Compressed Sensing Using Binary Matrices of Nearly Optimal Dimensions

Amitesh Shekhar
IIT Bombay
22b0014@iitb.ac.in

Anupam Rawat
IIT Bombay
22b3982@iitb.ac.in

Toshan Achintya Golla
IIT Bombay
22b2234@iitb.ac.in

March 25, 2025

Contents

1 Introduction to Compressive Sensing

Compressive Sensing (CS) is a signal processing framework that enables the reconstruction of sparse or compressible signals from a small number of linear measurements, often fewer than traditionally required by the Nyquist sampling theorem. The core idea is that many natural signals (e.g., images, audio, or medical signals) have a sparse representation in some basis or dictionary, meaning that most of their coefficients are zero or negligible.

Mathematically, suppose we want to recover a signal $x \in \mathbb{R}^n$, which is known to be sparse, from linear measurements:

$$y = Ax,$$

where $y \in \mathbb{R}^m$ is the measurement vector, and $A \in \mathbb{R}^{m \times n}$ is the measurement matrix with $m \ll n$. Since the system is under-determined, it has infinitely many solutions. However, if x is sparse (i.e., it has only $s \ll n$ non-zero entries), we can aim to recover the sparsest solution by solving:

$$\min ||x||_0 \text{ subject to } y = Ax.$$

Because the l_0-norm minimization is NP-hard, it is commonly replaced by the convex relaxation:

$$\min ||x||_1 \text{ subject to } y = Ax,$$

which can be solved efficiently using linear programming or iterative methods.

Compressive sensing has found applications in MRI imaging, astronomy, compressive photography, and even wireless sensor networks, where data acquisition and transmission costs must be minimized.

Measurement Matrices
The measurement matrix $\mathbb{A}$ is a critical component of the CS framework. It determines how the high-dimensional signal is projected into a lower-dimensional measurement space. To guarantee successful recovery of sparse signals, the matrix must satisfy certain mathematical properties.

The most celebrated of these is the **Restricted Isometry Property (RIP)**, introduced by *Candès and Tao*. A matrix $\mathbb{A}$ satisfies the RIP of order s if there exists a constant $\delta_s \in (0, 1)$ such that for all s-sparse vectors x:

$$(1 - \delta_s)||x||_2^2 \leq ||Ax||_2^2 \leq (1 + \delta_s)||x||_2^2$$

This condition ensures that $\mathbb{A}$ preserves the geometry of sparse signals and prevents them from being projected into indistinguishable measurements.

Another important concept is **incoherence**, which measures the correlation between the rows of $\mathbb{A}$ and the basis in which the signal is sparse. Low coherence ensures that each measurement captures a distinct aspect of the signal.

Random matrices with i.i.d. Gaussian or Bernoulli entries have been shown to satisfy RIP with high probability when **m=O(slog(n/s))**. However, these matrices are dense and require significant memory and computational resources to store and use, which becomes a bottleneck in practical systems.

Motivation for Binary Matrices Despite the theoretical success of random dense matrices, they pose serious challenges in implementation, particularly in embedded, real-time, or energy-constrained environments. This motivates the study of binary matrices, where the entries of $\mathbb{A}$ are restricted to 0 or 1.

There are various benefits of choosing a Binary Matrix over the Real Valued Matrices. Firstly, they offer a **Simplified hardware design** since they are inherently simpler to implement in hardware using logic gates and switches. Secondly, Binary matrices can be stored using **minimal memory**, and operations like matrix-vector multiplication are significantly faster. Further, due to the above factors, they become significant in portable devices due to their lower energy consumption.

However, the use of binary matrices introduces new mathematical difficulties. Unlike Gaussian or Bernoulli matrices, binary matrices often fail to satisfy the RIP unless their design is carefully constrained. For this

reason, much of the research has focused on structured binary matrices, such as those derived from expander graphs, which retain good recovery guarantees despite lacking traditional RIP.

The paper "*Compressed Sensing Using Binary Matrices of Nearly Optimal Dimensions*" directly addresses this challenge. It constructs binary matrices with provable recovery guarantees using techniques from combinatorics and graph theory, offering a bridge between theoretical performance and practical usability.

2 Summary of the Paper and Theoretical Contributions

The paper *"Compressed Sensing Using Binary Matrices of Nearly Optimal Dimensions"* introduces a novel framework for compressed sensing that leverages sparse binary matrices constructed from unbalanced bipartite expander graphs. This work bridges the gap between theoretical optimality and practical feasibility, offering a memory-efficient and computation-friendly alternative to Gaussian matrices, without compromising on recovery guarantees. Below, we outline the key contributions of the paper.

The authors employ *unbalanced bipartite expander graphs* to construct binary measurement matrices. Consider a bipartite graph with n left nodes (signal components) and m right nodes (measurements). Each left node is connected to exactly d right nodes, forming a left d-regular graph. The adjacency matrix $A \in \{0,1\}^{m\times n}$ derived from this graph serves as the sensing matrix. Such matrices inherently possess the *expansion property*, ensuring that small subsets of columns influence many distinct rows, which is crucial for sparse signal recovery.

Although these matrices do not satisfy the traditional Restricted Isometry Property (RIP), the authors prove that they offer robust and accurate recovery based on expansion properties. Specifically, if A corresponds to a (k,ϵ)-expander with $\epsilon < \frac{1}{6}$, then every k-sparse signal x is the unique minimizer of the ℓ_1-minimization problem. Furthermore, in the presence of noise or when the signal is only approximately sparse, the paper establishes the following recovery bound:

$$\|x - \hat{x}\|_1 \leq C \cdot \min_{x_k} \|x - x_k\|_1 + D \cdot \|e\|_1,$$

where x_k is the best k-sparse approximation of x, and e denotes the noise vector. Constants C and D depend on the graph parameters.

The proposed binary expander matrices offer several advantages over traditional Gaussian or Bernoulli matrices, especially in practical settings. A comparative summary is provided in Table 1.

Table 1: Comparison of Binary Expander Matrices with Gaussian Matrices

Property	**Binary Expander**	**Gaussian/Bernoulli**
Entries	0 or 1	Real-valued
Sparsity	Sparse	Dense
Construction	Deterministic/Pseudorandom	Fully Random
Memory Efficiency	High	Low
RIP Satisfaction	No	Yes (w.h.p.)
Recovery Method	Expansion-based	RIP-based
Hardware Suitability	Excellent	Moderate

The use of sparse binary matrices in compressed sensing introduces several practical benefits like Memory Efficiency, Fast Computation and Scalability. Additionally, the compatibility of these matrices with greedy and iterative algorithms (e.g., message-passing or expander-based decoding) makes them ideal for low-power embedded applications.

3 Mathematical Analysis and Proofs

Theorem 1 : Robust Sparse Recovery under RIP Conditions
Let $A \in \mathbb{R}^{m\times n}$ be a measurement matrix satisfying the Restricted Isometry Property (RIP) of order tk with constant δ_{tk}. Then the following holds:

$$\delta_{tk} < \begin{cases} \frac{t-1}{t}, & \text{for } t \geq \frac{4}{3}, \\ \frac{t}{4-t}, & \text{for } 0 < t < \frac{4}{3}, \end{cases} \quad \Rightarrow \quad (A, \Delta_{\text{BP}}) \text{ achieves robust recovery of } k\text{-sparse signals.}$$

Moreover, these bounds are tight and coincide at $t = \frac{4}{3}$, yielding a continuous threshold on δ_{tk} for all $t > 0$.

