k)$ of "active" nodes, that is intervals that contain some heavy hitter. At every level we run a version of the Count-Min Sketch to find the heavy intervals (those that have ℓ_1 mass larger than $\frac{\|x_{\text{tail}(k)}\|_1}{k}$) and proceed by considering their children as active. The algorithm terminates when we reach the last level. Observe that at all times we want to guarantee that the list is of size $\mathcal{O}(k)$ and hence $\mathcal{O}(k \log n)$ nodes are visited in total.

In our case, however, we only have sign information about our measurements and we do not assume that x_i are non-negative. Remember that every node of the tree corresponds to an interval. For any interval I in each level of the tree we add a normal random variable in front of every x_i for $i \in I$ and when we hash nodes to buckets, we combine with a random sign. We explain what this means. Let us focus on some interval/node I. If we hash this node to U buckets using a hash function $h_r : [2^l] \to [U]$, then the contribution of the interval to the value of bucket $h_r(a)$ will be $\sigma_I \cdot \sum_{i \in I} g_i^I x_i$, where g_i^I is the gaussian corresponding to each $i \in I$ and σ_I is the sign associated with I. The idea is that $\sum_{i \in I} g_i^I x_i$ essentially approximates the ℓ_2 mass of the interval and hence one can expect $\sigma_I \cdot \sum_{i \in T} g_i^I x_i$ to be roughly $\sigma_I \cdot \|x_I\|_2$. This would mean that if I contains a heavy hitter, the sign of the

measurement it participates in should roughly be the same as the sign of $\sum_{i\in I} g_i^I x_i$ and hence, by repeating a lot of times, we can hope to clasify this interval as heavy. In this approach, however, there are technical hurdles, one of them being that we should not refresh the gaussians across iterations in the same level otherwise the signs will be uniformly at random. In the next section we show how to take care of all the details.

In the noiseless case, we use techniques and schemes developed in the context of two-stage group testing. More specifically, we show that if we take a (k, k)-disjunct matrix A and replace each non-zero entry with a normal random variable, we get a for-each scheme for identifying a superset S of the support of x. Using this idea and Lemma 7 we can detect a superset S of the support of x. If we also keep in parallel matrix G each entry of which is a normal random variable, we can use Theorem 3 by restricting G on the columns indexed by S to get the result of Theorem 9. We take a similar approach for the for-all version of the problem, namely Theorem 10. In this case we have to suffer an additional multiplicative factor of k in the measurements. More specifically, let $A \in \{0,1\}^{t\times n}$ be a (k, k)-disjunct matrix and let V be a $k \times n$ matrix. Then, the measurement matrix for is the vertical concatenation of $A \otimes V$ concatenated with G. For every i and x, $(a_i \otimes V)x$ can be seen as single test on x: $(a_i \otimes V)x = 0$ if and only if $\mathrm{supp(a_i)} \cap \mathrm{supp(x)} = \emptyset$. Vertically concatenating $(A \otimes V)$ and $((-A) \otimes V)$ we can check whether $(a_i \otimes V)x = 0$ or not. Then, a modification of Algorithm 1 and the for-all theorem of [32] (Result 1 from Table 1) gives us the desired result. For the support recovery problem, our approach is similar: We use a deterministic k-disjunct matrix A guaranteed by [23] and form the vertical concatenation of $A \otimes V$ and $(-A) \otimes V$. A similar reasoning as above, along with Lemma 6 gives the desired result. The reason we make use of a k-disjunct matrix and not a list-disjunct one is because we are interested in finding exactly the support.

3 For-each δ-ℓ_2/ℓ_2 One-Bit Compressed Sensing

As mentioned in the previous section, the algorithm is based on a two-stage approach. The first stage identifies the set S of the "heavy" coordinates of the vector x; these coordinates carry most of the ℓ_2 mass of x and hence the crucial information needed to approximate it. The second stage runs the convex program of [32] with the universe being S instead of $[n]$.

The most important part of the algorithm and essentially our contribution, that enables sublinear decoding time, is the procedure that finds the set S. As mentioned before, we turn our attention to the Count-Sketch and the dyadic trick [7, 10], and show that, with only a constant multiplicative increase in the measurement complexity, we can modify them so that they also work with one-bit measurements. We note that these algorithms appeared in the linear case of the very-relevant problem of finding heavy hitters in data streams.

