Local correctability of expander codes

Brett Hemenway Rafail Ostrovsky Mary Wootters

IAS

April 14, 2014

The point(s) of this talk

- Locally decodable codes are codes which admit sublinear time decoding of small pieces of a message.
- Expander codes are a family of error correcting codes based on expander graphs.
- In this work, we show that (appropriately instantiated) expander codes *are* high-rate locally decodable codes.

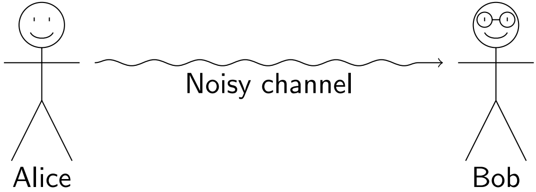
- Only two families of codes known in this regime [KSY'11,GKS'12].
- Expander codes (and the corresponding decoding algorithm and analysis) are very different from existing constructions!

Outline

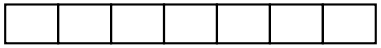
Outline

Outline

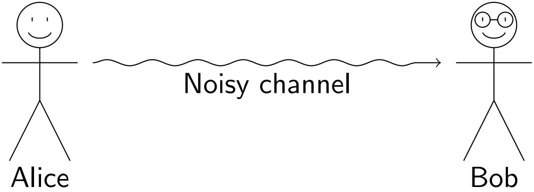
Error correcting codes



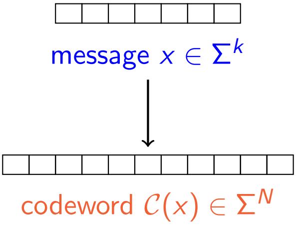
Error correcting codes

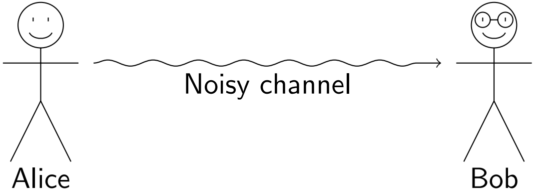


message $x \in \Sigma^k$

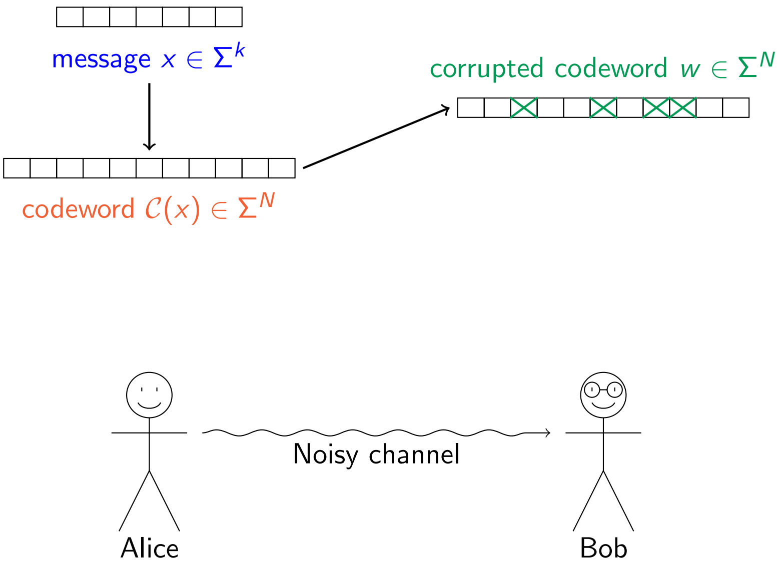


Error correcting codes

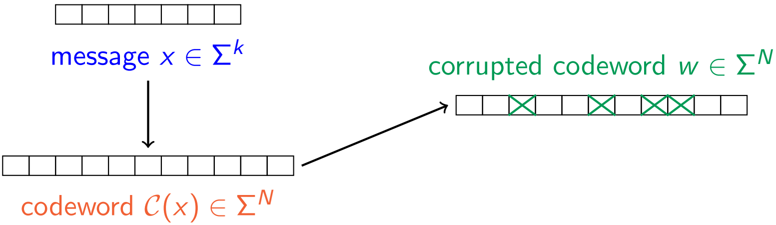


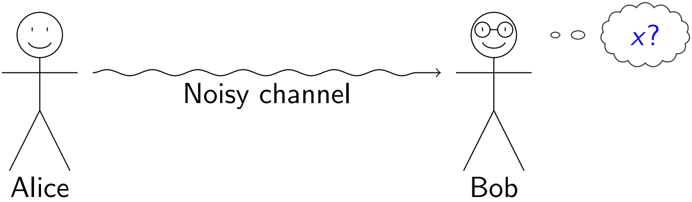


Error correcting codes

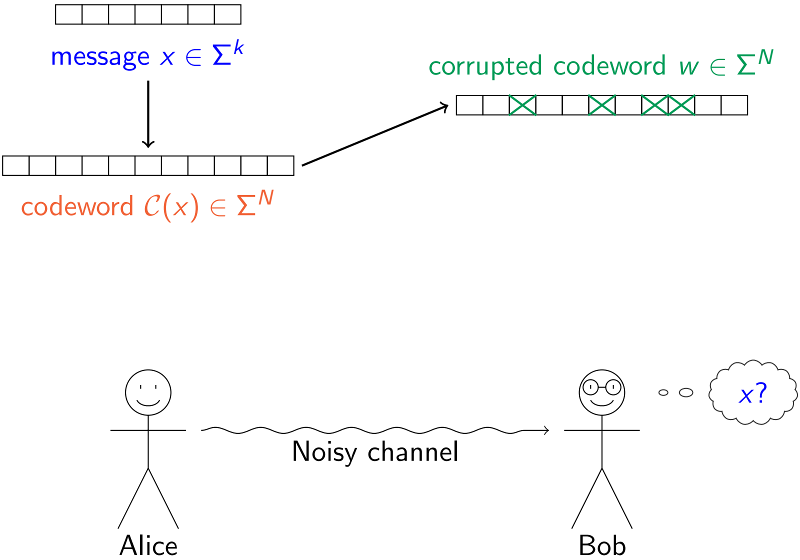


Error correcting codes

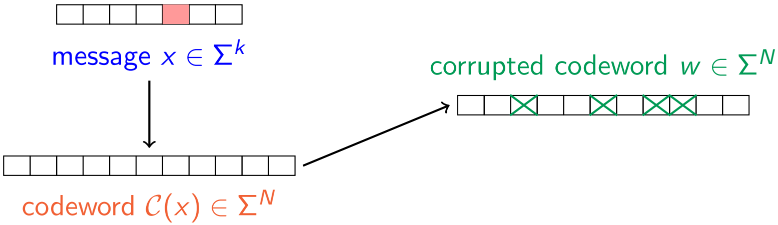


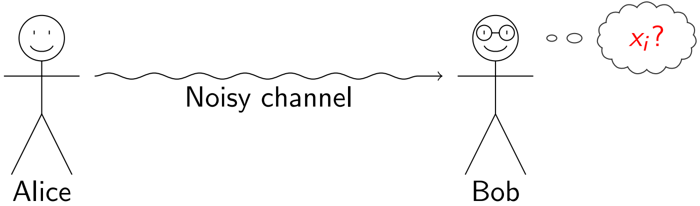


Locally decodable codes

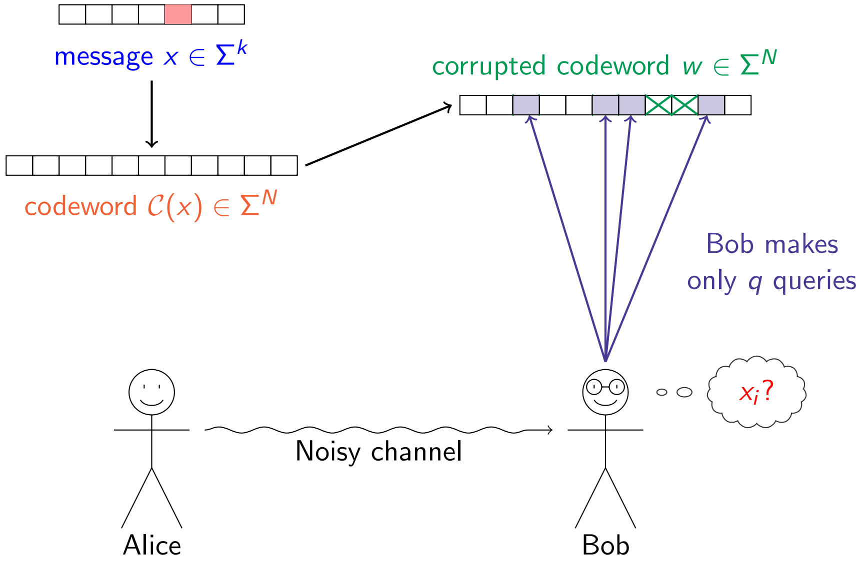


Locally decodable codes

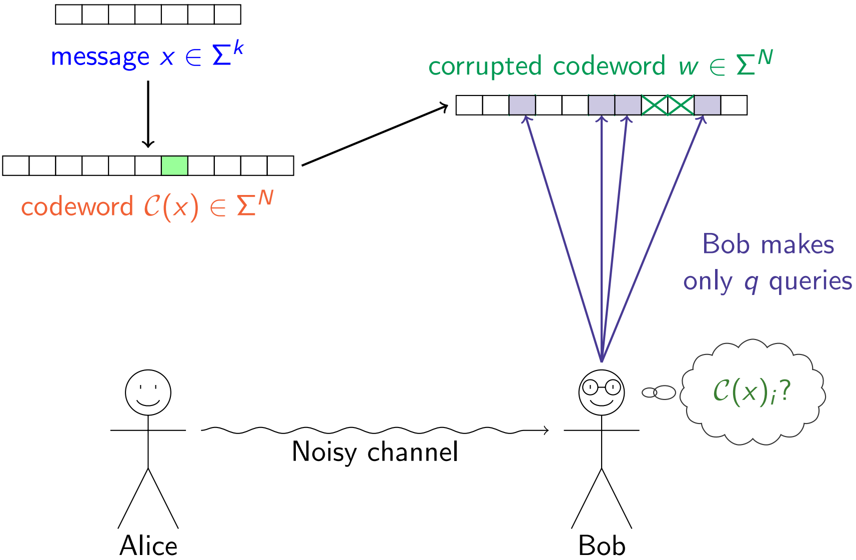




Locally decodable codes



Locally correctable codes



Locally correctable codes, sans stick figures

Definition

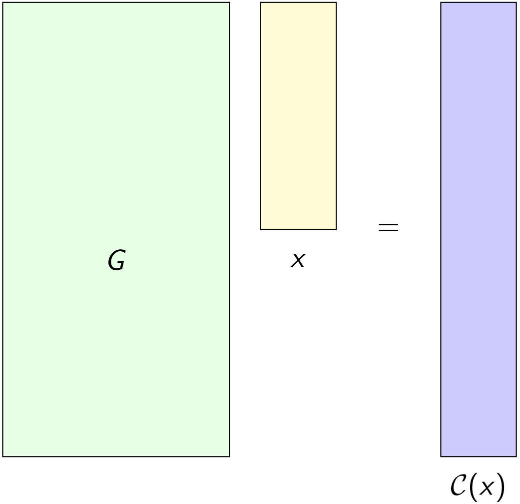
$\mathcal{C}$ is (q, δ, η)-**locally correctable** if for all $i \in [N]$, for all $x \in \Sigma^k$, and for all $w \in \Sigma^N$ with $d(w, \mathcal{C}(x)) \leq \delta N$,

$$\mathbb{P}\left\{\text{Bob correctly guesses } \mathcal{C}(x)_i\right\} \geq 1 - \eta.$$

Bob reads only q positions in the corrupted word, w.

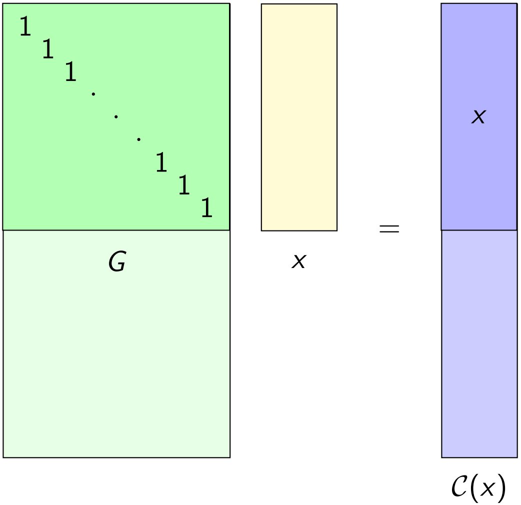
Local correctability vs. local decodability

When $\mathcal{C}$ is *linear*, local correctability implies local decodability.



Local correctability vs. local decodability

When $\mathcal{C}$ is *linear*, local correctability implies local decodability.



Before we get too far

Some notation

For a code $\mathcal{C} : \Sigma^k \to \Sigma^N$

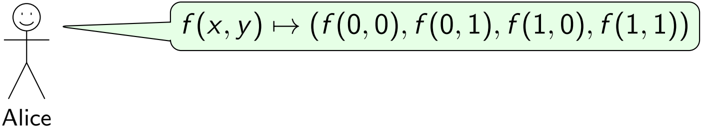
- The **message length** is k, the length of the message.
- The **block length** is N, the length of the codeword.
- The **rate** is k/N.
- The **locality** is q, the number of queries Bob makes.

Goal: large rate, small locality.

Outline

1. Local correctability
   - Definitions and notation
   - Example: Reed-Muller codes