Theorem 2 : RIP Satisfaction by Gaussian Matrices
Let $\mathbb{A} = \frac{1}{\sqrt{m}}\phi$ where $\phi \in \mathbb{R}^{m\times n}$ consists of independent samples of a standard Gaussian random variable. Define

$$g = 1 + \frac{1}{2}ln(\frac{en}{k}), \eta = \frac{\sqrt{1+\delta}-1}{g}$$

Then the matrix $\mathbb{A}$ satisfies the Restricted Isometric Property (RIP) or order k with constant δ and with probability at least $1-\xi$, provided that

$$m \geq \frac{2}{\eta^2}\left(kln\left(\frac{en}{k}\right) + ln\left(\frac{2}{\xi}\right)\right)$$

Theorem 3 : Lower Bound on Measurements via Information-Theoretic Argument
Suppose $\mathbb{A} \in \mathbb{R}^{m\times n}$, and there exists a recovery map $\Delta : \mathbb{R}^m \to \mathbb{R}^n$ such that the pair $(\mathbb{A}, \Delta)$ achieves stable k-sparse recovery with constant C. Define the sparsity ratio $\theta = \frac{n}{k}$. Then a necessary condition for such recovery is:

$$m \geq \frac{1 - H_\theta(\frac{1}{2})}{ln(4+2C)} \cdot kln(\theta)$$

where $H_\theta(p) = p \cdot log_\theta(\theta - 1) - p \cdot log_\theta(p) - (1-p)log_\theta(1-p)$ is the base-θ entropy function
Comparing Theorems 2 and 3 shows that $\mathbf{m = O(kln(n/k))}$ measurements are both necessary and sufficient for robust k-sparse recovery.

Theorem 4 : Stable Recovery via the Stable Null Space Property
Suppose the matrix $A \in \mathbb{R}^{m\times n}$ satisfies the *Stable Null Space Property (SNSP)* of order k with constant $\rho \in (0,1)$. That is, for all $v \in \ker(A)$, the following inequality holds:

$$\|v_S\|_1 \leq \rho\|v_{S^c}\|_1 \quad \text{for all } S \subseteq [n], |S| \leq k.$$

Then, the pair (A, Δ_{BP}), where Δ_{BP} denotes recovery via Basis Pursuit, achieves stable k-sparse recovery. Specifically, for any $x \in \mathbb{R}^n$, the recovery estimate $\hat{x} = \Delta_{\text{BP}}(Ax)$ satisfies:

$$\|\hat{x} - x\|_1 \leq C \cdot \sigma_k(x)_1,$$

where $\sigma_k(x)_1$ is the best k-term approximation error in the ℓ_1-norm, and the constant is given by:

$$C = \frac{2(1+\rho)}{1-\rho}.$$

Theorem 5 : Robust Recovery via the Robust Null Space Property
Suppose the matrix $A \in \mathbb{R}^{m \times n}$ satisfies the *Robust Null Space Property (RNSP)* of order k with respect to a norm $\|\cdot\|$, with constants $\rho \in (0,1)$ and $\tau > 0$. That is, for all $v \in \mathbb{R}^n$,

$$\|v_S\|_1 \leq \rho \|v_{S^c}\|_1 + \tau \|Av\| \quad \text{for all } S \subseteq [n], |S| \leq k.$$

Then, the pair (A, Δ_{BP}), with Basis Pursuit decoder Δ_{BP}, achieves robust k-sparse recovery. That is, for any signal $x \in \mathbb{R}^n$ and noisy measurement $y = Ax + e$, the recovery estimate $\hat{x} = \Delta_{\text{BP}}(y)$ satisfies:

$$\|\hat{x} - x\|_1 \leq C \cdot \sigma_k(x)_1 + D \cdot \|e\|,$$

with constants

$$C = \frac{2(1+\rho)}{1-\rho}, \quad D = \frac{4\tau}{1-\rho}.$$

Theorem 6 : Robust Recovery via Coherence-Based Condition
Let $A \in \mathbb{R}^{m \times n}$ be a measurement matrix with mutual coherence μ, defined as:

$$\mu = \max_{i \neq j} \frac{|\langle a_i, a_j \rangle|}{\|a_i\|_2 \|a_j\|_2},$$

where a_i and a_j are the columns of A. Suppose the signal recovery is performed using Basis Pursuit, denoted by Δ_{BP}. Then the pair (A, Δ_{BP}) achieves robust k-sparse recovery if the coherence satisfies:

$$\left(\frac{3}{2}k - 1\right)\mu < \frac{1}{3}.$$

This condition can equivalently be expressed as a bound on the sparsity level:

$$k < \frac{2}{3\mu} + \frac{2}{3}.$$

Theorem 7 : Bounds on Null Space of Left-Regular Matrix
Suppose $A \in \{0,1\}^{m \times n}$ is a left-regular matrix with left degree d_L, and suppose that the maximum inner product between any two columns of A is λ. Then, for every $v \in \mathbb{R}^n$ such that $v \in \ker(A)$, we have:

$$|v_i| \leq \frac{\lambda}{2d_L} \|v\|_1, \quad \forall i \in [n],$$

where $[n] = \{1, \ldots, n\}$. This bound holds for all i in the index set.

Remarks:

1. If the matrix A has girth (the length of the shortest cycle in the associated bipartite graph) six or more, then the maximum inner product between any two columns of A is at most 1. Therefore, the above bound becomes:
$$|v_i| \leq \frac{1}{2d_L} \|v\|_1, \quad \forall i \in [n].$$

2. If the girth is at least 10, this bound can be improved further, as demonstrated in the next theorem.

An Important Result: Relating Girth and Coherence **Claim.** If the matrix $A \in \{0,1\}^{m \times n}$ is the biadjacency matrix of a bipartite graph with girth at least 6, then the inner product between any two distinct columns is at most 1.

Proof. Let A be the biadjacency matrix of a bipartite graph $G = (L \cup R, E)$, where each column of A corresponds to a vertex in R and each row to a vertex in L. An entry $A_{ij} = 1$ indicates an edge between $l_i \in L$ and $r_j \in R$.

Let a_i and a_j be two distinct columns of A. Then $\langle a_i, a_j \rangle$ equals the number of rows (i.e., vertices in L) where both a_i and a_j have a 1, i.e., the number of common neighbors of r_i and r_j.

Suppose $\langle a_i, a_j \rangle \geq 2$, i.e., r_i and r_j share at least two common neighbors l_1 and l_2. Then there exists the following cycle in the bipartite graph:

$$r_i \to l_1 \to r_j \to l_2 \to r_i$$

This forms a 4-cycle, contradicting the assumption that the girth is at least 6.

Therefore, $\langle a_i, a_j \rangle \leq 1$ for all $i \neq j$.