In what follows, we assume that n is a power of 2. Let C_{-1}, C_0, C_1, C_2 be large enough constants to be defined later. For each l, we consider a partition of $[n]$ to 2^l equal-sized disjoint intervals and we denote by L_l^a the a-th interval in this partition. We also set $\Delta = \frac{1}{C_{-1}k\log n}, \Delta' = \log(\frac{1}{\Delta})$, where C_{-1} is an absolute constant larger than 1.

The sensing matrix Φ is the vertical concatenation of matrices $E^{(\log k)}, E^{(\log k+1)}, \ldots, E^{(\log n)}, A$. The number of rows of each $E^{(l)}$ is $C_0 \cdot C_1 \cdot C_2 \cdot k\Delta'$ and the number of rows of A is $\mathcal{O}(\delta^{-2}k)$. For $\log k \le l \le \log n$ let $E^{(l)}$ be the l-th matrix. $E^{(l)}$ consists of submatrices $E_1^{(l)}, \ldots, E_{C_2\Delta'}^{(l)}$. Each matrix $E_m^{(l)}$ consists of C_1 matrices $E_{m,t}^{(l)}$, $t = 1, \ldots, C_1$. Let $h_{l,m,t} : [2^l] \to [C_0k]$ be a hash function that maps intervals/nodes at level l to C_0k buckets. We define the q-th row of $E_{m,t}^{(l)}$ via its dot product with x:

$$\left\langle e_q^T E_{m,t}^{(l)}, x \right\rangle = \sum_{a:h_{l,m,t}(a)=q} \sigma_{m,t}^{l,a} \sum_{j\in L_l^a} g_{j,m}^{(l)} \cdot x_j,$$

OneBitHeavyHitters(y):

1. $S_{\log k-1} \leftarrow \{L^1_{\log k-1}, L^2_{\log k-1} \dots, L^{k/2}_{\log k-1}\}$
2. For $l = \log k$ to $\log n$:
3. For each L^i_{l-1} in S_{l-1} add L^{2i-1}_l and L^{2i}_l to S_l.
4. For every element L^a_l in S_l
5. If CheckIfHeavy(L^a_l) = 'light' remove L^a_l from S_l
6. Output every x in $S_{\log n}$

Figure 1 Recovery of ℓ_2 Heavy Hitters from One-Bit Measurements.

where $g^{(l)}_{j,m} \sim \mathcal{N}(0,1)$ and $\sigma^{l,a}_{m,t}$ are random signs. The above expression states that in each sketch $E^{(l)}_{m,t}$ we hash the nodes at level l in $C_2 k$ buckets.

In other words, every $E^{(l)}$ holds a hierarchical separation of $[n]$ into 2^l intervals of length $n/2^l$. Fix now some $m \in [C_2\Delta']$. Then, in each $E^{(l)}_{m,t}$, every node/interval is hashed to some bucket and the coordinates inside this interval are combined with standard Gaussians. Moreover, every interval is assigned a random sign. The intuition is that with constant probability, we do expect the term $\sum_{j \in L^a_l} g^{(l)}_{j,m,t} \cdot x_j$, to behave roughly like the l_2 mass of the interval itself. Then, by keeping the same gaussians, we take C_1 such hashing schemes (we refresh only the σ variables and the hash functions). For fixed m, l, this means that we use in total $C_1 C_2 \Delta'$ measurements, $C_2\Delta'$ for each of the C_1 rounds. Let this scheme be called Scheme 1. Then, for each level, we repeat the Scheme 1 $C_2\Delta'$ times, for $m = 1, \dots, C_2\Delta'$. Note now that across E^m_l for different m, l we use new g variables. The reason we have to make this additional repetition, in contrast to the standard dyadic trick, is that we only have sign information and we cannot use fresh gaussians at every measurement, since this would imply uniformity of the signs of the measurements. In other words, we would roughly see half $+1$ and half -1 and we would not be able to distinguish the 'heavy' intervals from the 'light' ones, as we will see next.