   - Previous work and our contribution
2. Expander codes
3. Local correctability of expander codes
   - Requirement for the inner code: smooth reconstruction
   - Decoding algorithm
   - Example instantiation: finite geometry codes
4. Conclusion

Example: Reed-Muller Codes



- **Message:** multivariate polynomial of total degree d,

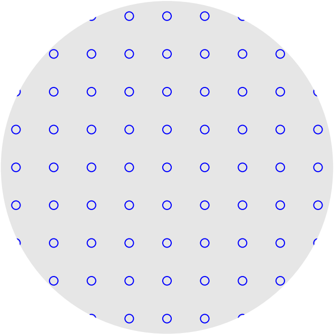
$$f \in \mathbb{F}_q[z_1, \ldots, z_m].$$

- **Codeword:** the evaluation of f at points in $\mathbb{F}_q^m$:

$$\mathcal{C}(f) = \{f(\vec{x})\}_{\vec{x} \in \mathbb{F}_q^m}$$

Locally Correcting Reed Muller Codes

Points in $\mathbb{F}_q^m$

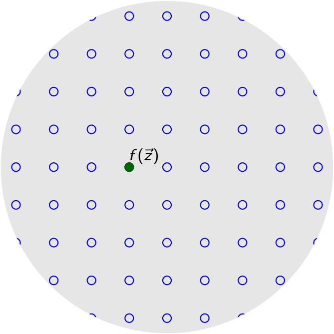


message is $f \in \mathbb{F}_q[z_1, \ldots, z_m]$

codeword is $\{f(\vec{x})\}_{\vec{x} \in \mathbb{F}_q^m}$

Locally Correcting Reed Muller Codes

Points in $\mathbb{F}_q^m$



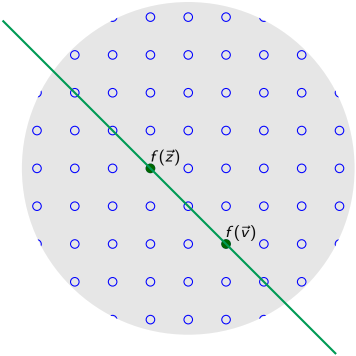
message is $f \in \mathbb{F}_q[z_1, \ldots, z_m]$

codeword is $\{f(\vec{x})\}_{\vec{x} \in \mathbb{F}_q^m}$

- We want to correct $\mathcal{C}(f)_{\vec{z}} = f(\vec{z})$.

Locally Correcting Reed Muller Codes

Points in $\mathbb{F}_q^m$



message is $f \in \mathbb{F}_q[z_1, \ldots, z_m]$

codeword is $\{f(\vec{x})\}_{\vec{x} \in \mathbb{F}_q^m}$

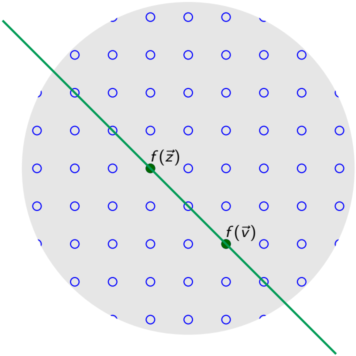
- We want to correct $\mathcal{C}(f)_{\vec{z}} = f(\vec{z})$.
- Choose a random line through $\vec{z}$, and consider the restriction

$$g(t) = f(\vec{z} + t\vec{v})$$

to that line.

Locally Correcting Reed Muller Codes

Points in $\mathbb{F}_q^m$



message is $f \in \mathbb{F}_q[z_1, \ldots, z_m]$

codeword is $\{f(\vec{x})\}_{\vec{x} \in \mathbb{F}_q^m}$

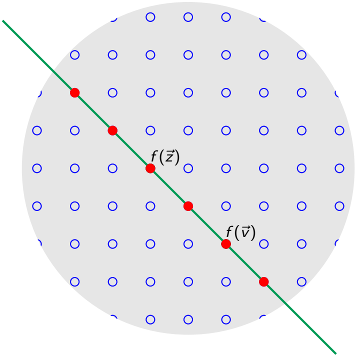
- We want to correct $\mathcal{C}(f)_{\vec{z}} = f(\vec{z})$.
- Choose a random line through $\vec{z}$, and consider the restriction

$$g(t) = f(\vec{z} + t\vec{v})$$

  to that line.
- This is a *univariate* polynomial, and $g(0) = f(\vec{z})$.

Locally Correcting Reed Muller Codes

Points in $\mathbb{F}_q^m$



message is $f \in \mathbb{F}_q[z_1, \ldots, z_m]$

codeword is $\{f(\vec{x})\}_{\vec{x} \in \mathbb{F}_q^m}$

- We want to correct $\mathcal{C}(f)_{\vec{z}} = f(\vec{z})$.
- Choose a random line through $\vec{z}$, and consider the restriction

$$g(t) = f(\vec{z} + t\vec{v})$$

  to that line.
- This is a *univariate* polynomial, and $g(0) = f(\vec{z})$.
- Query all of the points on the line.

Resulting parameters

- Rate is $\binom{m+d}{m}/q^m$ (we needed $d = O(q)$, so we can decode)
- Locality is q (the field size)

If we choose m constant, we get:

- Rate is constant, but less than $1/2$.
- Locality is $N^{1/m} = N^{\varepsilon}$.

