

RUHR-UNIVERSITÄT BOCHUM

On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes

Eurocrypt 2015, Sofia

Alexander May, Ilya Ozerov

Chair for Cryptology and IT Security

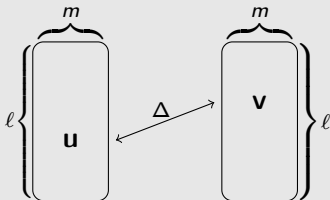
Horst Görtz Institute, Ruhr University Bochum

1. Algorithm for the *Nearest Neighbor Problem*
2. Application to Decoding of Random Binary Linear Codes
 - Stern (1989)
 - BJMM (2012)

Nearest Neighbor Problem

In our work we discuss the (m, ℓ, Δ) *Nearest Neighbor Problem* in \mathbb{F}_2 :

Given two lists $\mathbf{L}, \mathbf{R} \subset \mathbb{F}_2^m$ of ℓ uniform and pairwise independent vectors, find all $(\mathbf{u}, \mathbf{v}) \in \mathbf{L} \times \mathbf{R}$ with a Hamming distance of Δ .

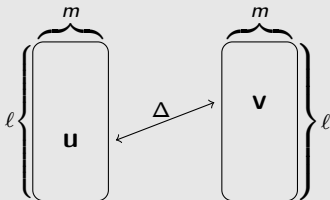


Interesting case: $\ell = 2^{\lambda m}$

Nearest Neighbor Problem

In our work we discuss the (m, ℓ, Δ) *Nearest Neighbor Problem* in \mathbb{F}_2 :

Given two lists $\mathbf{L}, \mathbf{R} \subset \mathbb{F}_2^m$ of ℓ uniform and pairwise independent vectors, find all $(\mathbf{u}, \mathbf{v}) \in \mathbf{L} \times \mathbf{R}$ with a Hamming distance of Δ .



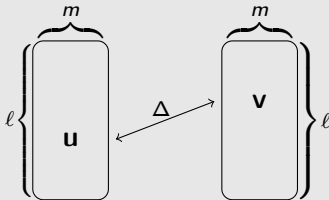
Naive approach: $\tilde{O}(\ell^2)$

Interesting case: $\ell = 2^{\lambda m}$

Nearest Neighbor Problem

In our work we discuss the (m, ℓ, Δ) *Nearest Neighbor Problem* in \mathbb{F}_2 :

Given two lists $\mathbf{L}, \mathbf{R} \subset \mathbb{F}_2^m$ of ℓ uniform and pairwise independent vectors, find all $(\mathbf{u}, \mathbf{v}) \in \mathbf{L} \times \mathbf{R}$ with a Hamming distance of Δ .



Naive approach: $\tilde{O}(\ell^2)$

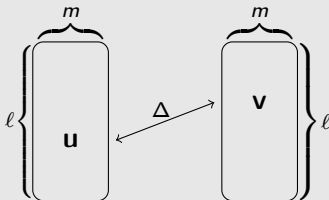
- $\tilde{\Theta}(\ell^2)$, if $\Delta = \frac{m}{2}$

Interesting case: $\ell = 2^{\lambda m}$

Nearest Neighbor Problem

In our work we discuss the (m, ℓ, Δ) *Nearest Neighbor Problem* in \mathbb{F}_2 :

Given two lists $\mathbf{L}, \mathbf{R} \subset \mathbb{F}_2^m$ of ℓ uniform and pairwise independent vectors, find all $(\mathbf{u}, \mathbf{v}) \in \mathbf{L} \times \mathbf{R}$ with a Hamming distance of Δ .



Naive approach: $\tilde{O}(\ell^2)$

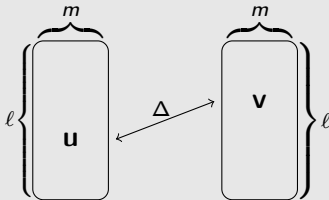
- $\tilde{\Theta}(\ell^2)$, if $\Delta = \frac{m}{2}$
- but $\tilde{O}(\ell)$, if $\Delta = 0$

Interesting case: $\ell = 2^{\lambda m}$

Nearest Neighbor Problem

In our work we discuss the (m, ℓ, Δ) *Nearest Neighbor Problem* in \mathbb{F}_2 :

Given two lists $\mathbf{L}, \mathbf{R} \subset \mathbb{F}_2^m$ of ℓ uniform and pairwise independent vectors, find all $(\mathbf{u}, \mathbf{v}) \in \mathbf{L} \times \mathbf{R}$ with a Hamming distance of Δ .