The decoding algorithm processes these intervals in increasing l for $l = \log k$ up to $\log n$ and keeps a list of intervals at each time (the list is denoted by S_l in the pseudocode). In the beginning of each step l, every node is hashed to $C_0 k$ buckets. Suppose for a moment, that we have the ℓ_2 mass of each interval and we hash these values, instead, into $C_0 k$ buckets combined with random signs. If an interval I contains a node that is 'heavy' and also is hashed to a bucket b, then we expect that its ℓ_2 mass dominates the ℓ_2 mass of other coordinates hashed to the same bucket. Thus, the sign of the sum must be determined by the sign of the 'heavy' interval. To overcome the fact that we do not have the ℓ_2 mass of the interval (since we can only make use of linear measurements) we add a standard random variable in front of every node in the inteval, before hashing. We exploit the aforementioned intuition, along with 2-stability of the Gaussian distribution , to show that we can identify all "heavy" intervals and that we do not introduce a big number of erroneous intervals (intervals that are not 'heavy'). We repeat this hashing scheme C_1 times with the same gaussians and try to find the intervals whose sign agrees or disagrees with the measurement they participate in most of the time. We consider them good. As mentioned in the previous paragraph, this whole hashing scheme called Scheme 1. Now, we repeat Scheme 1 $C_2\Delta'$ times with completely fresh randomness. We then find the intervals which were consider good at least $\frac{2}{3}C_2\Delta'$ times and add them to a list. At the end of each step l, every interval L^i_l that belongs to the list and is substituted by its two sub-intervals $L^{2i-1}_{l-1}, L^{2i}_{l-1}$.

CheckIfHeavy(L_l^a):

1. isheavy $\leftarrow 0$
2. For $m = 1$ to $C_2\Delta'$
3. cnt $\leftarrow 0$
4. For $t = 1$ to C_1
5. $y_q \leftarrow$ value of bucket $h_{l,m,t}(a)$
6. If $\text{sign}(y_q) = \sigma_{m,t}^{l,a}$
7. cnt $\leftarrow$ cnt $+ 1$
8. If cnt $> 0.8C_1$ *or* cnt $< 0.2C_1$
9. isheavy $\leftarrow$ isheavy $+ 1$
10. If isheavy $> \frac{2}{3}C_2\Delta'$
11. return 'heavy', else return 'light'

Figure 2 Check if an Interval at a Specific Level is Heavy.

OneBitCS(y):

1. $S \leftarrow$ OneBitHeavyHitters(y).
2. $\hat{x} = \text{argmax}\,\langle y, A_S z\rangle$ subject to $\|z\|_2 \leq 1, \|z\|_1 \leq \sqrt{k}$ (Algorithm of [32]).
3. Output $\hat{x}$.

Figure 3 One-Bit Compressed Sensing.

▶ **Definition 12.** For a coordinate i and a level l, let $b^l(i)$ be such that $i \in L_l^{b^l(i)}$. If the level l is implicit, we may simplify the notation to $b(i)$. In other words, $b^l(i)$ is the interval on level l which contains i.

Fix some matrix $E_{m_0}^l$ an an interval I in level l. Then, we will say that $E_{m_0}^l$ classifies interval I as good, if the variable isheavy is incremented in the execution of CheckIfHeavy(I) when $m = m_0$. Intuitively, $E_{m_0}^l$ classifies I as good if I appears to contain a heavy hitter in it. The next two lemmas are crucial components of our proof.

▶ **Lemma 13.** *Let C_h be an absolute constant. Fix m, l. Let $i \in [n]$ such that $|x_i|^2 > \frac{1}{C_h k}\|x_{\text{tail}(k)}\|_2^2$. Let I an interval at level l. Assume that $L_l^{b(i)}, I \in S_l$. Then, for some absolute constant c, the following claims hold:*

- *$E_m^{(l)}$ will classify $L_l^{b(i)}$ as good with constant probability, strictly larger than $\frac{1}{2}$.*
- *If there are at least ck intervals at the same level l which have greater ℓ_2 mass than I, then, with constant probabiliy, $E_m^{(l)}$ will not classify I as good.*

The proof of the aforementioned lemma is deferred to the full version.

▶ **Lemma 14.** *Let $S =$ OneBitHeavyHitters(y). then, with probability $1 - \mathcal{O}(\frac{1}{k\log(n/k)})$, for all $\log k \leq l \leq \log n$, the following holds for the set S_l:*

- *If $|x_i|^2 > \frac{1}{C_h k}\|x_{\text{tail}(k)}\|_2^2$ and $i \in L_l^{b(i)}$, then $i \in S$.*
- *$|S_l| \leq ck$, for some absolute constant c.*

Proof. For the proof of this lemma, we need to introduce some additional definitions. We will refer to any interval that contains a node i such that $|x_i|^2 > \frac{1}{10k}\|x_{\text{tail}(k)}\|_2^2$, as a type 1 interval. For a level l, we say that an interval at level l is of type 2, if there exist at least ck intervals at the same level that have greater ℓ_2 mass than this interval.