Outline

1. Local correctability
   - Definitions and notation
   - Example: Reed-Muller codes
   - Previous work and our contribution
2. Expander codes
3. Local correctability of expander codes
   - Requirement for the inner code: smooth reconstruction
   - Decoding algorithm
   - Example instantiation: finite geometry codes
4. Conclusion

Question:

Reed-Muller Codes have locality N^{ε} and constant rate,
but rate is less than $1/2$.

Question:

Reed-Muller Codes have locality N^{ε} and constant rate, but rate is less than $1/2$.

Are there locally decodable codes with locality N^{ε}, and rate arbitrarily close to 1?

Previous Work

Rate $\to 1$ and locality N^{ε}:

- Multiplicity codes
  [Kopparty, Saraf, Yekhanin 2011]
- Lifted codes
  [Guo, Kopparty, Sudan 2012]

Previous Work

Rate $\to 1$ and locality N^{ε}:

- Multiplicity codes
  [Kopparty, Saraf, Yekhanin 2011]
- Lifted codes
  [Guo, Kopparty, Sudan 2012]

These have decoders similar to RM:
the queries form a good code.

Previous Work

Rate $\to 1$ and locality N^{ε}:

- Multiplicity codes [Kopparty, Saraf, Yekhanin 2011]
- Lifted codes [Guo, Kopparty, Sudan 2012]

These have decoders similar to RM: the queries form a good code.

Another regime:

Rate bad $\left(N / 2^{2^{O(\sqrt{\log(N)})}} \right)$, but locality 3:

- Matching vector codes [Yekhanin 2008, Efremenko 2009, ...]

These decoders are different:

- The queries cannot tolerate any errors.
- There are so few queries that they are probably all correct.

Previous Work

Rate $\to 1$ and locality N^{ε}:

- Multiplicity codes
  [Kopparty, Saraf, Yekhanin 2011]
- Lifted codes
  [Guo, Kopparty, Sudan 2012]

These have decoders similar to RM: the queries form a good code.

- Expander codes
  [H., Ostrovsky, Wootters 2013]

Decoder is similar in spirit to low-query decoders. The queries will *not* form an error correcting code.

Another regime:

Rate bad $\left(N / 2^{2^{O(\sqrt{\log(N)})}} \right)$, but locality 3:

- Matching vector codes
  [Yekhanin 2008, Efremenko 2009, ...]

These decoders are different:

- The queries cannot tolerate any errors.
- There are so few queries that they are probably all correct.

Outline

Tanner Codes [Tanner'81]

Given:

- A d-regular graph G with n vertices and $N = \frac{nd}{2}$ edges
- An *inner code* $\mathcal{C}_0$ with block length d over Σ

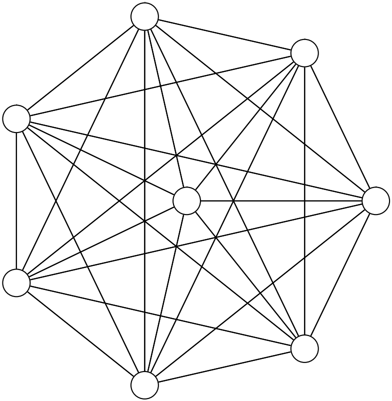
We get a *Tanner code* $\mathcal{C}$.

- $\mathcal{C}$ has block length N and alphabet Σ.
- Codewords are labelings of edges of G.
- A labeling is in $\mathcal{C}$ if the labels on each vertex form a codeword of $\mathcal{C}_0$.

Example [Tanner'81]

G is K_8, and $\mathcal{C}_0$ is the $[7, 4, 3]$-Hamming code.

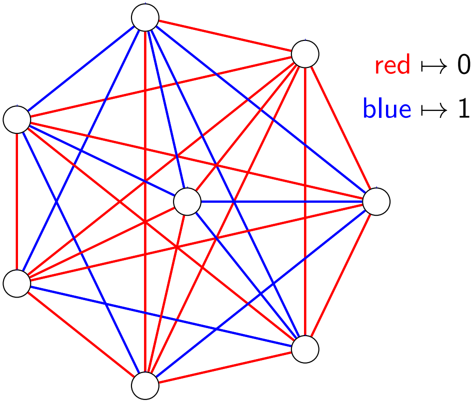
$$N = \binom{8}{2} = 28 \text{ and } \Sigma = \{0, 1\}$$



Example [Tanner'81]

G is K_8, and $\mathcal{C}_0$ is the $[7, 4, 3]$-Hamming code.

A codeword of $\mathcal{C}$ is a labeling of edges of G.

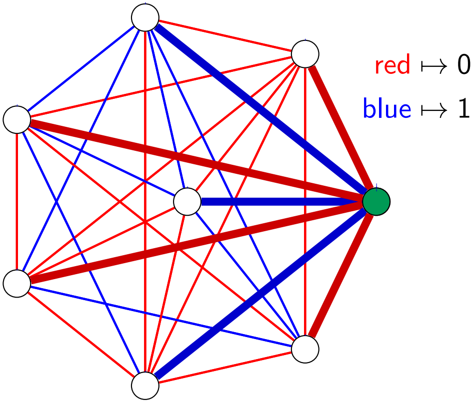


$$(0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1) \in \mathcal{C} \subset \{0, 1\}^{28}$$

Example [Tanner'81]

G is K_8, and $\mathcal{C}_0$ is the $[7,4,3]$-Hamming code.

These edges form a codeword in the Hamming code



$$(0,0,0,0,0,0,0,1,1,0,1,1,1,0,1,0,0,1,0,1,0,0,0,1,0,0,1,1) \in \mathcal{C} \subset \{0,1\}^{28}$$

Encoding Tanner Codes

Encoding is Easy!

1. Generate parity-check matrix
   Requires:
   - Edge-vertex incidence matrix of graph
   - Parity-check matrix of inner code
2. Calculate a basis for the kernel of the parity-check matrix
3. This basis defines a generator matrix for the linear Tanner Code
4. Encoding is just multiplication by this generator matrix

Linearity

If the inner code $\mathcal{C}_0$ is linear, so is the Tanner code $\mathcal{C}$

- $\mathcal{C}_0 = \mathrm{Ker}(H_0)$ for some *parity check* matrix H_0.

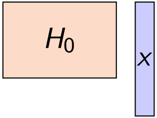
$$x \in \mathcal{C}_0 \iff H_0 \, x = 0$$

Linearity

If the inner code $\mathcal{C}_0$ is linear, so is the Tanner code $\mathcal{C}$

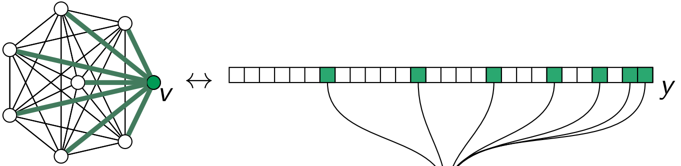
- $\mathcal{C}_0 = \mathrm{Ker}(H_0)$ for some *parity check* matrix H_0.

$x \in \mathcal{C}_0 \iff$

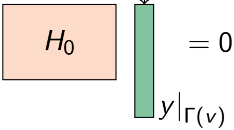


$= 0$

- So codewords of the Tanner code $\mathcal{C}$ also are defined by linear constraints:



$y \in \mathcal{C} \iff \forall v \in G,$



$= 0$

Example: vertex edge incidence matrix of K_8

1 row for each vertex

1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	0	0	0
0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	1	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	1	1

1 column for each edge

- Columns have weight 2
  (Each edge hits two vertices)
- Rows have weight 7
  (Each vertex has degree seven)

Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code

Parity-check
of Hamming code

$$\begin{matrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

$$\begin{matrix}
1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&0&0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&1&0&0&0&0&0&1&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0 \\
0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&1&1&1&0&0&0&0&0&0 \\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&0&1&1&1&0&0&0 \\
0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&1&0 \\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&0&1 \\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&1
\end{matrix}$$

Edge-vertex incidence matrix of K_8

Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code

$$\begin{matrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

Vertex 1

$$\begin{matrix}
1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&0&0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&1&0&0&0&0&0&1&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0 \\
0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&1&1&1&0&0&0&0&0&0 \\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&0&1&1&1&0&0&0 \\
0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&1&0 \\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&0&1 \\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&1
\end{matrix}$$

Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code

$$\begin{matrix}
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1
\end{matrix}$$

$$\begin{matrix}
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&0&0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&1&0&0&0&0&0&1&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0 \\
0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&1&1&1&0&0&0&0&0&0 \\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&0&1&1&1&0&0&0 \\
0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&1&0 \\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&0&1 \\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&1
\end{matrix}$$

Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code

$$\begin{array}{ccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}$$

$$\begin{array}{cccccccccccccccccccccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\color{blue}{1} & 0 & 0 & 0 & 0 & 0 & 0 & \color{blue}{1} & \color{blue}{1} & \color{blue}{1} & \color{blue}{1} & \color{blue}{1} & \color{blue}{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1
\end{array}$$

Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code

$$
\begin{matrix}
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1
\end{matrix}
$$

$$
\begin{matrix}
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&1&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&1&0&0&0&0&0&1&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0 \\
0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&1&1&1&0&0&0&0&0&0 \\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&0&1&1&1&0&0&0 \\
0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&1&0 \\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&0&1 \\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&1
\end{matrix}
$$

Example: parity-check matrix of a Tanner code

K_8 and the $[7, 4, 3]$-Hamming code

1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0

If the inner code has good rate, so does the outer code

Say that $\mathcal{C}_0$ is linear

- If $\mathcal{C}_0$ has rate r_0, it satisfies $(1 - r_0)d$ linear constraints.
- Each of the n vertices of G must satisfy these constraints.