□

Theorem 8 : Improved Bound on Null Space with Girth at Least 6
Statement:
Suppose $A \in \{0,1\}^{m \times n}$ has girth $g \geq 6$. Then, for every $v \in \mathbb{R}^n$ such that $v \in \ker(A)$, we have the following bound:

$$|v_i| \leq \frac{\|v\|_1}{C}, \quad \forall i \in [n],$$

where C depends on the girth g. Specifically:

- If $g = 4t + 2$, then:
$$C := 2^t \prod_{i=0}^{t-1} (d_L - 1)^i.$$
- If $g = 4t$, then:
$$C := 2^{t-1} \prod_{i=0}^{t-1} (d_L - 1)^i.$$

Remarks:

1. When the girth $g = 6$ ($t = 1$), the constant C becomes $C = 2$, and the bound in this theorem becomes identical to that in Theorem 7, after noting that $\lambda = 1$.
2. If $g = 8$ ($t = 2$), then C simplifies to $C = 2$ as well. Therefore, Theorem 8 improves over Theorem 7 only when the girth $g \geq 10$.

Implications:

- The bounds provided in Theorems 7 and 8 are crucial for deriving sufficient conditions for a matrix A to satisfy the Stable Null Space Property (SNSP).
- They are also used to infer the Robust Null Space Property (RNSP) of A, providing robust recovery guarantees even in the presence of noise.
- This is a significant improvement over SNSP, as the robust version guarantees recovery under noisy measurements, which is not necessarily true for the stable version alone.
- These results are vital in showing that Basis Pursuit can achieve robustness in noisy settings.

Lemma : Null Space Property for Matrices Satisfying a Bound on Elements
Statement:
Suppose $A \in \mathbb{R}^{m\times n}$ and let $\|\cdot\|$ be any norm on $\mathbb{R}^m$. Assume there exist constants $\alpha > 2$ and $\beta > 0$ such that for all $i \in [n]$ and for all $h \in \mathbb{R}^n$, the following inequality holds:

$$|h_i| \leq \alpha\|h\|_1 + \beta\|Ah\|, \quad \forall i \in [n], \forall h \in \mathbb{R}^n. \tag{22}$$

Then, for any $k < \frac{\alpha}{2}$, the matrix A satisfies the Robust Null Space Property (RNSP) of order k. Specifically, for any subset $S \subseteq [n]$ with $|S| \leq k$, the following inequality holds:

$$\|h_S\|_1 \leq \frac{k}{\alpha - k}\|h_{S^c}\|_1 + \frac{\alpha k\beta}{\alpha - k}\|Ah\|, \tag{23}$$

where h_S denotes the vector of entries of h indexed by S, and S^c is the complement of S in $[n]$.

Proof:
Let $S \subseteq [n]$ be an arbitrary subset with $|S| \leq k$. We begin by analyzing the norm $\|h_S\|_1$ for $h \in \mathbb{R}^n$. We have:

$$\|h_S\|_1 = \sum_{i\in S} |h_i| \leq \sum_{i\in S} \left(\alpha\|h\|_1 + \beta\|Ah\|\right),$$

where the inequality follows from (22). Expanding this sum:

$$\|h_S\|_1 \leq k\alpha\|h\|_1 + k\beta\|Ah\|.$$

Using the decomposition $\|h\|_1 = \|h_S\|_1 + \|h_{S^c}\|_1$, we get:

$$\|h_S\|_1 \leq k\alpha(\|h_S\|_1 + \|h_{S^c}\|_1) + k\beta\|Ah\|.$$

Rearranging terms:

$$(1 - k\alpha)\|h_S\|_1 \leq k\alpha\|h_{S^c}\|_1 + k\beta\|Ah\|.$$

Since $k < \frac{\alpha}{2} < \frac{1}{\alpha}$, we can divide both sides by $(1 - k\alpha)$ (which is positive) to obtain:

$$\|h_S\|_1 \leq \frac{k\alpha}{1 - k\alpha}\|h_{S^c}\|_1 + \frac{k\beta}{1 - k\alpha}\|Ah\|.$$

Finally, multiplying numerator and denominator by -1 gives the desired result:

$$\|h_S\|_1 \leq \frac{k}{\alpha - k}\|h_{S^c}\|_1 + \frac{\alpha k\beta}{\alpha - k}\|Ah\|.$$

Thus, we conclude that the matrix A satisfies the RNSP with the given constants. □

Remarks:

1. The result shows that the Robust Null Space Property (RNSP) holds under the condition that the matrix A satisfies the bound in equation (22). This property is crucial for ensuring stable recovery of sparse signals in compressed sensing.

2. The constants α and β determine the quality of the recovery process. The term $\frac{k}{\alpha-k}$ provides a measure of how the sparsity of the signal (given by k) affects the recovery.

3. The condition $k < \frac{\alpha}{2}$ ensures that the denominator $\alpha - k$ remains positive and the bounds are meaningful.

An Important Result Let $A \in \mathbb{R}^{m \times n}$ have the singular value decomposition (SVD):

$$A = U\Sigma V^\top,$$

where

- $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices,
- $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, where $r = \text{rank}(A)$.

The null space of A, denoted $\mathcal{N}(A)$, consists of all vectors $x \in \mathbb{R}^n$ such that $Ax = 0$. The orthogonal complement $\mathcal{N}^\perp = [\mathcal{N}(A)]^\perp$ is spanned by the first r columns of V, corresponding to nonzero singular values.

Let $u \in \mathcal{N}^\perp$. Then we can write:

$$u = V_r z,$$

where $V_r \in \mathbb{R}^{n \times r}$ contains the first r columns of V, and $z \in \mathbb{R}^r$.

Now compute Au:

$$Au = AV_r z = U\Sigma V^\top V_r z = U\Sigma_r z,$$

where $\Sigma_r \in \mathbb{R}^{r \times r}$ is the diagonal matrix of nonzero singular values.

Then:

$$\|Au\|_2 = \|U\Sigma_r z\|_2 = \|\Sigma_r z\|_2 \quad \text{(since } U \text{ is orthogonal).}$$

$$\|u\|_2 = \|V_r z\|_2 = \|z\|_2 \quad \text{(since } V_r \text{ has orthonormal columns).}$$

Therefore:

$$\|Au\|2 = \|\Sigma_r z\|_2 \geq \sigma\text{min}\|z\|2 = \sigma\text{min}\|u\|_2,$$

which gives:

$$\|u\|2 \leq \frac{1}{\sigma\text{min}}\|Au\|_2.$$

Theorem 9 : RNSP for Left-Regular Binary Matrices
Let $A \in \{0,1\}^{m \times n}$ be a left-regular binary matrix with left degree d_L, and let λ denote the maximum inner product between any two columns of A (so $\lambda \leq d_L$). Let $\sigma_{\min}$ denote the smallest nonzero singular value of A, and let $\|\cdot\|$ be any norm on $\mathbb{R}^m$, with constant c such that:

$$\|y\|_2 \leq c\|y\|, \quad \forall y \in \mathbb{R}^m. \tag{24}$$

Then, the matrix A satisfies the condition (22) with:

$$\alpha = \frac{2d_L}{\lambda}, \quad \beta = \left(\frac{\lambda}{2d_L} + 1\right)\frac{c\sqrt{n}}{\sigma_{\min}}. \tag{26}$$