We now proceed by induction on the number of levels. The base case $l = \log k$ is trivial. We focus on some level l and assume that the induction hypothesis holds for all previous levels l. We prove the first bullet. Let i be a coordinate such that $|x_i|_2^2 > \frac{1}{C_h k}\|x_{\text{tail}(k)}\|_2^2$. By the induction hypothesis we get that $L_l^{b^l(i)} \in S_l$. From Lemma 14 we know that for any l, m, E_l^m will classify a type 1 interval as good with constant probability $> \frac{2}{3}$. Moreover, it will not classify any type 2 interval as good, again with constant probability $> \frac{2}{3}$. This implies that after repeating the same scheme $C_2\Delta' = C_2 \log(\frac{1}{\Delta})$ times, we will know, with probability at least $1 - \Delta$, if a specific interval is a type-1 interval or a type-2 interval or none of these. Because $\Delta = \Theta(\frac{1}{k \log(n/k)})$ we can take a union-bound over all possible intervals we might consider ($\mathcal{O}(k)$ at each of the $\log(n/k)$ levels), we can guarantee that every type-1 interval will remain in S_l, while any type-2 interval will be discarded from S_l. This implies that at any step we have at most ck intervals in S_l with every type-1 inteval belonging to S_l. ◀

We are now ready to prove the main result of this section.

Proof. By running OneBitCS(y), we obtain a set S that satisfies the guarantees of Lemma 14. Then, $y = \text{sign}(Ax) = \text{sign}(Ax_S + Ax_{[n]-S}) = \text{sign}(A_S x_S + v)$, where v is a vector each entry of which follows normal distribution with variance $\|x_{[n]-S}\|_2^2 \leq 1$. Clearly, A_S and v are independent and hence Theorem 3 applies. The number of rows needed equals $\Omega(\delta^{-2} k \log(ck/k)) = \Omega(\delta^{-2}k)$. The convex program of 3 outputs a vector $\hat{x}$ such that $\|\hat{x} - x_S\|_2^2 \leq \delta$. Since every coordinate i with $|x_i|^2 \geq \frac{1}{C_h k}\|x_{\text{tail}(k)}\|_2^2$ is contained in S,

$$\|x_{[n]-S}\|_2^2 \leq \|x_{\text{tail}(k)}\|_2^2 + ck\frac{1}{C_h k}\|x_{\text{tail}(k)}\|_2^2 = (1 + \frac{c}{C_h})\|x_{\text{tail}(k)}\|_2^2.$$

This implies that $\|x - \hat{x}\|_2^2 \leq (1 + \frac{c}{C_h})\|x_{\text{tail}(k)}\|_2^2 + \delta$, as desired. ◀

Acknowledgements. The author would like to thank Ely Porat for pointing him to [30], as well as Jelani Nelson for helpful discussions.

References

1 Rudolf Ahlswede, Lars Bäumer, Ning Cai, Harout K. Aydinian, Vladimir Blinovsky, Christian Deppe, and Haik Mashurian, editors. *General Theory of Information Transfer and Combinatorics*, volume 4123 of *Lecture Notes in Computer Science*. Springer, 2006.

2 Richard Baraniuk, Simon Foucart, Deanna Needell, Yaniv Plan, and Mary Wootters. Exponential decay of reconstruction error from binary measurements of sparse signals. *arXiv preprint arXiv:1407.8246*, 2014.

3 Petros Boufounos and Richard G. Baraniuk. 1-bit compressive sensing. In *42nd Annual Conference on Information Sciences and Systems, CISS 2008, Princeton, NJ, USA, 19-21 March 2008*, pages 16–21, 2008. doi:10.1109/CISS.2008.4558487.

4 E. J. Candes and T. Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *Information Theory, IEEE Transactions on*, 52(12):5406–5425, Dec 2006. doi:10.1109/TIT.2006.885507.

5 Emmanuel Candes, Mark Rudelson, Terence Tao, and Roman Vershynin. Error correction via linear programming. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 668–681. IEEE, 2005.

6 Fei-Huang Chang, Huilan Chang, and Frank K. Hwang. Pooling designs for clone library screening in the inhibitor complex model. *J. Comb. Optim.*, 22(2):145–152, 2011. doi:10.1007/s10878-009-9279-9.

7 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

8 Hong-Bin Chen and Frank K. Hwang. A survey on nonadaptive group testing algorithms through the angle of decoding. *J. Comb. Optim.*, 15(1):49–59, 2008. doi:10.1007/s10878-007-9083-3.

9 Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.

10 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

11 Annalisa De Bonis, Leszek Gasieniec, and Ugo Vaccaro. Optimal two-stage algorithms for group testing problems. *SIAM Journal on Computing*, 34(5):1253–1270, 2005.