If the inner code has good rate, so does the outer code

Say that $\mathcal{C}_0$ is linear

- If $\mathcal{C}_0$ has rate r_0, it satisfies $(1 - r_0)d$ linear constraints.
- Each of the n vertices of G must satisfy these constraints.

$$\Downarrow$$

- $\mathcal{C}$ is defined by at most $n \cdot (1 - r_0)d$ constraints.

If the inner code has good rate, so does the outer code

Say that $\mathcal{C}_0$ is linear

- If $\mathcal{C}_0$ has rate r_0, it satisfies $(1 - r_0)d$ linear constraints.
- Each of the n vertices of G must satisfy these constraints.

$$\Downarrow$$

- $\mathcal{C}$ is defined by at most $n \cdot (1 - r_0)d$ constraints.
- Length of $\mathcal{C} = N = \#$ edges $= nd/2$

If the inner code has good rate, so does the outer code

Say that $\mathcal{C}_0$ is linear

- If $\mathcal{C}_0$ has rate r_0, it satisfies $(1 - r_0)d$ linear constraints.
- Each of the n vertices of G must satisfy these constraints.

$$\Downarrow$$

- $\mathcal{C}$ is defined by at most $n \cdot (1 - r_0)d$ constraints.
- Length of $\mathcal{C} = N = \#$ edges $= nd/2$
- The rate of $\mathcal{C}$ is

$$R = \frac{k}{N} \geq \frac{N - n \cdot (1 - r_0)d}{N} = 2r_0 - 1.$$

Better rate bounds?

- The lower bound $R > 2r_0 - 1$ is *independent* of the ordering of edges around a vertex
- Tanner already noticed that order matters.
  Let G be the complete bipartite graph with 7 vertices per side
  Let $\mathcal{C}_0$ be the $[7, 4, 3]$ hamming code
  Then different "natural" orderings achieve a Tanner code with
  - $[49, 16, 9]$ ($\frac{16}{49} \approx .327$)
  - $[49, 12, 16]$ ($\frac{12}{49} \approx .245$)
  - $[49, 7, 17]$ ($\frac{7}{49} \approx .142$) Meets lower bound of $2 \cdot \frac{4}{7} - 1$

Expander codes

When the underlying graph is an *expander graph*, the Tanner code is a *expander code.*

- Expander codes admit very fast decoding algorithms [Sipser and Spielman 1996]
- Further improvements in [Sipser'96, Zemor'01, Barg and Zemor'02,'05,'06]

Outline

Outline

1. Local correctability
   - Definitions and notation
   - Example: Reed-Muller codes
   - Previous work and our contribution
2. Expander codes
3. Local correctability of expander codes
   - Requirement for the inner code: smooth reconstruction
   - Decoding algorithm
   - Example instantiation: finite geometry codes
4. Conclusion

Main Result

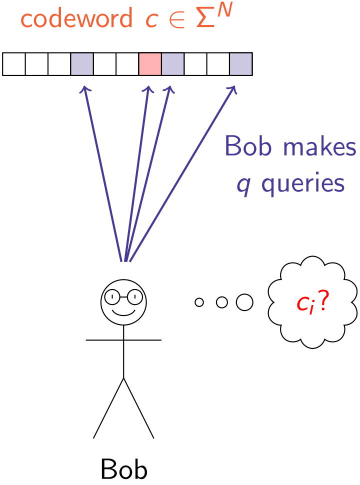
Given:

- a d-regular expander graph;
- an inner code of length d with **smooth reconstruction.**

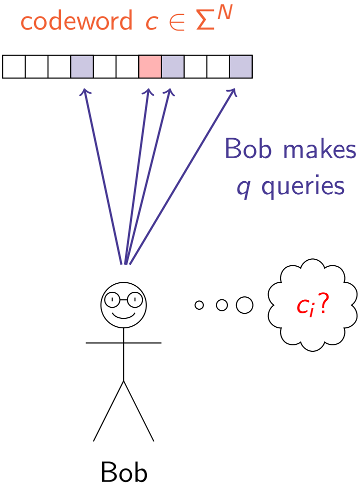
Then:

- We will give a local-correcting procedure for this expander code.

Smooth Reconstruction



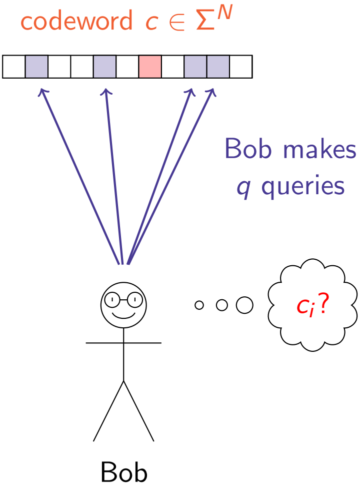
Smooth Reconstruction



Suppose that:

- Each Bob's q queries is (close to) uniformly distributed (they don't need to be independent!)

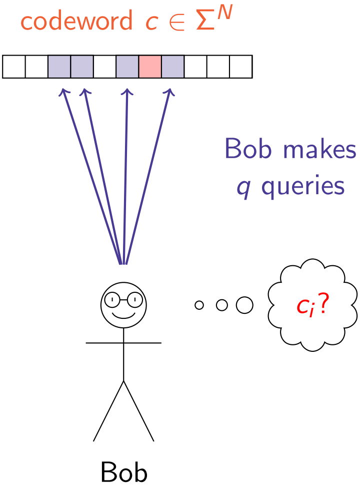
Smooth Reconstruction



Suppose that:

- Each Bob's q queries is (close to) uniformly distributed (they don't need to be independent!)

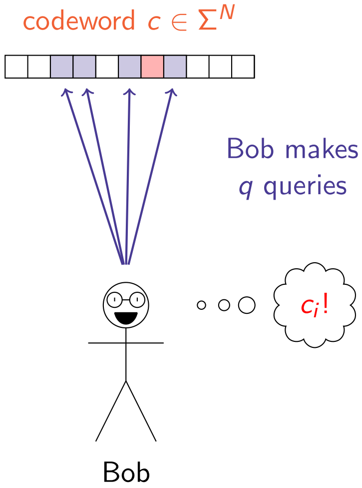
Smooth Reconstruction



Suppose that:

- Each Bob's q queries is (close to) uniformly distributed (they don't need to be independent!)

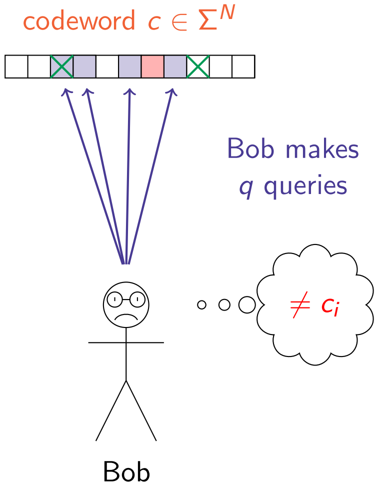
Smooth Reconstruction



Suppose that:

- Each Bob's q queries is (close to) uniformly distributed (they don't need to be independent!)
- From the (uncorrupted) queries, he can always recover c_i.

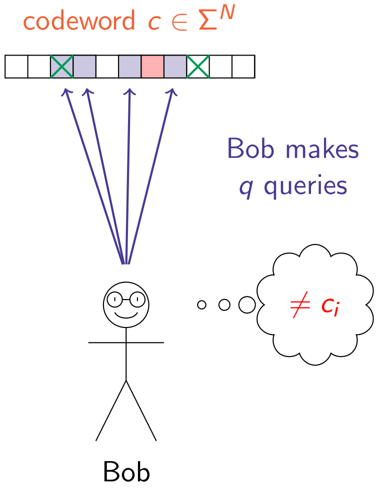
Smooth Reconstruction



Suppose that:

- Each Bob's q queries is (close to) uniformly distributed (they don't need to be independent!)
- From the (uncorrupted) queries, he can always recover c_i.
- But! He doesn't need to tolerate any errors.

Smooth Reconstruction



Suppose that:

- Each Bob's q queries is (close to) uniformly distributed (they don't need to be independent!)
- From the (uncorrupted) queries, he can always recover c_i.
- But! He doesn't need to tolerate any errors.

Then:

- We say that the code has a **smooth reconstruction algorithm.**

Smooth reconstruction, sans stick figures

Definition

A code $\mathcal{C}_0 \subset \Sigma^d$ has a q-query **smooth reconstruction algorithm** if, for all $i \in [d]$ and for all codewords $c \in \mathcal{C}_0$:

- Bob can always determine c_i from a set of queries $c_{i_1}, \ldots, c_{i_q}$
- Each c_{i_j} is (close to) uniformly distributed in $[d]$.

Outline

Main Result

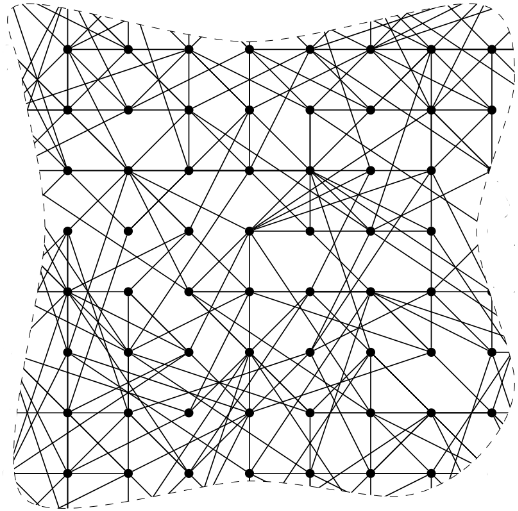
Given:

- a d-regular expander graph;
- an inner code of length d with smooth reconstruction.

