Codes on graphs and iterative decoding

Bane Vasić

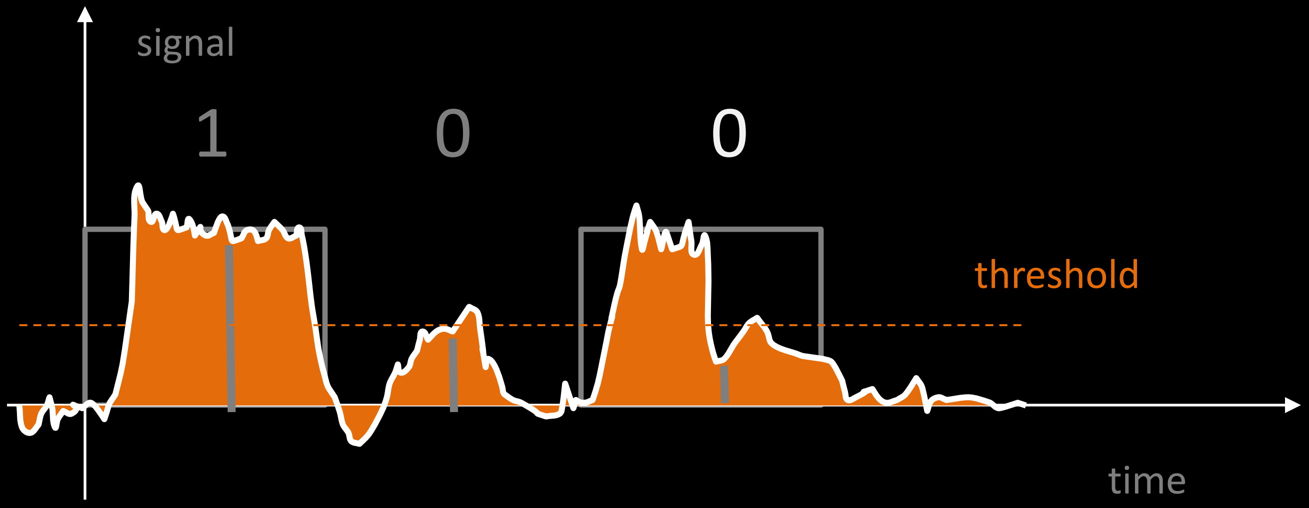
Error Correction Coding Laboratory
University of Arizona

THE UNIVERSITY OF ARIZONA®
TUCSON ARIZONA

Prelude

Information transmission

Information transmission



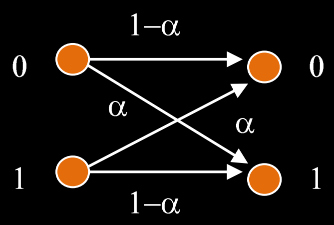
Noisy memoryless channels

$$x_1\ x_2\ x_3\ x_4\ \ldots \longrightarrow \boxed{\mathrm{p}(\ y_{\mathrm{i}}\ |\ x_{\mathrm{i}}\ )} \longrightarrow y_1\ y_2\ y_3\ y_4\ \ldots$$

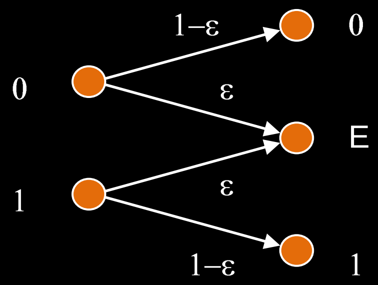
$$p\ \ y_1, \ldots, y_n \mid x_1, \ldots, x_n\ \ = \prod_{i=1}^{n} p\ \ y_j \mid x_i$$

Simple memoryless channels

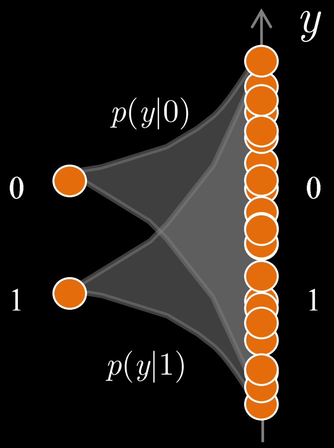
- Binary symmetric channel (BSC)