Interesting case: $\ell = 2^{\lambda m}$

Naive approach: $\tilde{O}(\ell^2)$

- $\tilde{\Theta}(\ell^2)$, if $\Delta = \frac{m}{2}$
- but $\tilde{O}(\ell)$, if $\Delta = 0$
- subquadratic, if $0 < \Delta < \frac{m}{2}$?

Previous Results

There are mainly two previous results solving the problem:

- Valiant (2012): $\tilde{O}(\ell^{1.8})$ with fast matrix multiplication:
 - exponent stays at 1.8 even for small values of Δ .

Previous Results

There are mainly two previous results solving the problem:

- Valiant (2012): $\tilde{O}(\ell^{1.8})$ with fast matrix multiplication:
 - exponent stays at 1.8 even for small values of Δ .
- Dubiner (2010): $\tilde{O}(\ell^{\frac{1}{1-\frac{\Delta}{m}}})$ with “bucketing codes”:
 - pros:
 - ▶ best known result for small Δ .
 - ▶ interpolates between the special cases $\Delta = 0$ and $\Delta = \frac{m}{2}$.

Previous Results

There are mainly two previous results solving the problem:

- Valiant (2012): $\tilde{O}(\ell^{1.8})$ with fast matrix multiplication:
 - exponent stays at 1.8 even for small values of Δ .
- Dubiner (2010): $\tilde{O}(\ell^{\frac{1}{1-\frac{\Delta}{m}}})$ with “bucketing codes”:
 - pros:
 - ▶ best known result for small Δ .
 - ▶ interpolates between the special cases $\Delta = 0$ and $\Delta = \frac{m}{2}$.
 - cons:
 - ▶ only holds for ℓ that are sub-exponential in m .
 - ▶ so far no result for the case of $\ell = 2^{\lambda m}$.

Our Result

We present an *Algorithmic Tool* that

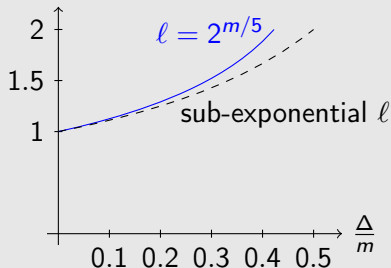
- gives a conceptually easy algorithm,
- is build for list sizes ℓ that are exponential in m ,
- implies a better decoding algorithm for linear codes.

Our Result

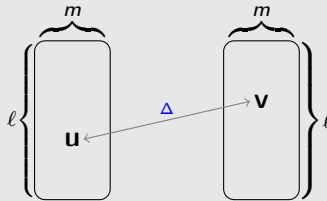
We present an *Algorithmic Tool* that

- gives a conceptually easy algorithm,
- is build for list sizes ℓ that are exponential in m ,
- implies a better decoding algorithm for linear codes.

complexity exponent

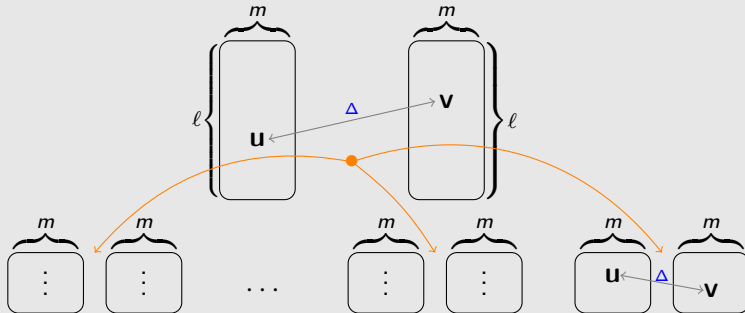


High Level Idea



- Algorithm creates exponentially many easy copies of the problem.
 - by reducing the number of elements, *individually* in **L** and **R**.
- It guarantees that “distance Δ pairs” are in one of these copies.