Then:

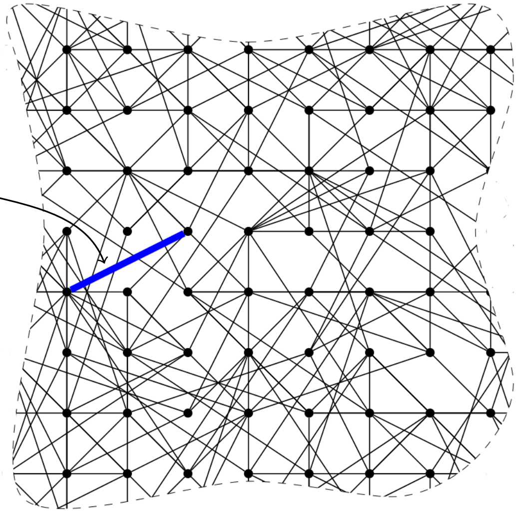
- We will give a local-correcting procedure for this expander code.

Decoding algorithm: main idea



Decoding algorithm: main idea

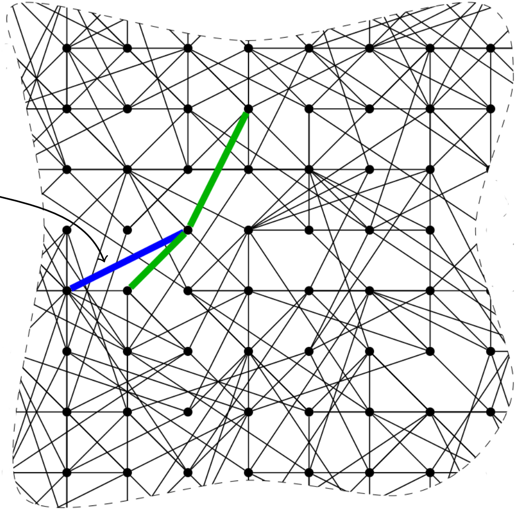
Want to
correct the label
on this edge



For this
diagram
$q = 2$

Decoding algorithm: main idea

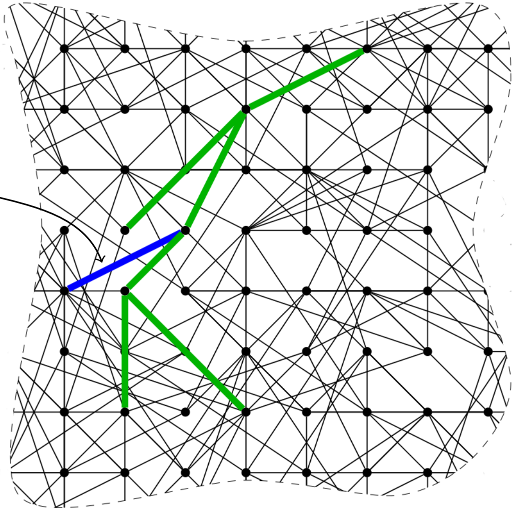
Want to correct the label on this edge



For this diagram $q = 2$

Decoding algorithm: main idea

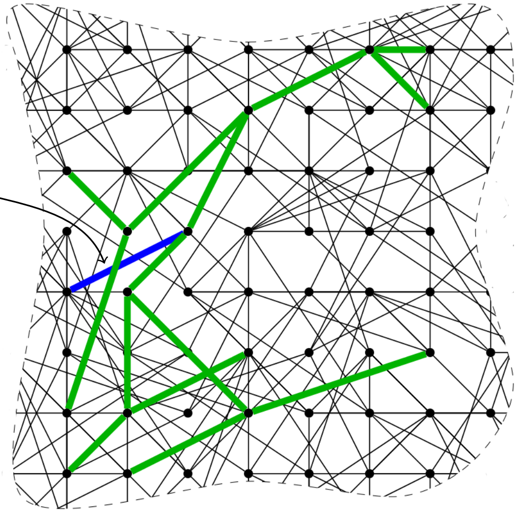
Want to correct the label on this edge



For this diagram $q = 2$

Decoding algorithm: main idea

Want to correct the label on this edge

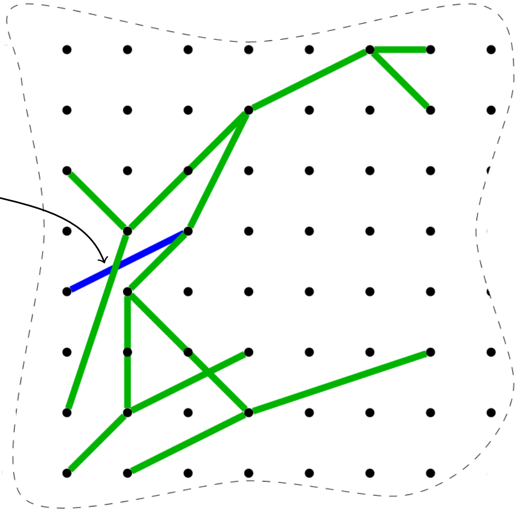


For this diagram $q = 2$

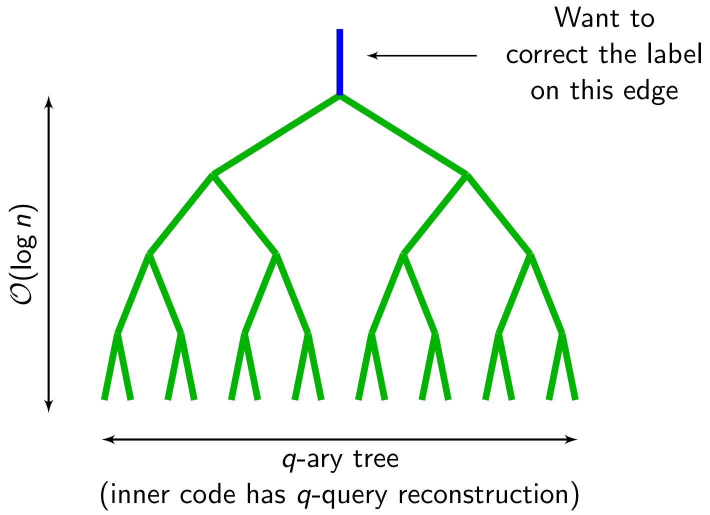
Decoding algorithm: main idea

Want to
correct the label
on this edge

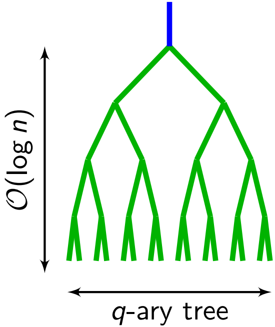
For this
diagram
$q = 2$



The expander walk as a tree



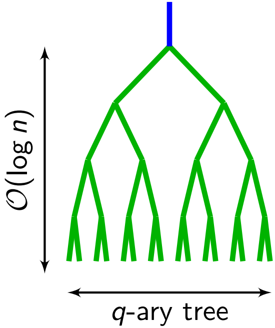
The expander walk as a tree



True Statements:

- The symbols on the leaves determine the symbol on the root.
- There are $q^{\mathcal{O}(\log(n))} \approx N^{\varepsilon}$ leaves.
- The leaves are (nearly) uniformly distributed in G.

The expander walk as a tree

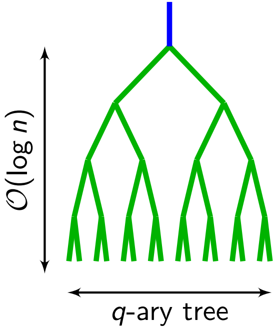


True Statements:

- The symbols on the leaves determine the symbol on the root.
- There are $q^{\mathcal{O}(\log(n))} \approx N^{\varepsilon}$ leaves.
- The leaves are (nearly) uniformly distributed in G.

Idea: Query the leaves!

The expander walk as a tree



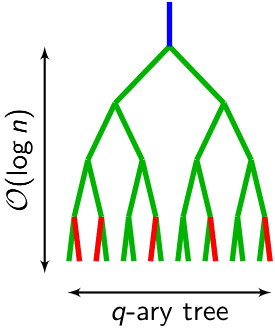
True Statements:

- The symbols on the leaves determine the symbol on the root.
- There are $q^{\mathcal{O}(\log(n))} \approx N^{\varepsilon}$ leaves.
- The leaves are (nearly) uniformly distributed in G.

Idea: Query the leaves!

Problems:

The expander walk as a tree



True Statements:

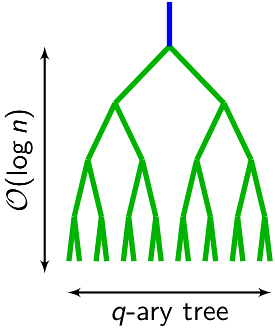
- The symbols on the leaves determine the symbol on the root.
- There are $q^{\mathcal{O}(\log(n))} \approx N^{\varepsilon}$ leaves.
- The leaves are (nearly) uniformly distributed in G.

Idea: Query the leaves!

Problems:

- There are errors on the leaves.