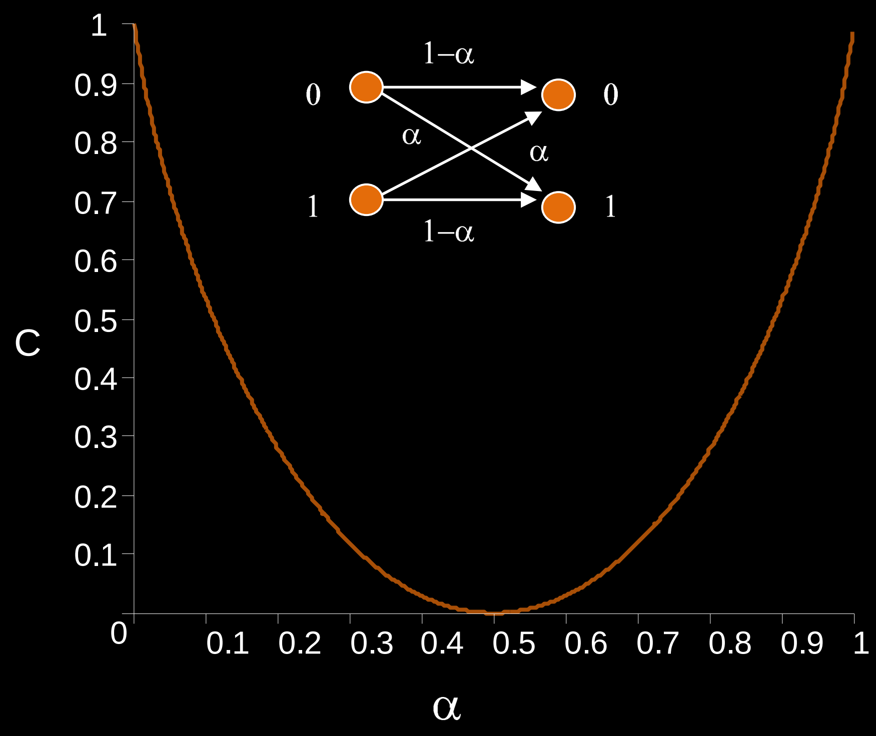
- Binary erasure channel (BEC)



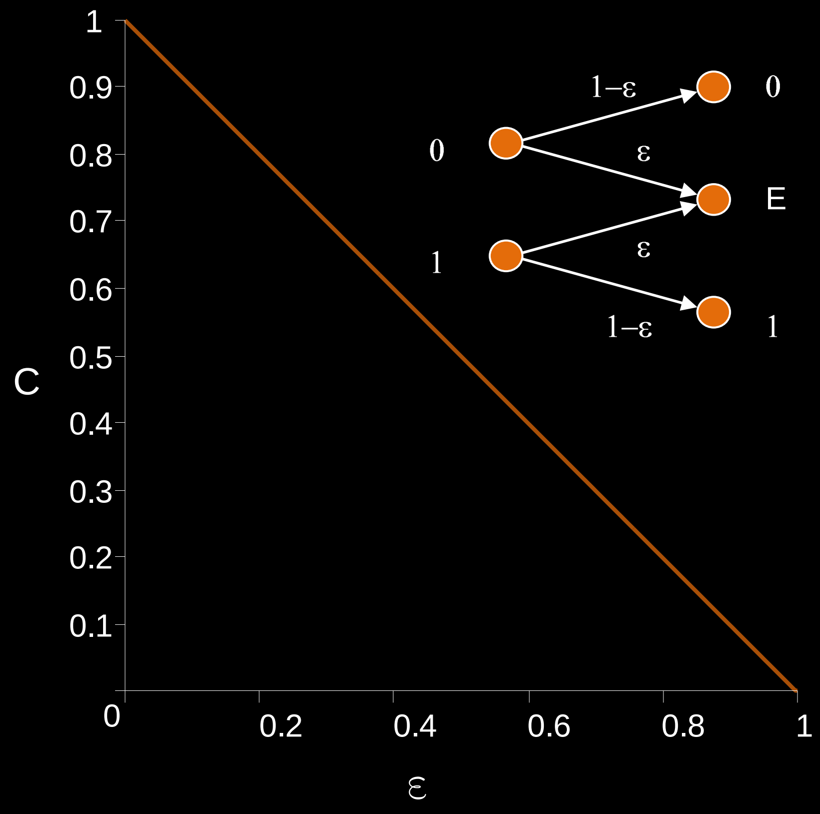
- Binary input additive white Gaussian noise (AWGN) channel, σ^2