12 David L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006. doi:10.1109/TIT.2006.871582.

13 Robert Dorfman. The detection of defective members of large populations. *Ann. Math. Statist.*, 14(4):436–440, 12 1943. doi:10.1214/aoms/1177731363.

14 Ding-Zhu Du and Frank K. Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 1999.

15 Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. Approximate sparse recovery: optimizing time and measurements. *SIAM Journal on Computing*, 41(2):436–453, 2012.

16 Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. For-all sparse recovery in near-optimal time. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 538–550, 2014. doi:10.1007/978-3-662-43948-7_45.

17 Anna C. Gilbert, Hung Q. Ngo, Ely Porat, Atri Rudra, and Martin J. Strauss. l2/l2-foreach sparse recovery with low risk. In *Automata, Languages, and Programming*, pages 461–472. Springer, 2013.

18 Anna C. Gilbert, Martin J. Strauss, Joel A. Tropp, and Roman Vershynin. One sketch for all: fast algorithms for compressed sensing. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 237–246. ACM, 2007.

19 Sivakant Gopi, Praneeth Netrapalli, Prateek Jain, and Aditya Nori. One-bit compressed sensing: Provable support and vector recovery. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 154–162, 2013.

20 Vivek K. Goyal, Martin Vetterli, and Nguyen T. Thao. Quantized overcomplete expansions in IR N: analysis, synthesis, and algorithms. *Information Theory, IEEE Transactions on*, 44(1):16–31, 1998.

21 C Sinan Güntürk, Mark Lammers, Alex Powell, Rayan Saab, and Özgür Yilmaz. Sigma delta quantization for compressed sensing. In *Information Sciences and Systems (CISS), 2010 44th Annual Conference on*, pages 1–6. IEEE, 2010.

22 Ankit Gupta, Robert D. Nowak, and Benjamin Recht. Sample complexity for 1-bit compressed sensing and sparse classification. In *ISIT*, pages 1553–1557, 2010.

23 Piotr Indyk, Hung Q. Ngo, and Atri Rudra. Efficiently decodable non-adaptive group testing. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'10, pages 1126–1142, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1873601.1873692.

24 L. Jacques, J. N. Laska, P. T. Boufounos, and R. G. Baraniuk. Robust 1-bit compressive sensing via binary stable embeddings of sparse vectors. *Information Theory, IEEE Transactions on*, 59(4):2082–2102, April 2013. doi:10.1109/TIT.2012.2234823.

25 Laurent Jacques, Jason N. Laska, Petros T. Boufounos, and Richard G. Baraniuk. Robust 1-bit compressive sensing via binary stable embeddings of sparse vectors. *IEEE Transactions on Information Theory*, 59(4):2082–2102, 2013. doi:10.1109/TIT.2012.2234823.

26 Raghunandan M. Kainkaryam, Angela Bruex, Anna C. Gilbert, John Schiefelbein, and Peter J. Woolf. poolMC: Smart pooling of mRNA samples in microarray experiments. *BMC Bioinformatics*, 11:299, 2010. doi:10.1186/1471-2105-11-299.

27 Felix Krahmer, Rayan Saab, and Özgür Yilmaz. Sigma–delta quantization of sub-gaussian frame expansions and its application to compressed sensing. *Information and Inference*, page iat007, 2014.

28 J. N. Laska, Zaiwen Wen, Wotao Yin, and R. G. Baraniuk. Trust, but verify: Fast and accurate signal recovery from 1-bit compressive measurements. *Signal Processing, IEEE Transactions on*, 59(11):5289–5301, Nov 2011. doi:10.1109/TSP.2011.2162324.

29 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005. doi:10.1561/0400000002.

30 Hung Ngo, Ely Porat, and Atri Rudra. Efficiently decodable error-correcting list disjunct matrices and applications. *Automata, languages and programming*, pages 557–568, 2011.

31 Yaniv Plan and Roman Vershynin. One-bit compressed sensing by linear programming. *Communications on Pure and Applied Mathematics*, 66(8):1275–1297, 2013.

32 Yaniv Plan and Roman Vershynin. Robust 1-bit compressed sensing and sparse logistic regression: A convex programming approach. *Information Theory, IEEE Transactions on*, 59(1):482–494, 2013.

33 Ely Porat and Martin J. Strauss. Sublinear time, measurement-optimal, sparse recovery for all. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1215–1227. SIAM, 2012.

34 A. M. Rashad. Random coding bounds on the rate for list-decoding superimposed codes. *Problems of Control and Information Theory – Problemy Upravleniya i Teorii Informatsii*, 19(2):141–149, 1990.