The expander walk as a tree



True Statements:

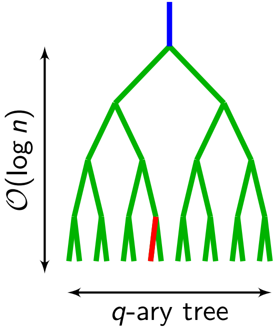
- The symbols on the leaves determine the symbol on the root.
- There are $q^{\mathcal{O}(\log(n))} \approx N^{\varepsilon}$ leaves.
- The leaves are (nearly) uniformly distributed in G.

Idea: Query the leaves!

Problems:

- There are errors on the leaves.

The expander walk as a tree



True Statements:

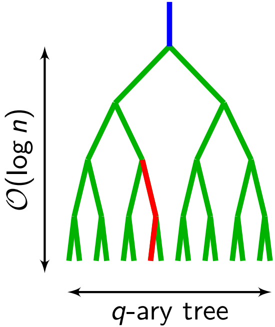
- The symbols on the leaves determine the symbol on the root.
- There are $q^{\mathcal{O}(\log(n))} \approx N^{\varepsilon}$ leaves.
- The leaves are (nearly) uniformly distributed in G.

Idea: Query the leaves!

Problems:

- There are errors on the leaves.
- Errors on the leaves propagate.

The expander walk as a tree



True Statements:

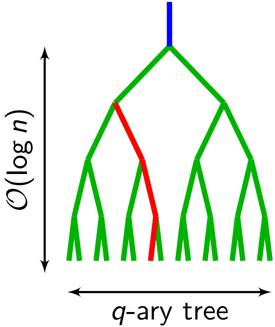
- The symbols on the leaves determine the symbol on the root.
- There are $q^{\mathcal{O}(\log(n))} \approx N^{\varepsilon}$ leaves.
- The leaves are (nearly) uniformly distributed in G.

Idea: Query the leaves!

Problems:

- There are errors on the leaves.
- Errors on the leaves propagate.

The expander walk as a tree



True Statements:

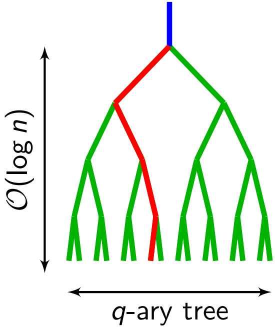
- The symbols on the leaves determine the symbol on the root.
- There are $q^{\mathcal{O}(\log(n))} \approx N^{\varepsilon}$ leaves.
- The leaves are (nearly) uniformly distributed in G.

Idea: Query the leaves!

Problems:

- There are errors on the leaves.
- Errors on the leaves propagate.

The expander walk as a tree



True Statements:

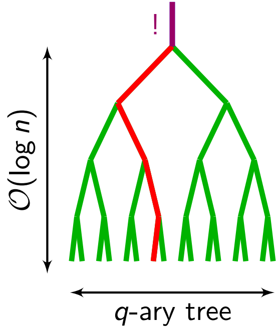
- The symbols on the leaves determine the symbol on the root.
- There are $q^{\mathcal{O}(\log(n))} \approx N^{\varepsilon}$ leaves.
- The leaves are (nearly) uniformly distributed in G.

Idea: Query the leaves!

Problems:

- There are errors on the leaves.
- Errors on the leaves propagate.

The expander walk as a tree



True Statements:

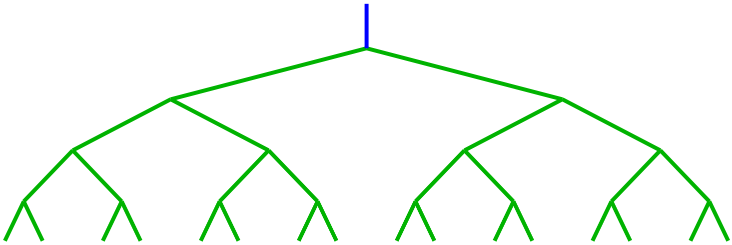
- The symbols on the leaves determine the symbol on the root.
- There are $q^{\mathcal{O}(\log(n))} \approx N^{\varepsilon}$ leaves.
- The leaves are (nearly) uniformly distributed in G.

Idea: Query the leaves!

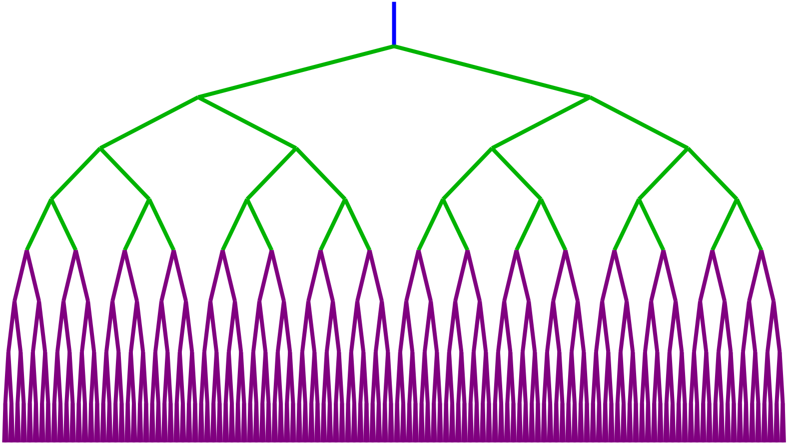
Problems:

- There are errors on the leaves.
- Errors on the leaves propagate.

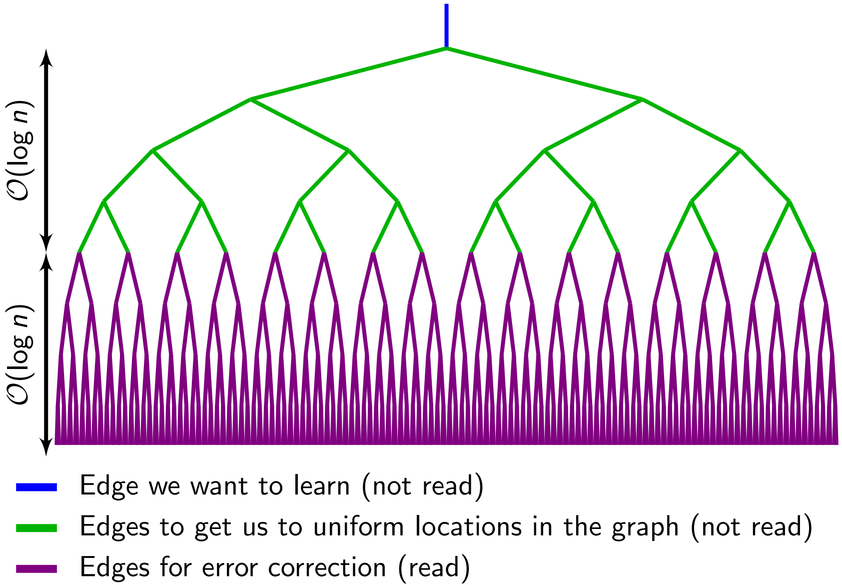
Correcting the last layer



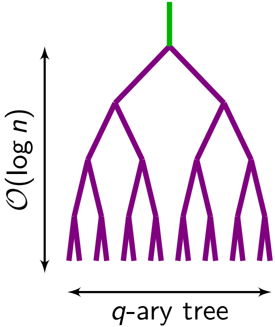
Correcting the last layer



Correcting the last layer

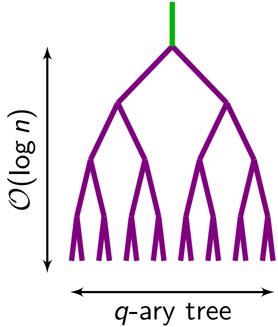


Why should this help?



- **Now** the queries can tolerate a few errors.

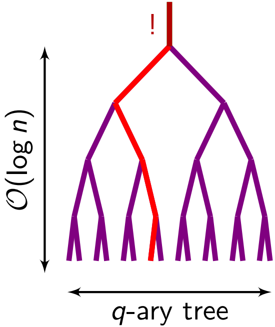
Why should this help?



False statement:

- ~~**Now** the queries can tolerate a few errors.~~

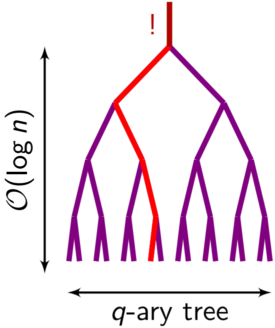
Why should this help?



False statement:

- ~~**Now** the queries can tolerate a few errors.~~

Why should this help?