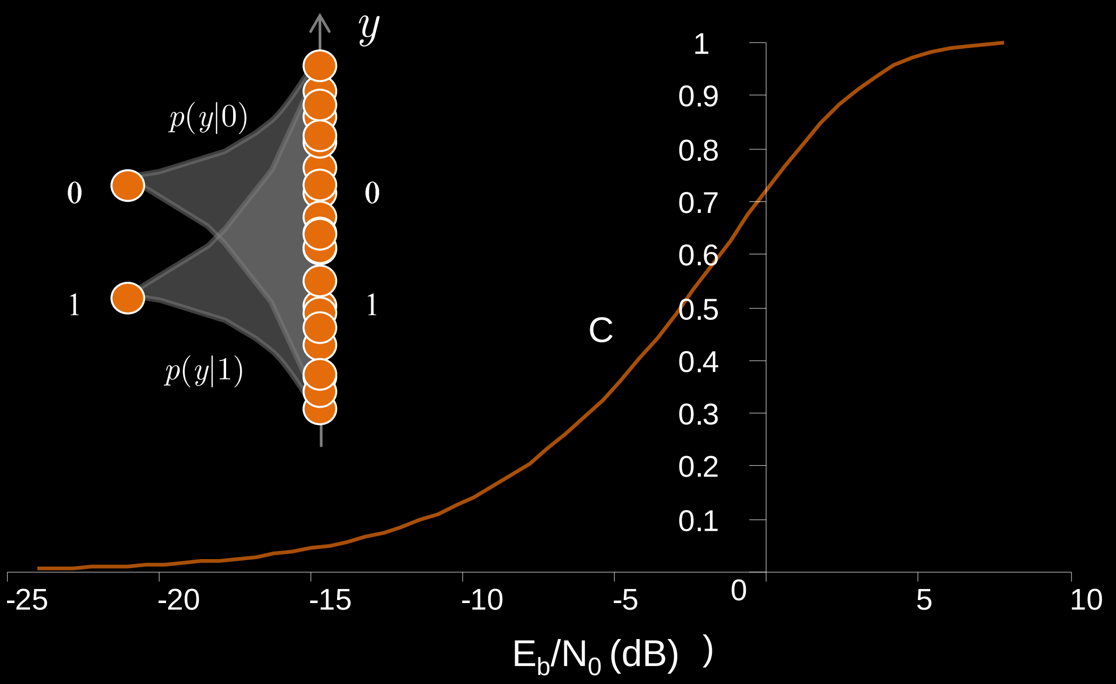
Channel capacity - BSC



Channel capacity - BEC



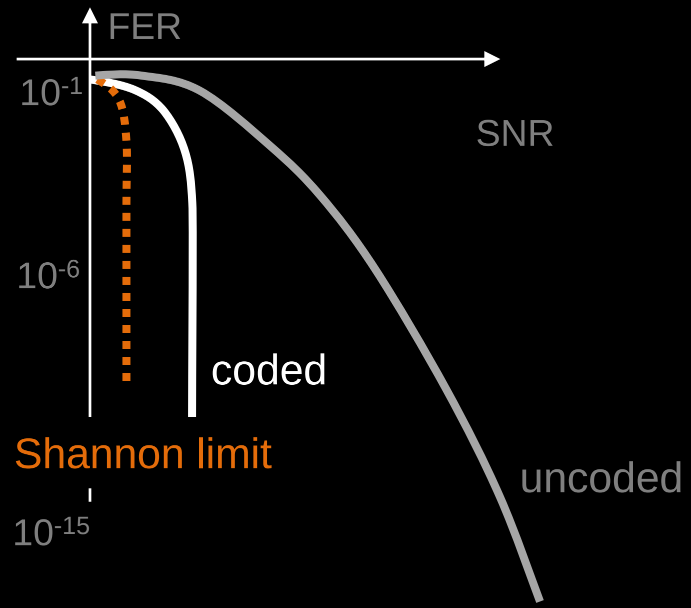
Channel capacity - BAWGN



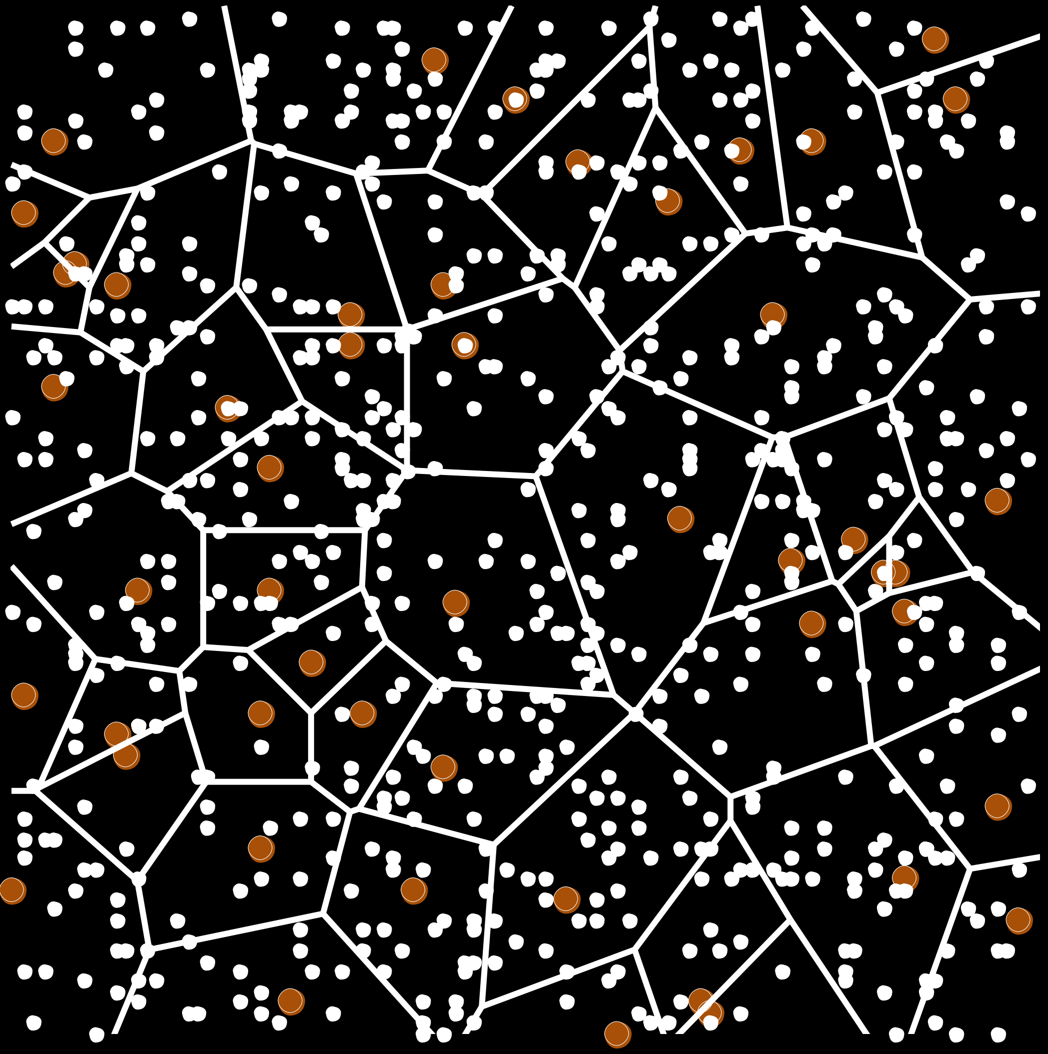
Error correction coding

- Message $m = (m_1, \ldots, m_k)$
- Codeword $x = (x_1, \ldots, x_n)$
- Received word $y = (y_1, \ldots, y_n)$
- Code rate $R = \frac{k}{n}$
- The decoder tries to find x (or m) from y so that the probability of bit/codeword error is minimal.
- In other words, decoder tries to find a codeword “closest” to y.