Consequently, for all $k < \alpha/2 = \frac{d_L}{\lambda}$, the matrix A satisfies the RNSP of order k with:

$$\rho = \frac{\lambda k}{2d_L - \lambda k}, \quad \tau = \frac{2d_L k}{2d_L - \lambda k} \cdot \beta. \tag{27}$$

Proof:
Let $h \in \mathbb{R}^n$ be arbitrary. Decompose it as:

$$h = v + u, \quad \text{where } v \in \mathcal{N}(A), u \in \mathcal{N}^\perp.$$

For each coordinate $i \in [n]$:

$$|h_i| = |v_i + u_i| \leq |v_i| + |u_i|.$$

Bounding $|v_i|$:
From Theorem 7, which analyzes left-regular binary matrices:

$$|v_i| \leq \frac{\lambda}{2d_L}\|v\|_1.$$

Using $\|v\|_1 \leq \|h\|_1 + \|u\|_1$ and the inequality:

$$\|u\|_1 \leq \sqrt{n}\|u\|_2 \leq \frac{c\sqrt{n}}{\sigma_{\min}}\|Au\| = \frac{c\sqrt{n}}{\sigma_{\min}}\|Ah\|,$$

we get:

$$|v_i| \leq \frac{\lambda}{2d_L}\left(\|h\|_1 + \frac{c\sqrt{n}}{\sigma_{\min}}\|Ah\|\right).$$

Bounding $|u_i|$:
Using $\|u\|_1 \leq \frac{c\sqrt{n}}{\sigma_{\min}}\|Ah\|$, we get:

$$|u_i| \leq \|u\|_1 \leq \frac{c\sqrt{n}}{\sigma_{\min}}\|Ah\|.$$

Combining both bounds:

$$|h_i| \leq |v_i| + |u_i| \leq \frac{\lambda}{2d_L}\|h\|_1 + \left(\frac{\lambda}{2d_L} + 1\right)\frac{c\sqrt{n}}{\sigma_{\min}}\|Ah\|.$$

Thus, inequality (22) is satisfied with:

$$\alpha = \frac{2d_L}{\lambda}, \quad \beta = \left(\frac{\lambda}{2d_L} + 1\right)\frac{c\sqrt{n}}{\sigma_{\min}}.$$

Applying Lemma 2, we get the RNSP constants:

$$\rho = \frac{\lambda k}{2d_L - \lambda k}, \quad \tau = \frac{2d_L k}{2d_L - \lambda k} \cdot \beta,$$

as in equation (27). □

Remarks:

1. The bound $|u_i| \leq \|u\|_1$ is tight but conservative, used primarily to estimate β.
2. Robust sparse recovery is guaranteed when $\rho < 1$, i.e., $k < \frac{d_L}{\lambda}$.
3. The constant τ affects the noise amplification in recovery but does not impact the feasibility of recovery per se.

Theorem 10 : RNSP for Left-Regular Binary Matrices with Girth ≥ 6
Let $\mathbb{A} \in \{0,1\}^{m\times n}$ be a left-regular binary matrix with left degree d_L and girth at least 6 (i.e., the shortest cycle in the bipartite graph associated with A has length 6). Then, for all:

$$k < \frac{C}{2}$$

the matrix A satisfies the **Robust Null Space Property** of order k , with constraints:

$$\rho = \frac{k}{C-k}, \ \tau = \frac{C-k}{Ck} \cdot \beta$$

Theorem 10 relies on the girth of the bipartite graph corresponding to matrix A. Intuitively, as the girth increases, the constant C appearing in the RNSP bounds also increases (as shown in Theorem 8). This has a direct consequence, i.e. As the girth of matrix A increases, so does the upper bound on the sparsity level k for which robust recovery is guaranteed.

Theorem 11 : Lower bound on measurements for left-regular graphs
Suppose $A \in \{0,1\}^{m \times n}$ is a d_L-left-regular matrix with $m \leq n$, and that every row and column of A contains at least two ones. If the girth g of the corresponding bipartite graph equals $g = 4t + 2$, then

$$m \geq \frac{\bar{k}^2}{t+1} \cdot n^{\frac{t}{t+1}},$$

and if $g = 4t$ for $t \geq 2$, then

$$m \geq \frac{\bar{k}^{(2t-1)}}{t(t-1)} \cdot n^{\frac{t-1}{t}}.$$

Here, $\bar{k} = (d_L - 1)^t$ if $g = 4t + 2$, and $\bar{k} = (d_L - 1)^{t-1}$ if $g = 4t$.

Theorem 12 : General lower bound on measurements using girth and average degrees
Suppose $A \in \{0,1\}^{m \times n}$ with $m < n$, and that in the bipartite graph associated with A, every node has degree at least two. Let E denote the total number of edges, and define the average left and right degrees as

$$\bar{d}_L = \frac{E}{n}, \quad \bar{d}_R = \frac{E}{m}.$$

If the graph has girth $g = 2r$, then

$$m \geq \prod_{i=0}^{r-1} \left((\bar{d}_L - 1)^{1/2} (\bar{d}_R - 1)^{1/2} \right) = \left((\bar{d}_L - 1)(\bar{d}_R - 1) \right)^{r/2}.$$

This bound holds for any graph with node degrees 2 and provides a lower bound on the number of measurements m, given n, $\bar{d}_L$, and g.

Theorem 13 : Girth exactly six for structured binary matrices
Let $A \in \{0,1\}^{\ell q \times q^2}$ be a binary matrix for integers $4 \leq \ell \leq q - 1$. Suppose:

1. The average left degree $\bar{d}_L \geq \ell$.
2. The maximum inner product between any two columns of A is one.
3. Every row and every column of A contains at least two ones.

Then the girth g of the bipartite graph corresponding to A is exactly six.

4 Implementation and Code

We evaluated the reconstruction on three different types of measurement matrices:

1. **Array Code**: are structured binary matrices inspired by coding theory, especially *Low-Density Parity-Check (LDPC)* codes. They have binary entries with regualr pattern, are sparse and have low coherence, which aid in sparse recovery. They are often reconstructed using cyclic permutations and block matrices like:

$$H(q,\ell) = \begin{bmatrix} I & I & I & \cdots & I \\ I & P & P^2 & \cdots & P^{q-1} \\ I & P^2 & P^4 & \cdots & P^{2(q-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I & P^{\ell-1} & P^{2(\ell-1)} & \cdots & P^{(\ell-1)(q-1)} \end{bmatrix}$$

   for P being a permutation matrix.
   These factors, make them hardware friendly in terms of implementation. They also offer **near-optimal girth**, which avoids short cycles in the associated bipartite graph.

2. **Guassian Matrices** are random matrices whose entries are independent and identically distributed, and are drawn from a Guassian Distribution:

$$A_{ij} \sim \mathcal{N}\left(0, \frac{1}{m}\right)$$

   They satisfy the Restricted Isometry Property (RIP) with high probability, ensuring accurate recovery and are universally good across many signal types and sparsity bases. However, they are very dense and not hardware-friendly or storage-efficient.

3. **Euler Matrices** are deterministic binary matrices constructed using Euler squares or combinatorial designs. They're based on placing symbols in a square array so that each symbol appears exactly once per row and column (like Latin squares, but with multiple orthogonal arrays). Euler matrices are used to construct binary matrices with Low mutual coherence and Structured sparsity. They've a guaranteed number of overlaps between column pairs.