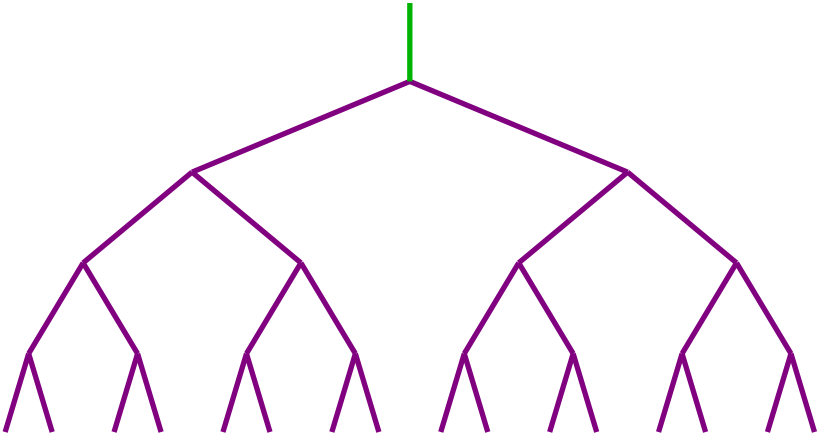
False statement:

- ~~**Now** the queries can tolerate a few errors.~~

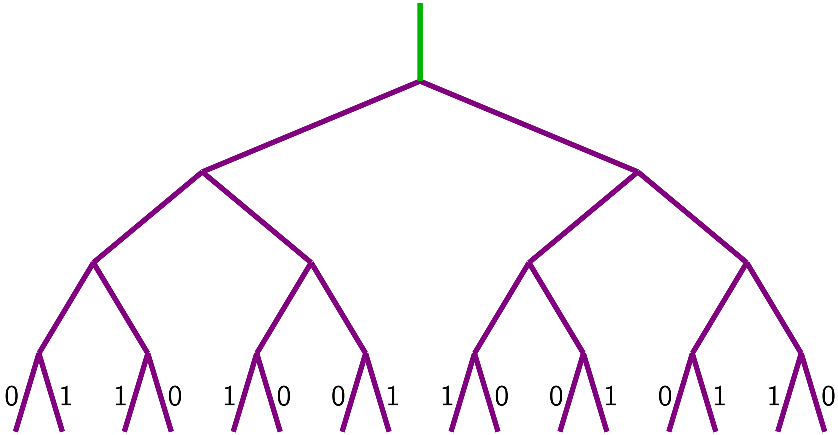
True statements:

- This is basically the only thing that can go wrong.
- Because everything in sight is (nearly) uniform, it probably won't go wrong.

Decoding algorithm

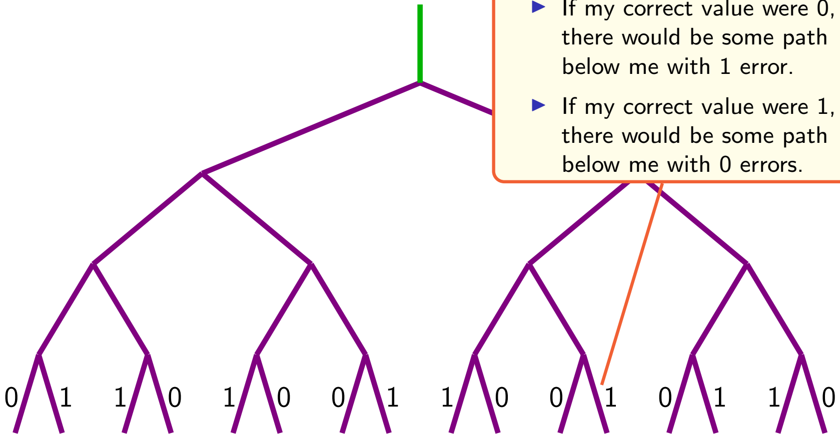


Decoding algorithm



Each leaf edge queries its symbol

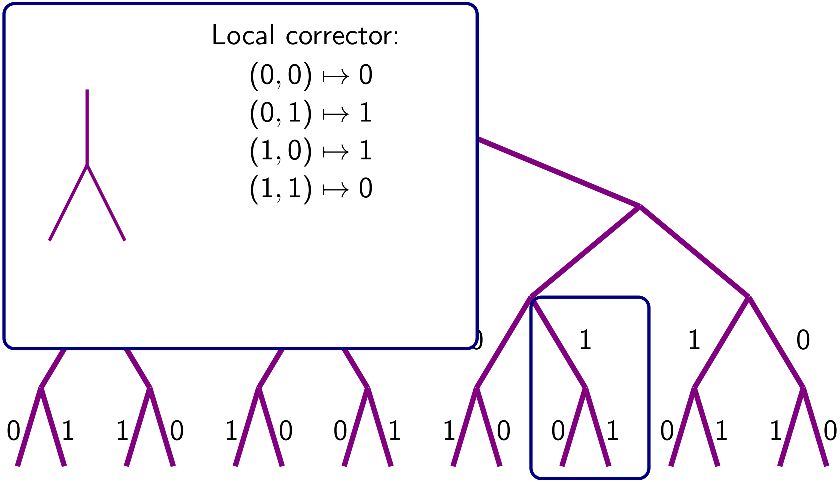
Decoding algorithm



- If my correct value were 0, there would be some path below me with 1 error.
- If my correct value were 1, there would be some path below me with 0 errors.

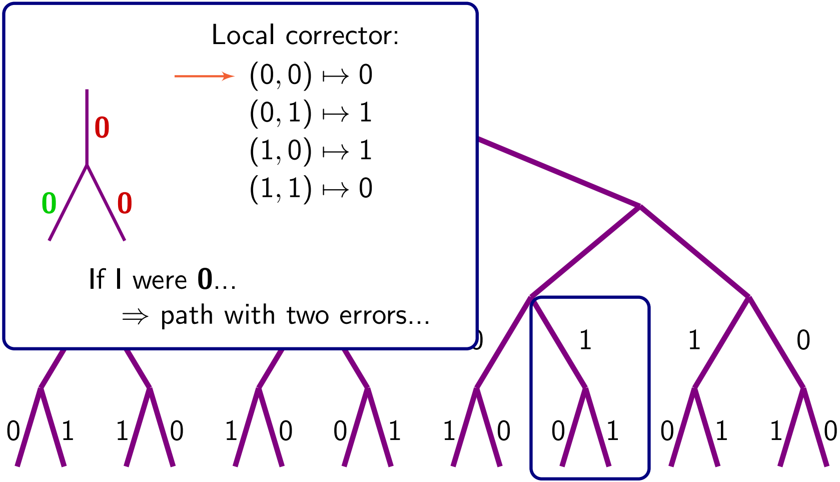
Each leaf edge thinks to itself...

Decoding algorithm



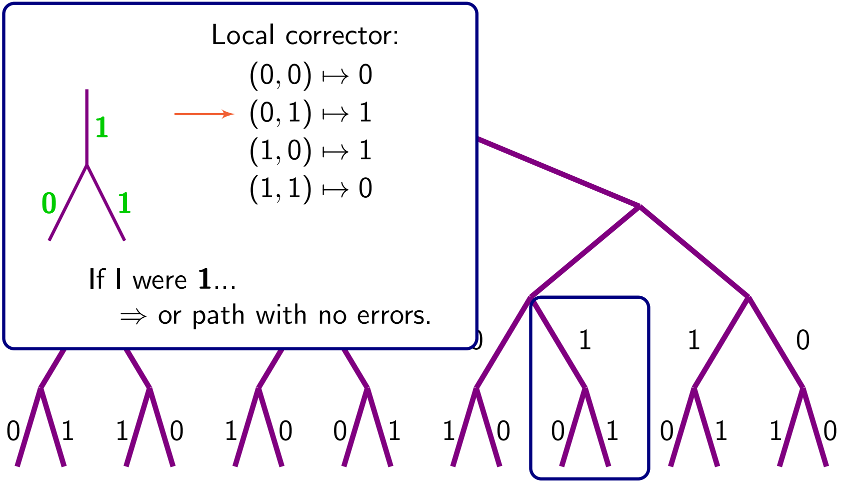
Each second-level edge reads its symbol and thinks to itself...

Decoding algorithm



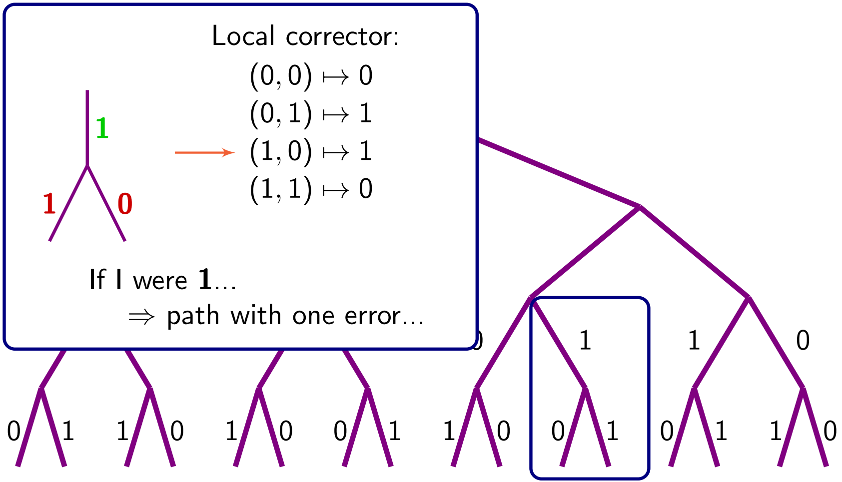
Each second-level edge reads its symbol and thinks to itself...

Decoding algorithm



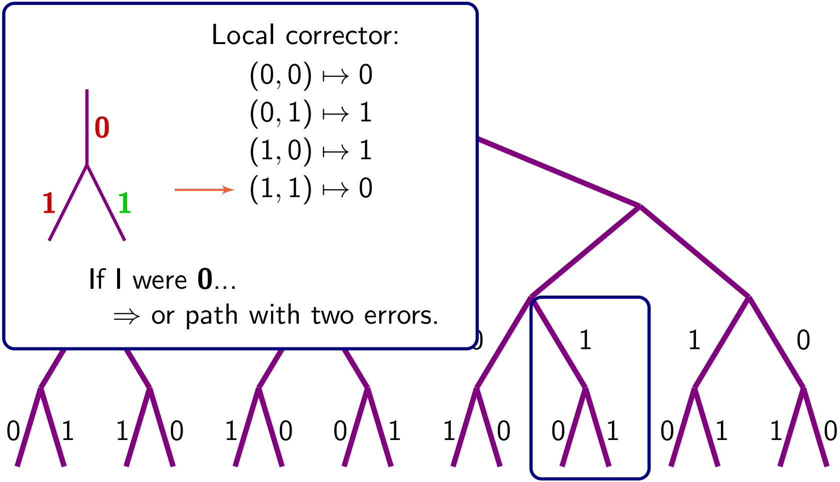
Each second-level edge reads its symbol and thinks to itself...