High Level Idea



- Algorithm creates exponentially many easy copies of the problem.
 - by reducing the number of elements, *individually* in \mathbf{L} and \mathbf{R} .
- It guarantees that “distance Δ pairs” are in one of these copies.

Nearest Neighbor Algorithm

Input: $\mathbf{L}, \mathbf{R}, m, \ell, \Delta$

Output: All $(\mathbf{u}, \mathbf{v}) \in \mathbf{L} \times \mathbf{R}$ with $\text{HammingDistance}(\mathbf{u}, \mathbf{v}) = \Delta$

1: choose optimized parameters r and w

2: **repeat** r times:

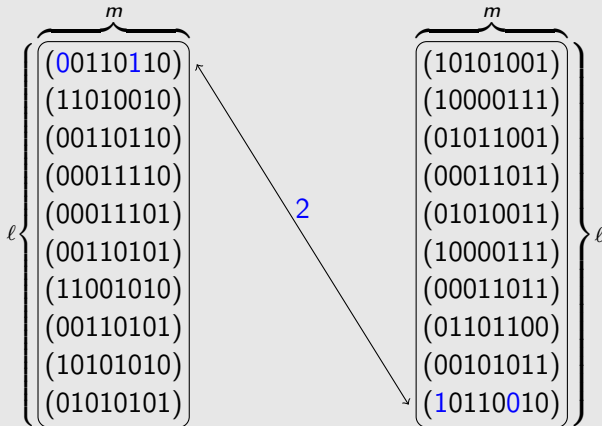
A: choose a uniformly random partition of the columns $\{1, \dots, m\}$

B: consider all elements of \mathbf{L}, \mathbf{R} with weight w on these columns

C: brute-force all these pairs and output those with distance Δ

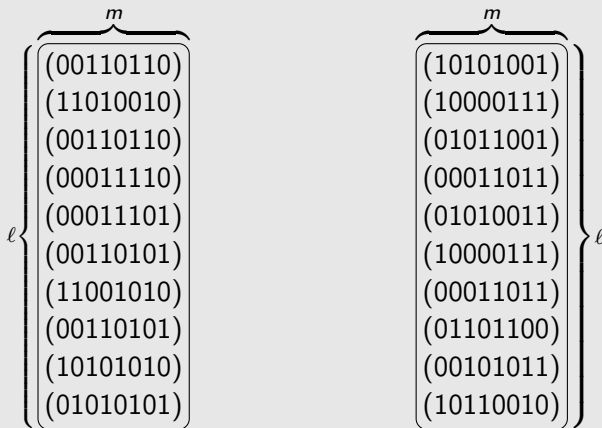
Example: $\Delta = 2$

Problem: find all pairs with distance 2.



Example: $\Delta = 2$

A: choose a uniformly random partition of the columns $\{1, \dots, m\}$



Example: $\Delta = 2$

B: consider all elements \mathbf{L}, \mathbf{R} with weight $w = 1$ on these columns

m			m			
ℓ	(11	10)	(10	01)
	(01	10)	(00	11)
	(11	10)	(01	01)
	(01	10)	(01	11)
	(01	01)	(01	11)
	(11	01)	(00	11)
	(00	10)	(01	11)
	(11	01)	(10	00)
	(10	10)	(10	11)
	(01	01)	(11	10)

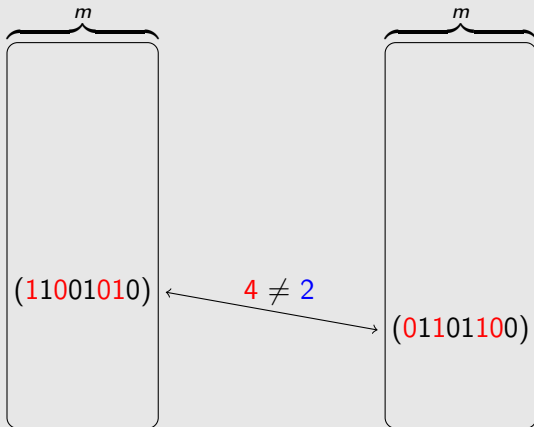
Example: $\Delta = 2$

C: brute-force all these pairs and output those with distance $\Delta = 2$

ℓ		m			
		(11	10)	[3]
		(01	10)	[2]
		(11	10)	[3]
		(01	10)	[2]
		(01	01)	[2]
		(11	01)	[3]
		(00	10)	$\langle 1 \rangle$
		(11	01)	[3]
		(10	10)	[2]
		(01	01)	[2]
		m			
ℓ		(10	01)	[2]
		(00	11)	[2]
		(01	01)	[2]
		(01	11)	[3]
		(01	11)	[3]
		(00	11)	[2]
		(01	11)	[3]
		(10	00)	$\langle 1 \rangle$
		(10	11)	[3]
		(10	11)	[3]
		(11	10)	[3]

Example: $\Delta = 2$

In most steps we won't find a pair with distance 2. Repeat!