   They offer deterministic construction, Low complexity and fast encoding. And also are close to satisfying RIP or mutual incoherence property.

The evaluation was based on three major factors:

- **Timing Analysis**: This metric records the time taken to reconstruct the sparse signal using different sensing matrices. Faster reconstruction is desirable for real-time applications and highlights computational efficiency.

- **Phase Transition:** In the context of compressive sensing, phase transition refers to the boundary in the (δ, ρ)-plane where successful recovery transitions to failure. Here, $\delta = \frac{m}{n}$ is the measurement rate, and $\rho = \frac{k}{m}$ is the sparsity ratio, where k is the number of non-zero elements in the signal, m is the number of measurements, and n is the signal dimension. The phase transition curve defines a threshold: below the curve, recovery is typically successful, while above it, recovery fails. This behavior is visualized to compare the theoretical and empirical performance of sensing matrices.

- **Structural Similarity Index (SSIM):** SSIM is a perceptual metric used to evaluate the visual similarity between the original and reconstructed signals (or images). It considers luminance, contrast, and structural information. The SSIM between two signals x and y is defined as:

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

  where:

- μ_x, μ_y are the means of x and y,
- σ_x^2, σ_y^2 are the variances,
- σ_{xy} is the covariance,
- $C_1 = (K_1 L)^2$, $C_2 = (K_2 L)^2$ are constants to stabilize the division (typically $K_1 = 0.01$, $K_2 = 0.03$, and L is the dynamic range of the image, e.g., 255 for 8-bit images).

4.1 Time Analysis

4.1.1 Array Code and Guassian Matrix

```
% Parameters
q = 89;
n = q * q;                  % Signal length
eps = 0;                    % Recovery tolerance (ideal case, no noise)
k_vals = [20];          % Sparsity levels

fprintf("===== Comparison of Array Code vs Gaussian Matrices =====\n");

for idx = 1:length(k_vals)
    k = k_vals(idx);
    d_L = k + 1;
    m = d_L * q;

    fprintf("\n--- For k = %d ---\n", k);

    % Generate k-sparse signal
    x = zeros(n, 1);
    support = randperm(n, k);
    x(support) = randn(k, 1);

    %% ==== ARRAY LDPC MATRIX ====
    H = generate_array_ldpc_matrix(q, d_L);
    A_bin = H(1:m, 1:n);
    y_bin = A_bin * x;

    tic;
    cvx_begin quiet
        variable x_bin_rec(n)
        minimize(norm(x_bin_rec, 1))
        subject to
            norm(A_bin * x_bin_rec - y_bin, 2) <= eps
    cvx_end
    time_bin = toc;

    %% ==== GAUSSIAN RANDOM MATRIX ====
    A_gauss = randn(m, n) / sqrt(m);
    y_gauss = A_gauss * x;

    tic;
    cvx_begin quiet
        variable x_gauss_rec(n)
        minimize(norm(x_gauss_rec, 1))
        subject to
            norm(A_gauss * x_gauss_rec - y_gauss, 2) <= eps
    cvx_end
    time_gauss = toc;

    %% ==== Report ====
    fprintf("Array Code Matrix Recovery Time:   %.4f seconds\n", time_bin);
    fprintf("Gaussian Matrix Recovery Time:     %.4f seconds\n", time_gauss);

    %% ==== Visualization ====
    figure('Name', sprintf('Signal Recovery for k = %d', k), 'NumberTitle', 'off');

```

```
    % Original signal
    subplot(3, 1, 1);
    stem(x, 'Marker', 'none');
    title(sprintf('Original Signal (k = %d)', k));
    xlabel('Index');
    ylabel('Amplitude');
    grid on;

    % Reconstructed from Array Code
    subplot(3, 1, 2);
    stem(x_bin_rec, 'r', 'Marker', 'none');
    title('Reconstructed Signal - Array Code');
    xlabel('Index');
    ylabel('Amplitude');
    grid on;

    % Reconstructed from Gaussian
    subplot(3, 1, 3);
    stem(x_gauss_rec, 'g', 'Marker', 'none');
    title('Reconstructed Signal - Gaussian');
    xlabel('Index');
    ylabel('Amplitude');
    grid on;
end

%% --- Structured Array LDPC Matrix Generator ---
function H = generate_array_ldpc_matrix(q, l)
    if ~isprime(q)
        error('q must be a prime number.');
    end
    if l >= q
        error('l must be less than q.');
    end

    P = zeros(q);
    for i = 1:q
        P(i, mod(i, q) + 1) = 1;
    end

    H = zeros(l * q, q^2);
    for i = 1:l
        for j = 1:q
            exponent = mod((i - 1) * (j - 1), q);
            block = P^exponent;
            row_range = (i - 1)*q + 1 : i*q;
            col_range = (j - 1)*q + 1 : j*q;
            H(row_range, col_range) = block;
        end
    end
end
```

Listing 1: My MATLAB Code

4.1.2 Euler Matrix

```
% Parameters
q = 89;                          % Prime number for Euler matrix
n = q * q;                       % Signal length = q^2
eps = 0;                         % Recovery tolerance (noiseless case)
k_vals = [20];                % Sparsity levels

fprintf("===== Euler Square Matrix Recovery Performance =====\n");

for idx = 1:length(k_vals)
    k = k_vals(idx);
    d_L = k + 1;                 % Column weight
    m = d_L * q;                 % Number of measurements

    fprintf("\n--- For k = %d ---\n", k);

    % Generate k-sparse signal
    x = zeros(n, 1);
    support = randperm(n, k);
    x(support) = randn(k, 1);

    %% ==== EULER SQUARE MATRIX ====
    A_euler = generate_euler_matrix(q, d_L);
    y_euler = A_euler * x;

    tic;
    cvx_begin quiet
        variable x_euler_rec(n)
        minimize(norm(x_euler_rec, 1))
        subject to
            norm(A_euler * x_euler_rec - y_euler, 2) <= eps
    cvx_end
    time_euler = toc;

    %% ==== SSIM Evaluation ====
    dim = sqrt(n);
    x_2D = reshape(x, dim, []);
    x_euler_2D = reshape(x_euler_rec, dim, []);
    ssim_euler = ssim(x_euler_2D, x_2D);

    %% ==== Reporting ====
    fprintf("Euler Matrix Recovery Time:    %.4f seconds\n", time_euler);
    fprintf("Euler Matrix SSIM:             %.4f\n", ssim_euler);

    %% ==== Visualization ====
    figure('Name', sprintf('Euler Recovery (k = %d)', k), 'NumberTitle', 'off');

    subplot(2, 1, 1);
    stem(x, 'Marker', 'none');
    title(sprintf('Original Signal (k = %d)', k));
    xlabel('Index');
    ylabel('Amplitude');
    grid on;

    subplot(2, 1, 2);
    stem(x_euler_rec, 'g', 'Marker', 'none');
    title(sprintf('Reconstructed - Euler (SSIM = %.4f)', ssim_euler));
    xlabel('Index');
    ylabel('Amplitude');
    grid on;
end

%% --- Euler Square Matrix Generator ---
function A = generate_euler_matrix(q, l)
    if l >= q
        error('l must be less than q');
    end
```

```
67
68      A = zeros(l*q, q^2);
69
70      for i = 0:q-1
71          for j = 0:q-1
72              col = i * q + j + 1;
73              for k = 0:l-1
74                  row = mod(i * k + j, q) + k * q + 1;
75                  A(row, col) = 1;
76              end
77          end
78      end
79  end
```