Decoding algorithm



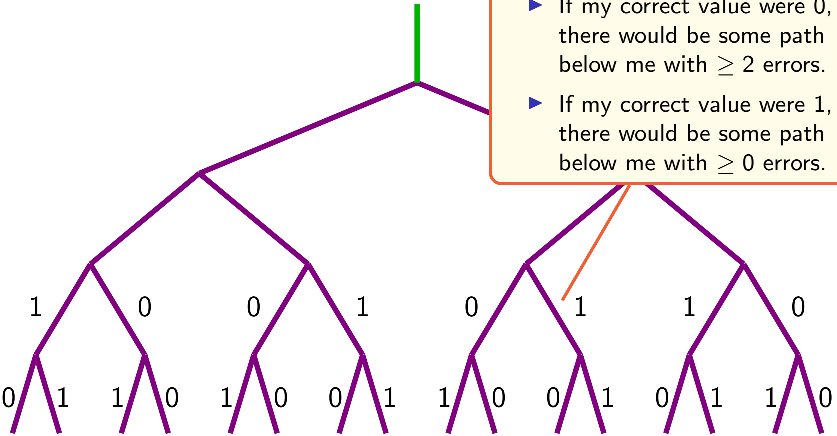
Each second-level edge reads its symbol and thinks to itself...

Decoding algorithm



Each second-level edge reads its symbol and thinks to itself...

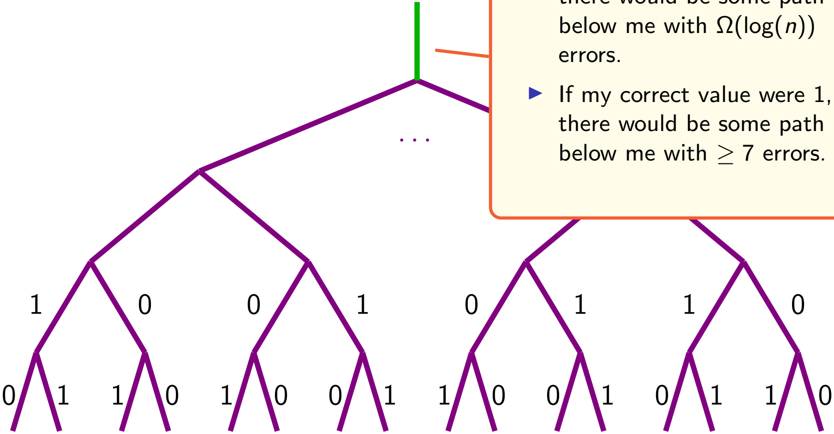
Decoding algorithm



- If my correct value were 0, there would be some path below me with ≥ 2 errors.
- If my correct value were 1, there would be some path below me with ≥ 0 errors.

Each second-level edge reads its symbol and thinks to itself...

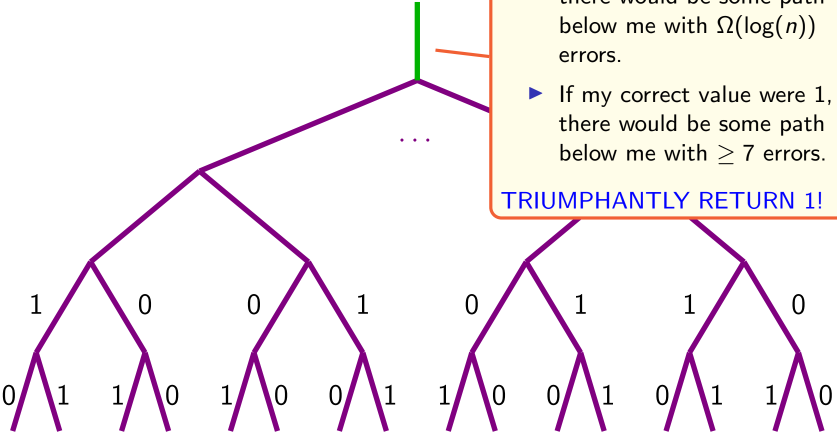
Decoding algorithm



- If my correct value were 0, there would be some path below me with $\Omega(\log(n))$ errors.
- If my correct value were 1, there would be some path below me with ≥ 7 errors.

etc.

Decoding algorithm



- If my correct value were 0, there would be some path below me with $\Omega(\log(n))$ errors.
- If my correct value were 1, there would be some path below me with ≥ 7 errors.

TRIUMPHANTLY RETURN 1!

This only fails if there exist a *path* that is heavily corrupted.
Heavily corrupted paths occur with exponentially small probability.

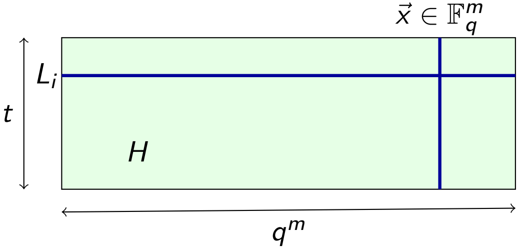
Outline

1. Local correctability
   - Definitions and notation
   - Example: Reed-Muller codes
   - Previous work and our contribution
2. Expander codes
3. Local correctability of expander codes
   - Requirement for the inner code: smooth reconstruction
   - Decoding algorithm
   - Example instantiation: finite geometry codes
4. Conclusion

One choice for inner code: based on affine geometry

See [Assmus, Key '94,'98] for a nice overview

- Let $L_1, \ldots, L_t$ be the r-dimensional affine subspaces of $\mathbb{F}_q^m$, and consider the code with parity-check matrix H:

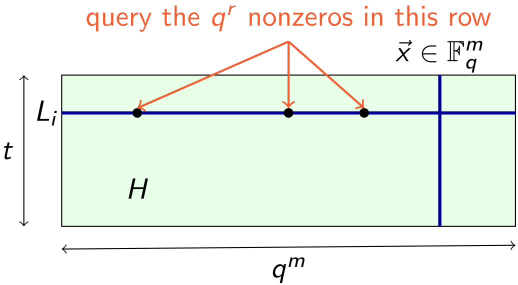


$$H_{i,\vec{x}} = \begin{cases} 1 & \vec{x} \in L_i \\ 0 & \vec{x} \notin L_i \end{cases}$$

One choice for inner code: based on affine geometry

See [Assmus, Key '94,'98] for a nice overview

- Let $L_1, \ldots, L_t$ be the r-dimensional affine subspaces of $\mathbb{F}_q^m$, and consider the code with parity-check matrix H:



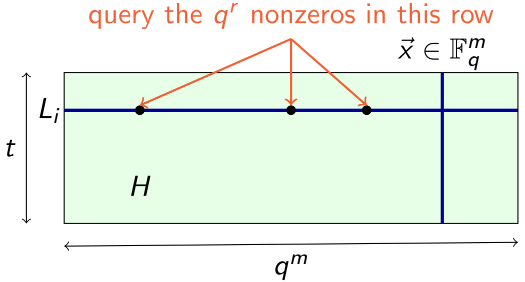
$$H_{i,\vec{x}} = \begin{cases} 1 & \vec{x} \in L_i \\ 0 & \vec{x} \notin L_i \end{cases}$$

- Smooth reconstruction: To learn a coordinate indexed by $\vec{x} \in \mathbb{F}_q^m$:
  - pick a random r-flat, L_i, containing $\vec{x}$.
  - query all of the points in L_i.

One choice for inner code: based on affine geometry

See [Assmus, Key '94,'98] for a nice overview

- Let $L_1, \ldots, L_t$ be the r-dimensional affine subspaces of $\mathbb{F}_q^m$, and consider the code with parity-check matrix H:



$$H_{i,\vec{x}} = \begin{cases} 1 & \vec{x} \in L_i \\ 0 & \vec{x} \notin L_i \end{cases}$$

- Smooth reconstruction: To learn a coordinate indexed by $\vec{x} \in \mathbb{F}_q^m$:
  - pick a random r-flat, L_i, containing $\vec{x}$.
  - query all of the points in L_i.
- Observe: This is *not* a very good LCC!

One good instantiation

Graph:

- Ramanujan graph

Inner code:

- Finite geometry code

Results:

For any $\alpha, \epsilon > 0$, for infinitely many N, we get a code with block length N, which

- has rate $1 - \alpha$
- has locality $(N/d)^{\epsilon}$
- tolerates constant error rate

Outline

1. Local correctability
   - Definitions and notation
   - Example: Reed-Muller codes
   - Previous work and our contribution
2. Expander codes
3. Local correctability of expander codes
   - Requirement for the inner code: smooth reconstruction
   - Decoding algorithm
   - Example instantiation: finite geometry codes
4. Conclusion

Summary

- When the inner code has smooth reconstruction, we give a local-decoding procedure for expander codes.
- This gives a new (and yet old!) family of linear locally correctable codes of rate approaching 1.

Open questions

- Can we use expander codes to achieve local correctability with lower query complexity?
- Can we use inner codes with rate $< 1/2$?

The end