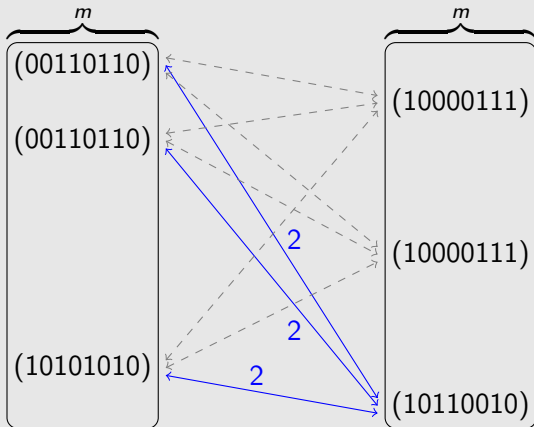
Example: $\Delta = 2$

What is a 'good' partition? If *both* vectors with $\Delta = 2$ have weight w .

$\left\{ \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \right.$		m			
		(0 10 0)	$\langle 1 \rangle$		
		(1 10 0)	[2]		
		(0 10 0)	$\langle 1 \rangle$		
		(0 11 0)	[2]		
		(0 11 1)	[3]		
		(0 10 1)	[2]		
		(1 01 0)	[2]		
		(0 10 1)	[2]		
		(0 01 0)	$\langle 1 \rangle$		
		(1 10 1)	[3]		
$\left\{ \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \right.$		m			
		[2] (0 01 1)			
		$\langle 1 \rangle$ (0 00 1)			
		[4] (1 11 1)			
		[3] (0 11 1)			
		[3] (1 10 1)			
		$\langle 1 \rangle$ (0 00 1)			
		[3] (0 11 1)			
		[2] (1 01 0)			
		[2] (0 01 1)			
		$\langle 1 \rangle$ (0 10 0)			

Example: $\Delta = 2$

In this case pairs with distance 2 'survive'.



Nearest Neighbor Algorithm

Input: $\mathbf{L}, \mathbf{R}, m, \ell, \Delta$

Output: All $(\mathbf{u}, \mathbf{v}) \in \mathbf{L} \times \mathbf{R}$ with $\text{HammingDistance}(\mathbf{u}, \mathbf{v}) = \Delta$

1: choose optimized parameters r and w

2: **repeat** r times:

A: choose a uniformly random partition of the columns $\{1, \dots, m\}$

B: consider all elements of \mathbf{L}, \mathbf{R} with weight w on these columns

C: brute-force all these pairs and output those with distance Δ

Choice of weight w and repetitions r

- w controls how many vectors 'survive' the filtering
- brute-force compares each 'survivor' of \mathbf{L} with each of \mathbf{R}

$\implies w(m, \ell)$ is chosen such that expected $\Theta(1)$ vectors 'survive'

Choice of weight w and repetitions r

- w controls how many vectors 'survive' the filtering
- brute-force compares each 'survivor' of \mathbf{L} with each of \mathbf{R}

$\implies w(m, \ell)$ is chosen such that expected $\Theta(1)$ vectors 'survive'

- \mathbf{u}, \mathbf{v} with $\text{dist}(\mathbf{u}, \mathbf{v}) = \Delta$ have to 'survive' in at least one repetition

\implies choice of $r(m, \Delta, w)$ guarantees it with overwhelming probability

Time Complexity

- Steps A (sampling) and C (brute-force) are easy.
- But how do we compute the list of 'weight w vectors' in step B ?
 - Traversing the whole input list would be worse than brute-force!
- Instead: tree based algorithm, decreasing list sizes on each level.
 - This makes our algorithm run in time $\tilde{O}(r)$.
- Unfortunately, it introduces a polynomial overhead of m^t .
 - **Open Problem:** Is it possible to get rid of this polynomial?