Error rate performance

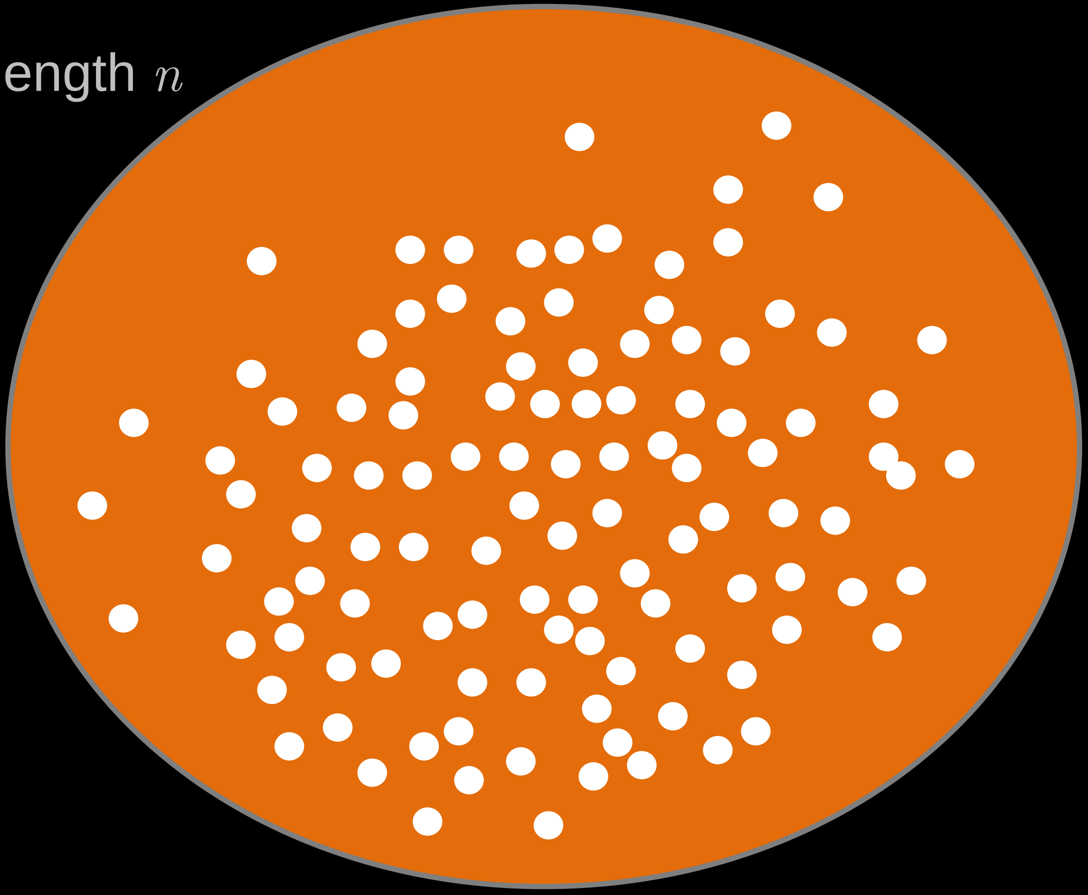


Maximum likelihood decoding

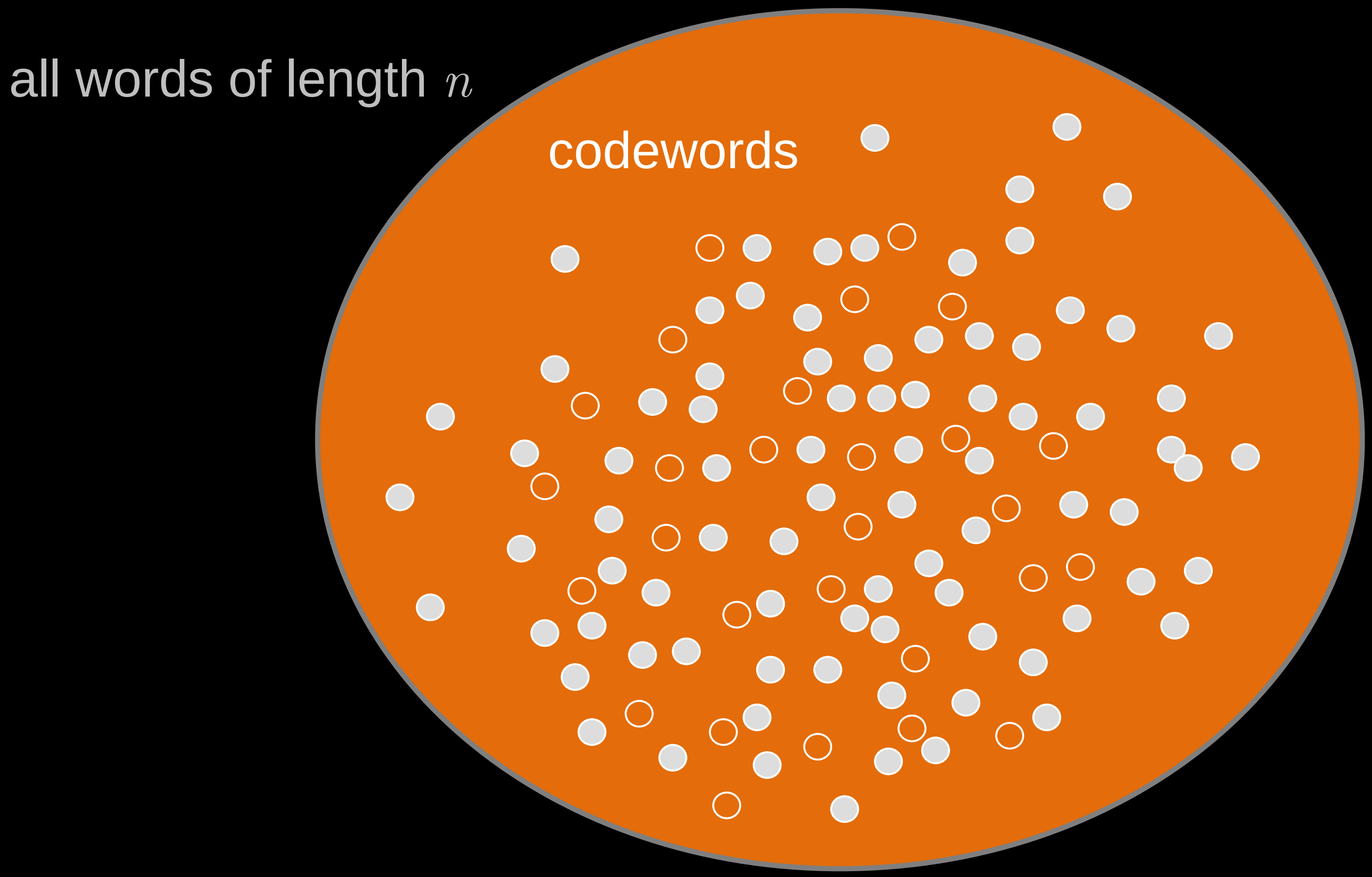


Protecting information by coding

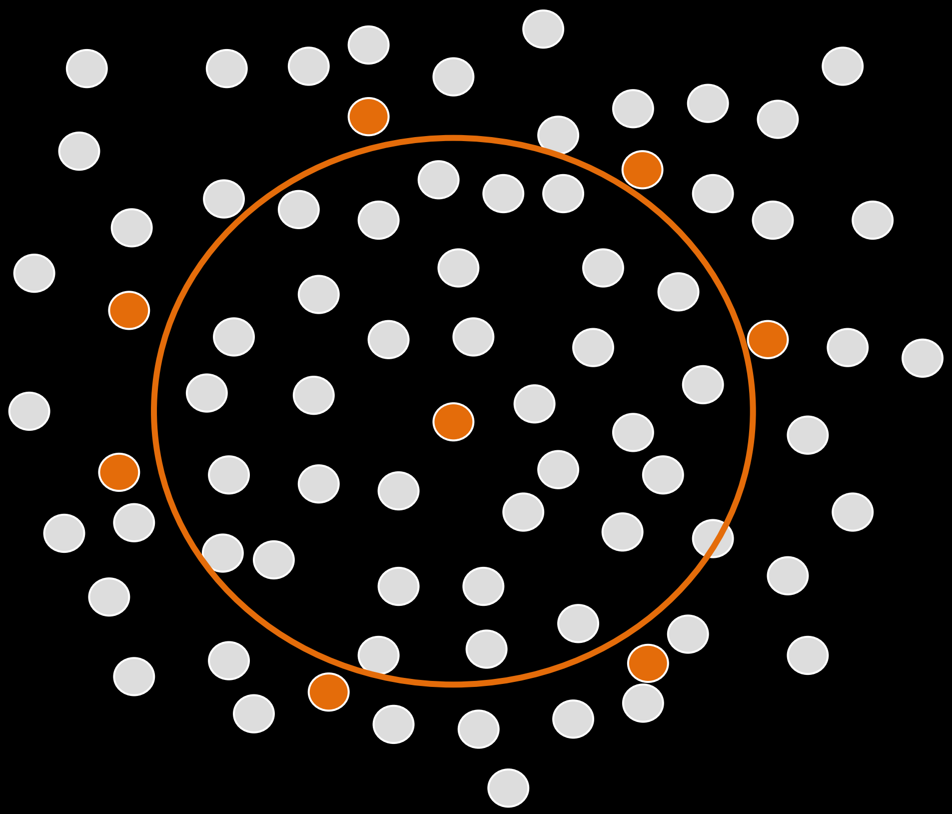
all words of length n



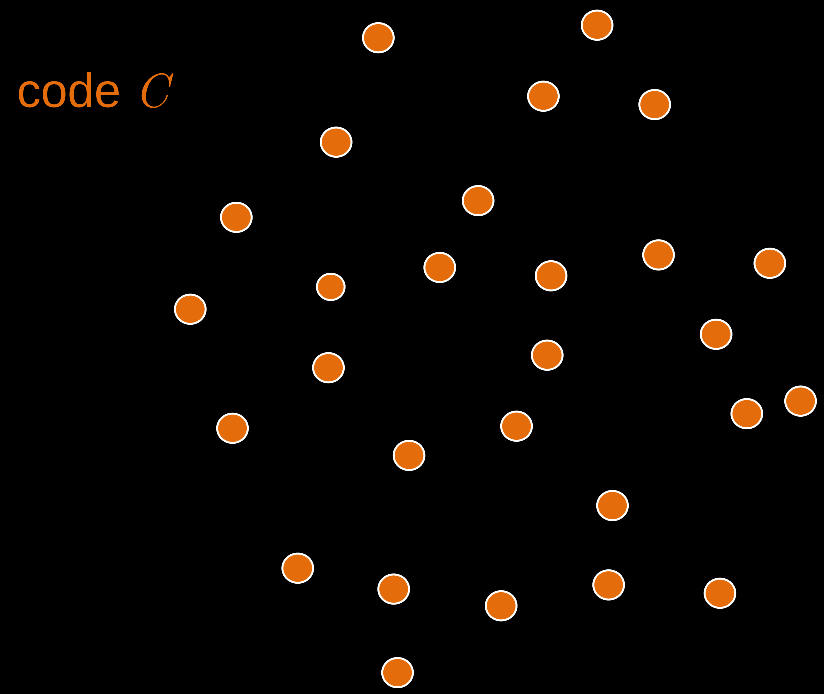
Protecting information by coding