Listing 2: My MATLAB Code

4.2 Phase Transition

```
1  n = 961;                    % Signal dimension (e.g., 31*31)
2  q = 31;                     % Prime such that n = q^2
3  eps = 1e-6;                 % Recovery tolerance
4  trials = 10;                 % Trials per (m, k)
5  success_map = zeros(20, 20); % Preallocate success grid
6
7  fprintf("=== Starting Phase Transition Simulation ===\n");
8
9  m_vals = linspace(100, 900, 9);  % Example measurement range
10 for i = 1:length(m_vals)
11     m = round(m_vals(i));
12     for k = 5:(m/20):m
13         d_L = ceil(m / q);
14         if d_L >= q
15             fprintf('Skipping m=%d, k=%d: d_L=%d     q\n', m, k, d_L);
16             continue;
17         end
18
19         success_count = 0;
20
21         for t = 1:trials
22             % Generate k-sparse signal
23             x = zeros(n, 1);
24             support = randperm(n, k);
25             x(support) = randn(k, 1);
26
27             % Generate LDPC sensing matrix
28             % H = generate_array_ldpc_matrix(q, d_L);
29             %if size(H,1) < m
30             %    continue;
31             %end
32             %A = H(1:m, 1:n);
33             A = randn(m, n) / sqrt(m);
34             y = A * x;
35
36             %   1   minimization
37             try
38                 cvx_begin quiet
39                     variable x_rec(n)
40                     minimize(norm(x_rec, 1))
41                     subject to
42                         norm(A * x_rec - y, 2) <= eps
43                 cvx_end
44
45                 if norm(x - x_rec) / norm(x) < 1e-2
46                     success_count = success_count + 1;
47                 end
48             catch
```

```
                continue;  % Skip on solver failure
            end
        end

        success_rate = success_count / trials;
        success_map(j, i) = success_rate;
        fprintf("m = %d, k = %d, success = %.2f\n", m, k, success_rate);
    end
end

[k_grid, m_grid] = meshgrid(k_vals, m_vals);
x_axis = m_grid / n;        % m/n
y_axis = k_grid ./ m_grid;  % k/m

figure;
imagesc(x_axis(1,:), y_axis(:,1), success_map');  % Transpose map!
axis xy;
xlabel('m / n (measurement rate)');
ylabel('k / m (sparsity density)');
title('Phase Transition Diagram (LDPC Matrix)');
colorbar;
colormap(jet);


function H = generate_array_ldpc_matrix(q, l)
    if ~isprime(q)
        error('q must be prime.');
    end
    if l >= q
        error('l must be less than q.');
    end
    P = zeros(q);
    for i = 1:q
        P(i, mod(i, q) + 1) = 1;
    end
    H = zeros(l * q, q^2);
    for i = 1:l
        for j = 1:q
            exponent = mod((i - 1)*(j - 1), q);
            block = P^exponent;
            H((i-1)*q+1:i*q, (j-1)*q+1:j*q) = block;
        end
    end
end
```

Listing 3: My MATLAB Code

4.3 Structural Similarity Index

```
% Fixed parameters
q = 19;                     % Prime number
n = q * q;                  % Signal dimension = 961
d_L = 5;                    % Left degree (fixed)
l = d_L;
m = d_L * q;                % Number of measurements = 62
eps = 0;                    % Recovery tolerance
num_trials = 5;            % Number of trials per k

% Range of sparsity levels
k_values = 1:m;
avg_ssim_scores = zeros(size(k_values));

% Generate array code matrix once
H = generate_array_ldpc_matrix(q, l);
%H = generate_euler_matrix(q, l);
%A = H(1:m, 1:n);
```

```
18
19 % Loop over different sparsity levels
20 for i = 1:length(k_values)
21     k = k_values(i);
22     ssim_trials = zeros(num_trials, 1);
23
24     for trial = 1:num_trials
25         % Generate random k-sparse signal
26         x = zeros(n, 1);
27         support = randperm(n, k);
28         x(support) = randn(k, 1);
29
30         % Compressed measurements
31         A = randn(m, n) / sqrt(m);
32         y = A * x;
33
34         % Basis Pursuit via CVX
35         cvx_begin quiet
36             variable x_rec(n)
37             minimize(norm(x_rec, 1))
38             subject to
39                 norm(A * x_rec - y, 2) <= eps
40         cvx_end
41
42         % Compute SSIM
43         dim = sqrt(n);
44         x_mat = reshape(x, dim, []);
45         x_rec_mat = reshape(x_rec, dim, []);
46         ssim_trials(trial) = ssim(x_rec_mat, x_mat);
47     end
48
49     % Average SSIM for this sparsity level
50     avg_ssim_scores(i) = mean(ssim_trials);
51 end
52
53 % Plot Average SSIM vs. Sparsity Level
54 figure;
55 plot(k_values, avg_ssim_scores, '-o', 'LineWidth', 2);
56 xlabel('Sparsity Level (k)');
57 ylabel('Average SSIM over 10 trials');
58 title(sprintf('Avg SSIM vs. k (Array Code, n = %d, m = %d, d_L = %d)', n, m, d_L));
59 grid on;
60
61 function H = generate_array_ldpc_matrix(q, l)
62     if ~isprime(q)
63         error('q must be a prime number.');
64     end
65     if l >= q
66         error('l must be less than q.');
67     end
68
69     P = zeros(q);
70     for i = 1:q
71         P(i, mod(i, q) + 1) = 1;
72     end
73
74     H = zeros(l * q, q^2);
75     for i = 1:l
76         for j = 1:q
77             exponent = mod((i - 1) * (j - 1), q);
78             block = P^exponent;
79             row_range = (i - 1)*q + 1 : i*q;
80             col_range = (j - 1)*q + 1 : j*q;
81             H(row_range, col_range) = block;
82         end
83     end
84 end
85
```

```
function A = generate_euler_matrix(q, l)
    % Generate a binary matrix A of size l*q x q^2 with column weight l
    % such that the max inner product between any two columns is <= 1
    % Works for ANY integer q >= 2

    if l >= q
        error('l must be less than q');
    end

    A = zeros(l*q, q^2);  % Preallocate binary matrix

    for i = 0:q-1
        for j = 0:q-1
            col = i * q + j + 1;  % Column index in [1, q^2]
            for k = 0:l-1
                % Compute row index: k * q + (i * k + j) mod q
                row = mod(i * k + j, q) + k * q + 1;
                A(row, col) = 1;
            end
        end
    end
end
```

Listing 4: My MATLAB Code

5 Experimental Results and Discussion

5.1 Timing Analysis

Below Table is a summary of Recovery Time and SSIM index for various type of matrices.