Application: Syndrome Decoding

Let \mathbf{H} be a parity check matrix of a random binary linear $[n, k, d]$ code.

Decoding Problem: find error $\mathbf{e} \in \mathbb{F}_2^n$, $\text{weight}(\mathbf{e}) = \frac{d}{2}$ s.t. $\mathbf{H} \cdot \mathbf{e} = \mathbf{s}$.

Application: Syndrome Decoding

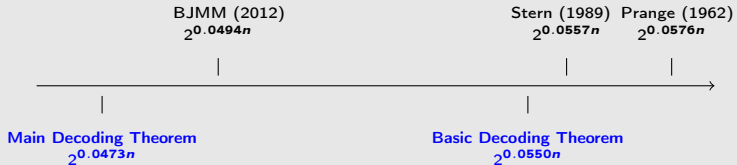
Let \mathbf{H} be a parity check matrix of a random binary linear $[n, k, d]$ code.

Decoding Problem: find error $\mathbf{e} \in \mathbb{F}_2^n$, $\text{weight}(\mathbf{e}) = \frac{d}{2}$ s.t. $\mathbf{H} \cdot \mathbf{e} = \mathbf{s}$.

Decoding Theorem

Our decoding algorithm solves the Decoding Problem in $2^{0.0473n}$.

Overview



Previous Results

Decoding Problem: find error $\mathbf{e} \in \mathbb{F}_2^n$, $\text{weight}(\mathbf{e}) = \frac{d}{2}$ s.t. $\mathbf{H} \cdot \mathbf{e} = \mathbf{s}$.

- Prange (1962) uses a brute-force approach:
 - guess a small part of \mathbf{e} that contains all ones
 - then use linear algebra to efficiently compute that part

Previous Results

Decoding Problem: find error $\mathbf{e} \in \mathbb{F}_2^n$, $\text{weight}(\mathbf{e}) = \frac{d}{2}$ s.t. $\mathbf{H} \cdot \mathbf{e} = \mathbf{s}$.

- Prange (1962) uses a brute-force approach:
 - guess a small part of \mathbf{e} that contains all ones
 - then use linear algebra to efficiently compute that part
- Stern (1989) generalizes to a meet-in-the-middle approach:
 - search for *exact* collisions in \mathbf{e}
 - use linear algebra to check if the weight matches

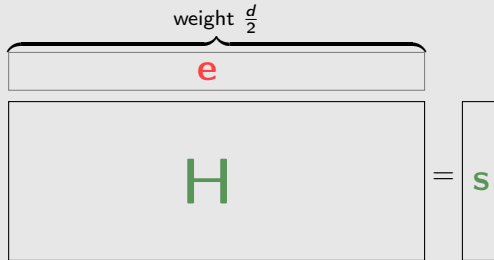
Previous Results

Decoding Problem: find error $\mathbf{e} \in \mathbb{F}_2^n$, $\text{weight}(\mathbf{e}) = \frac{d}{2}$ s.t. $\mathbf{H} \cdot \mathbf{e} = \mathbf{s}$.

- Prange (1962) uses a brute-force approach:
 - guess a small part of \mathbf{e} that contains all ones
 - then use linear algebra to efficiently compute that part
- Stern (1989) generalizes to a meet-in-the-middle approach:
 - search for *exact* collisions in \mathbf{e}
 - use linear algebra to check if the weight matches

Our approach: search directly for *approximate* collisions in \mathbf{e}

Our Decoding Algorithm



The diagram illustrates the decoding equation $H \cdot e = s$. It features a large rectangular box labeled H in green. Above this box is a smaller, narrower rectangular box labeled e in red. A horizontal curly brace is positioned above the e box, with the text "weight $\frac{d}{2}$ " centered above it. To the right of the H box is an equals sign, followed by a tall, narrow vertical rectangular box labeled s in green.