Table 2: Comparison of Recovery Time and SSIM for Different Matrix Types

Matrix Size (n)	k	**Matrix Type**	**Recovery Time (s)**	**SSIM**
31×31	8	Euler	0.3325	1.000
		Array Code	0.3837	1.000
		Gaussian	1.5895	1.000
	20	Euler	0.3325	1.000
		Array Code	0.5003	1.000
		Gaussian	4.6924	1.000
47×47	8	Euler	0.6579	1.000
		Array Code	0.7671	1.000
		Gaussian	9.4308	1.000
	20	Euler	1.5294	1.000
		Array Code	1.9075	1.000
		Gaussian	47.3575	1.000
67×67	8	Euler	1.8511	—
		Array Code	1.2478	1.000
		Gaussian	48.7888	1.000
	20	Euler	4.1303	—
		Array Code	3.3832	1.000
		Gaussian	224.2477	1.000
89×89	8	Euler	2.3728	—
		Array Code	1.9319	—
		Gaussian	187.3826	—

5.2 Phase Transition

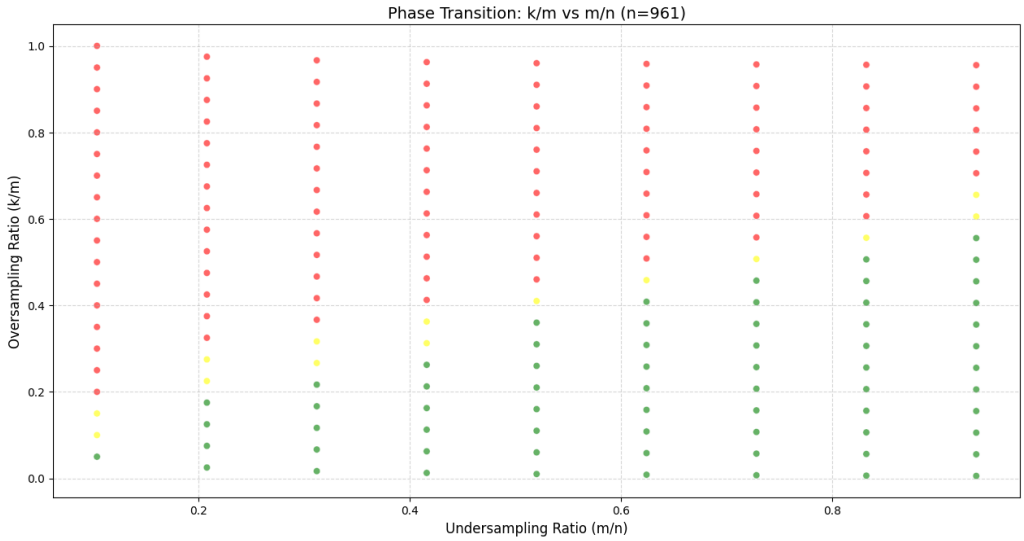

Figure 1: Phase Transition Analysis for Array Code

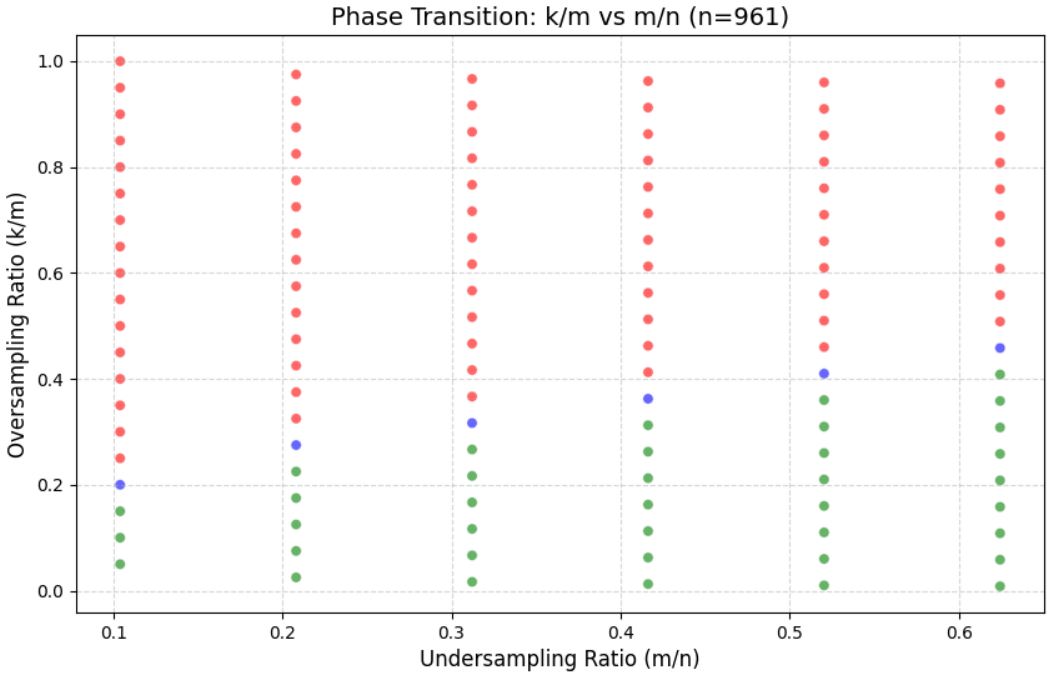

Figure 2: Phase Transition Analysis for Guassian Matrices