1. Randomly permute the columns of H and obtain a matrix $(P_1|P_2|Q)$.

Our Decoding Algorithm

$$\begin{array}{|c|c|c|} \hline \text{weight } \frac{d}{2} \\ \hline \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ \hline \end{array}
 \begin{array}{|c|c|c|} \hline \mathbf{P}_1 & \mathbf{P}_2 & \mathbf{Q} \\ \hline \end{array}
 =
 \begin{array}{|c|} \hline \mathbf{s} \\ \hline \end{array}$$

1. Randomly permute the columns of \mathbf{H} and obtain a matrix $(\mathbf{P}_1|\mathbf{P}_2|\mathbf{Q})$.
2. Multiply both sides by \mathbf{Q}^{-1} , define $\mathbf{A}_i := \mathbf{Q}^{-1} \cdot \mathbf{P}_i$ and $\mathbf{t} := \mathbf{Q}^{-1} \cdot \mathbf{s}$.

Our Decoding Algorithm

$$\begin{array}{|c|c|c|} \hline \text{weight } \frac{d}{2} \\ \hline \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ \hline \end{array}
 \begin{array}{|c|c|c|} \hline \mathbf{A}_1 & \mathbf{A}_2 & \mathbf{I} \\ \hline \end{array}
 = \begin{array}{|c|} \hline \mathbf{t} \\ \hline \end{array}$$

1. Randomly permute the columns of \mathbf{H} and obtain a matrix $(\mathbf{P}_1 | \mathbf{P}_2 | \mathbf{Q})$.
2. Multiply both sides by \mathbf{Q}^{-1} , define $\mathbf{A}_i := \mathbf{Q}^{-1} \cdot \mathbf{P}_i$ and $\mathbf{t} := \mathbf{Q}^{-1} \cdot \mathbf{s}$.
 - Hope that the error vector splits in weights $\frac{p}{2}$, $\frac{p}{2}$, and $\frac{d}{2} - p$.

Our Decoding Algorithm

$$\begin{array}{|c|c|c|} \hline \text{weight } \frac{p}{2} & \text{weight } \frac{p}{2} & \text{weight } \frac{d}{2} - p \\ \hline \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ \hline \mathbf{A}_1 & \mathbf{A}_2 & \mathbf{I} \\ \hline \end{array} = \mathbf{t}$$

1. Randomly permute the columns of \mathbf{H} and obtain a matrix $(\mathbf{P}_1 | \mathbf{P}_2 | \mathbf{Q})$.
2. Multiply both sides by \mathbf{Q}^{-1} , define $\mathbf{A}_i := \mathbf{Q}^{-1} \cdot \mathbf{P}_i$ and $\mathbf{t} := \mathbf{Q}^{-1} \cdot \mathbf{s}$.
 - Hope that the error vector splits in weights $\frac{p}{2}$, $\frac{p}{2}$, and $\frac{d}{2} - p$.
 - Move the $\mathbf{A}_2 \cdot \mathbf{e}_2$ part on the other side of the equation.

Our Decoding Algorithm

$$\begin{array}{c} \text{weight } \frac{p}{2} \\ \boxed{\mathbf{e}_1} \\ \boxed{\mathbf{A}_1} \end{array} + \begin{array}{c} \text{weight } \frac{d}{2} - p \\ \boxed{\mathbf{e}_3} \\ \boxed{\mathbf{I}} \end{array} = \begin{array}{c} \boxed{\mathbf{t}} \end{array} - \begin{array}{c} \text{weight } \frac{p}{2} \\ \boxed{\mathbf{e}_2} \\ \boxed{\mathbf{A}_2} \end{array}$$

1. Randomly permute the columns of \mathbf{H} and obtain a matrix $(\mathbf{P}_1|\mathbf{P}_2|\mathbf{Q})$.
2. Multiply both sides by \mathbf{Q}^{-1} , define $\mathbf{A}_i := \mathbf{Q}^{-1} \cdot \mathbf{P}_i$ and $\mathbf{t} := \mathbf{Q}^{-1} \cdot \mathbf{s}$.
 - Hope that the error vector splits in weights $\frac{p}{2}$, $\frac{p}{2}$, and $\frac{d}{2} - p$.
 - Move the $\mathbf{A}_2 \cdot \mathbf{e}_2$ part on the other side of the equation.
 - Since \mathbf{e}_3 is multiplied by \mathbf{I} , both sides are approximately the same.