5.3 Structural Similarity Index

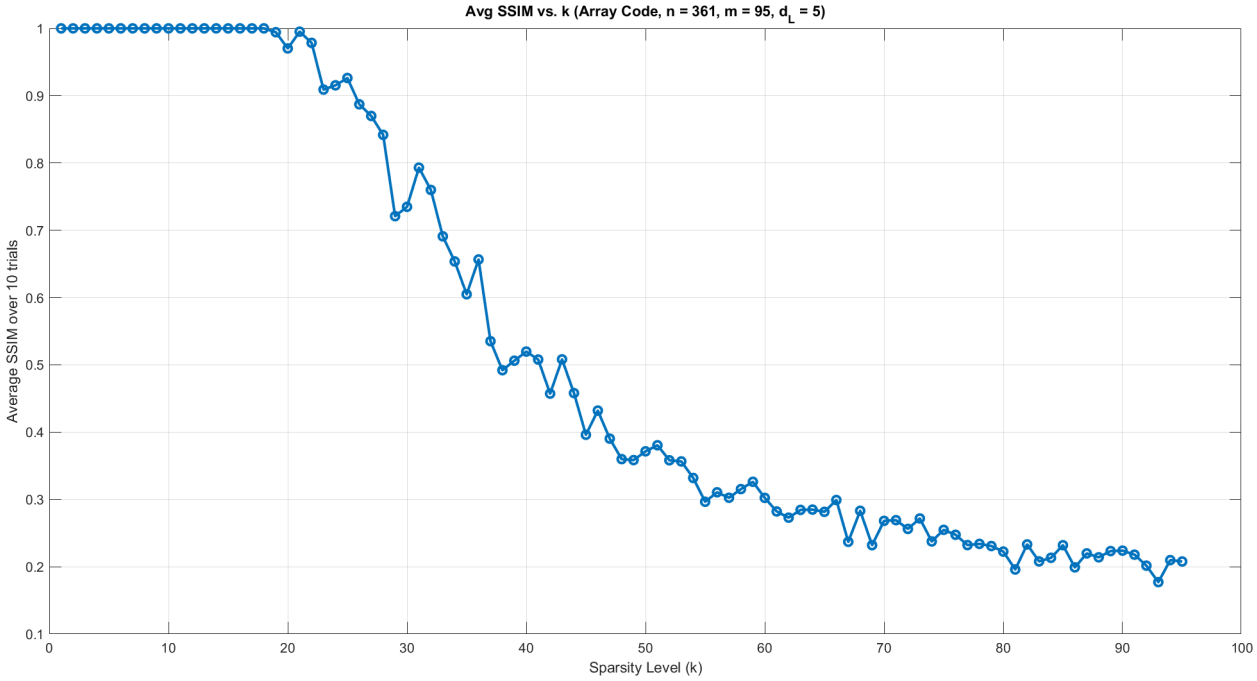

Figure 3: Structural Similarity Index for Array Code

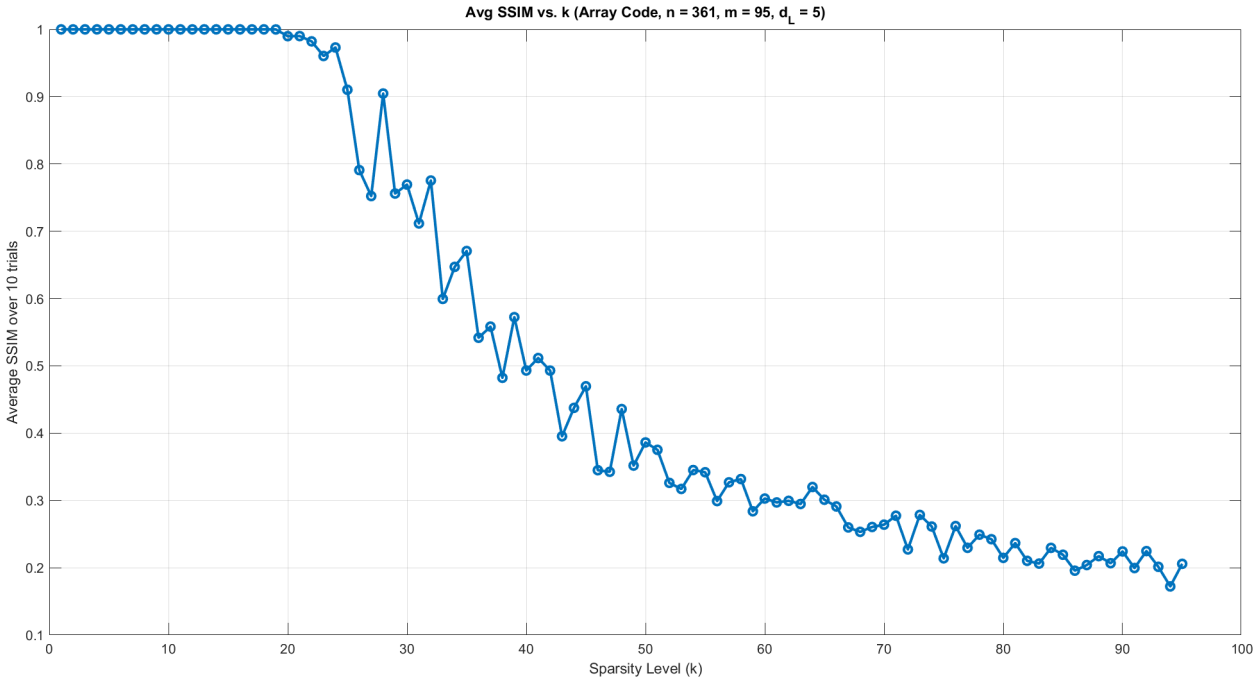

Figure 4: Structural Similarity Index for Euler Matrix

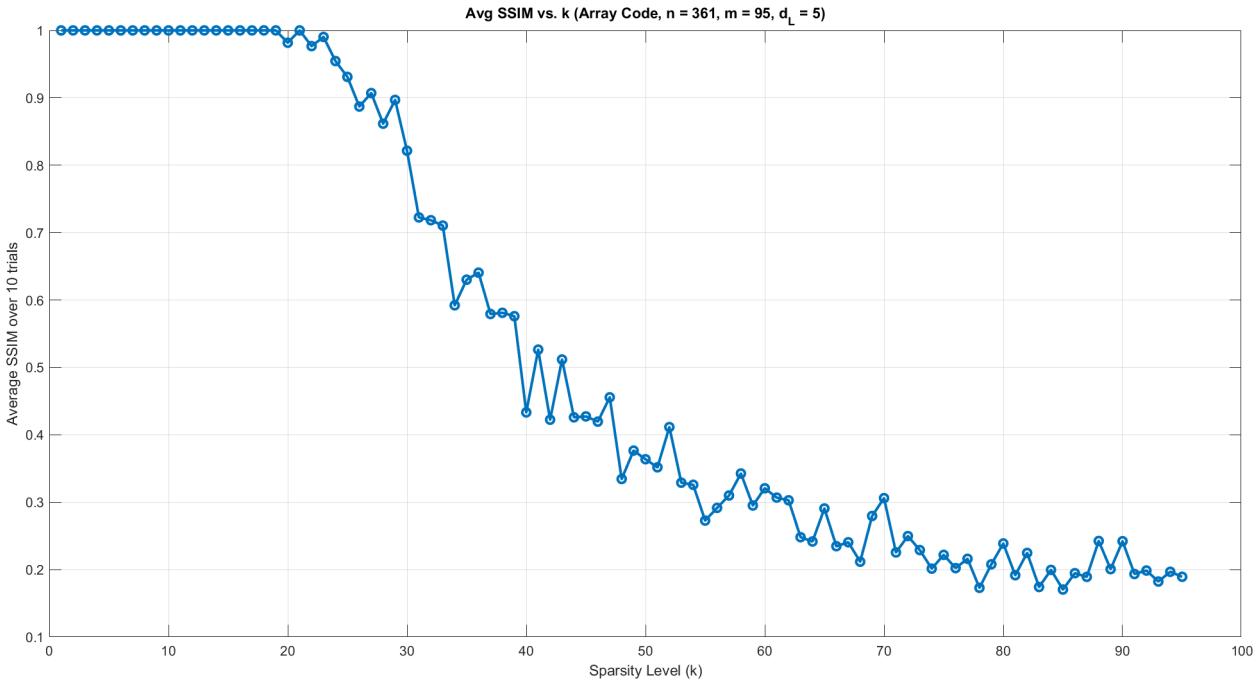

Figure 5: Structural Similarity Index for Guassian Matrices

6 Conclusion

Robust Null Space Property(RNSP) is a weaker condition than the **Restricted Isometric Property**(RIP), i.e. if RIP holds then RNSP will definitely hold, but the viva-versa may or may not hold. By virtue of this paper, we're able to exploit the reconstruction for Binary matrices efficiently compared to the standard baselines.

7 References

[1] M. Lotfi and M. Vidyasagar, "Compressed sensing using binary matrices of nearly optimal dimensions," IEEE Transactions on Signal Processing, vol. 68, pp. 3008–3022, 2020. doi: 10.1109/TSP.2020.2994260