Our Decoding Algorithm

$$\begin{array}{c} \text{weight } \frac{p}{2} \\ \boxed{\mathbf{e}_1} \\ \boxed{\mathbf{A}_1} \end{array} \approx_{\frac{d}{2}-p} \begin{array}{c} \text{weight } \frac{p}{2} \\ \boxed{\mathbf{e}_2} \\ \boxed{\mathbf{t}} - \boxed{\mathbf{A}_2} \end{array}$$

1. Randomly permute the columns of \mathbf{H} and obtain a matrix $(\mathbf{P}_1|\mathbf{P}_2|\mathbf{Q})$.
2. Multiply both sides by \mathbf{Q}^{-1} , define $\mathbf{A}_i := \mathbf{Q}^{-1} \cdot \mathbf{P}_i$ and $\mathbf{t} := \mathbf{Q}^{-1} \cdot \mathbf{s}$.
 - Hope that the error vector splits in weights $\frac{p}{2}, \frac{p}{2}$, and $\frac{d}{2} - p$.
 - Move the $\mathbf{A}_2 \cdot \mathbf{e}_2$ part on the other side of the equation.
 - Since \mathbf{e}_3 is multiplied by \mathbf{I} , both sides are approximately the same.
3. Solve the Nearest Neighbor Problem with $\Delta = \frac{d}{2} - p$.

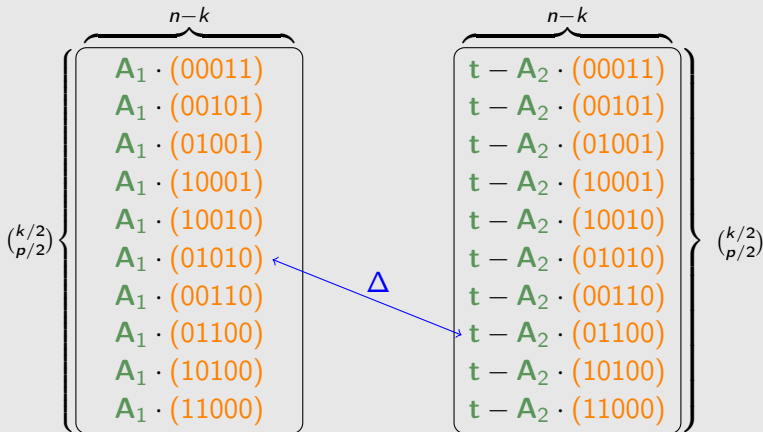
Nearest Neighbor Problem with $\Delta = \frac{d}{2} - p$

$$A_1 \cdot e_1 \approx t - A_2 \cdot e_2$$

	$n-k$		$n-k$	
$\left\{ \begin{matrix} (k/2) \\ (p/2) \end{matrix} \right\}$	$\left\{ \begin{matrix} A_1 \cdot (00011) \\ A_1 \cdot (00101) \\ A_1 \cdot (01001) \\ A_1 \cdot (10001) \\ A_1 \cdot (10010) \\ A_1 \cdot (01010) \\ A_1 \cdot (00110) \\ A_1 \cdot (01100) \\ A_1 \cdot (10100) \\ A_1 \cdot (11000) \end{matrix} \right\}$		$\left\{ \begin{matrix} t - A_2 \cdot (00011) \\ t - A_2 \cdot (00101) \\ t - A_2 \cdot (01001) \\ t - A_2 \cdot (10001) \\ t - A_2 \cdot (10010) \\ t - A_2 \cdot (01010) \\ t - A_2 \cdot (00110) \\ t - A_2 \cdot (01100) \\ t - A_2 \cdot (10100) \\ t - A_2 \cdot (11000) \end{matrix} \right\}$	$\left(\begin{matrix} k/2 \\ p/2 \end{matrix} \right)$

Nearest Neighbor Problem with $\Delta = \frac{d}{2} - p$

$$A_1 \cdot e_1 \approx t - A_2 \cdot e_2$$



- Algorithm for finding close vectors in large lists.
- Application: improved decoding exponent by 5%.
- Open Problems:
 - Is it possible to get rid of the polynomial overhead?
 - Are there more applications of the Nearest Neighbor algorithm?

Conclusion

- Algorithm for finding close vectors in large lists.
- Application: improved decoding exponent by 5%.
- Open Problems:
 - Is it possible to get rid of the polynomial overhead?
 - Are there more applications of the Nearest Neighbor algorithm?

Thanks for your attention!