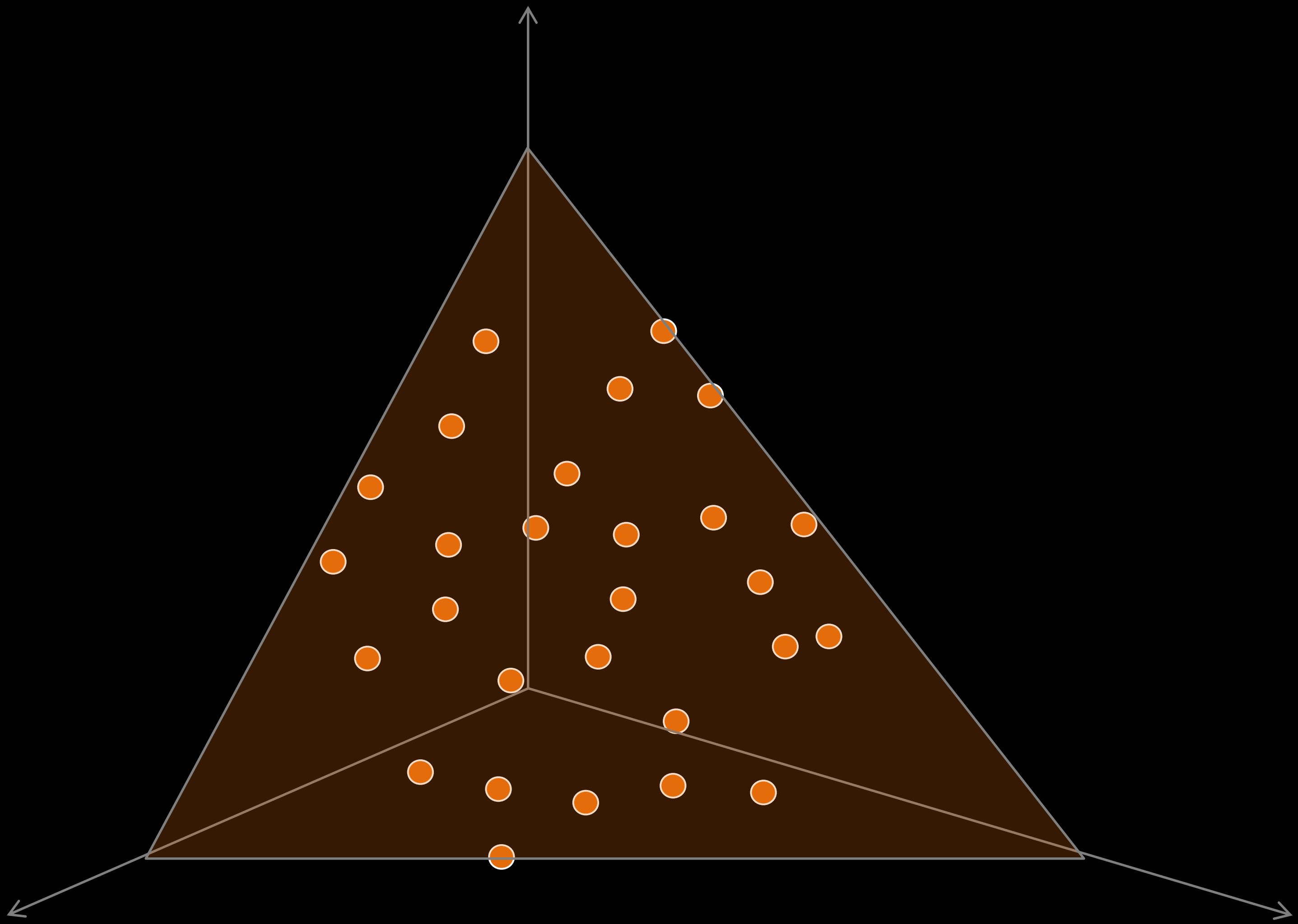
Minimum distance



Protecting information by coding



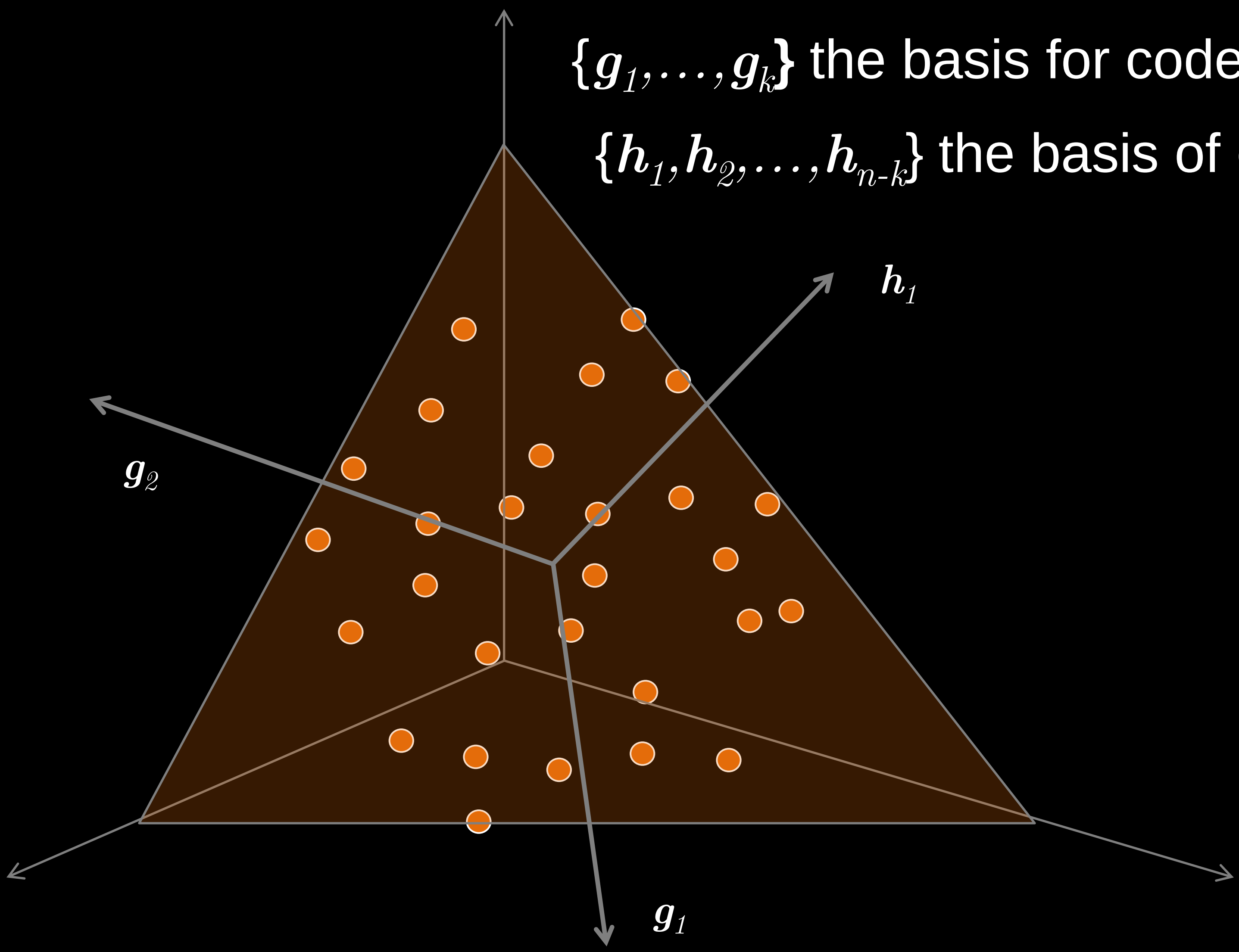
Linear block codes



Dimension of a linear block code

$\{\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k\}$ the basis for code C

$\{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_{n-k}\}$ the basis of $C^\perp$



Encoding

$$x = m_1 g_1 + m_2 g_2 + \ldots m_k g_k$$

$$G = \begin{bmatrix} 10101 \\ 00110 \end{bmatrix}$$

$$x = (m_1, m_2, \ldots, m_k) \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix}$$

$$\begin{matrix} m = (0,0) \\ m = (0,1) \\ m = (1,0) \\ m = (1,1) \end{matrix} \rightarrow \begin{matrix} x = (0,0,0,0,0) \\ x = (1,0,1,0,1) \\ x = (0,0,1,1,0) \\ x = (1,0,0,1,1) \end{matrix}$$

$$x = mG$$

$$G = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix}$$

Generator matrix

an k x n matrix of rank k

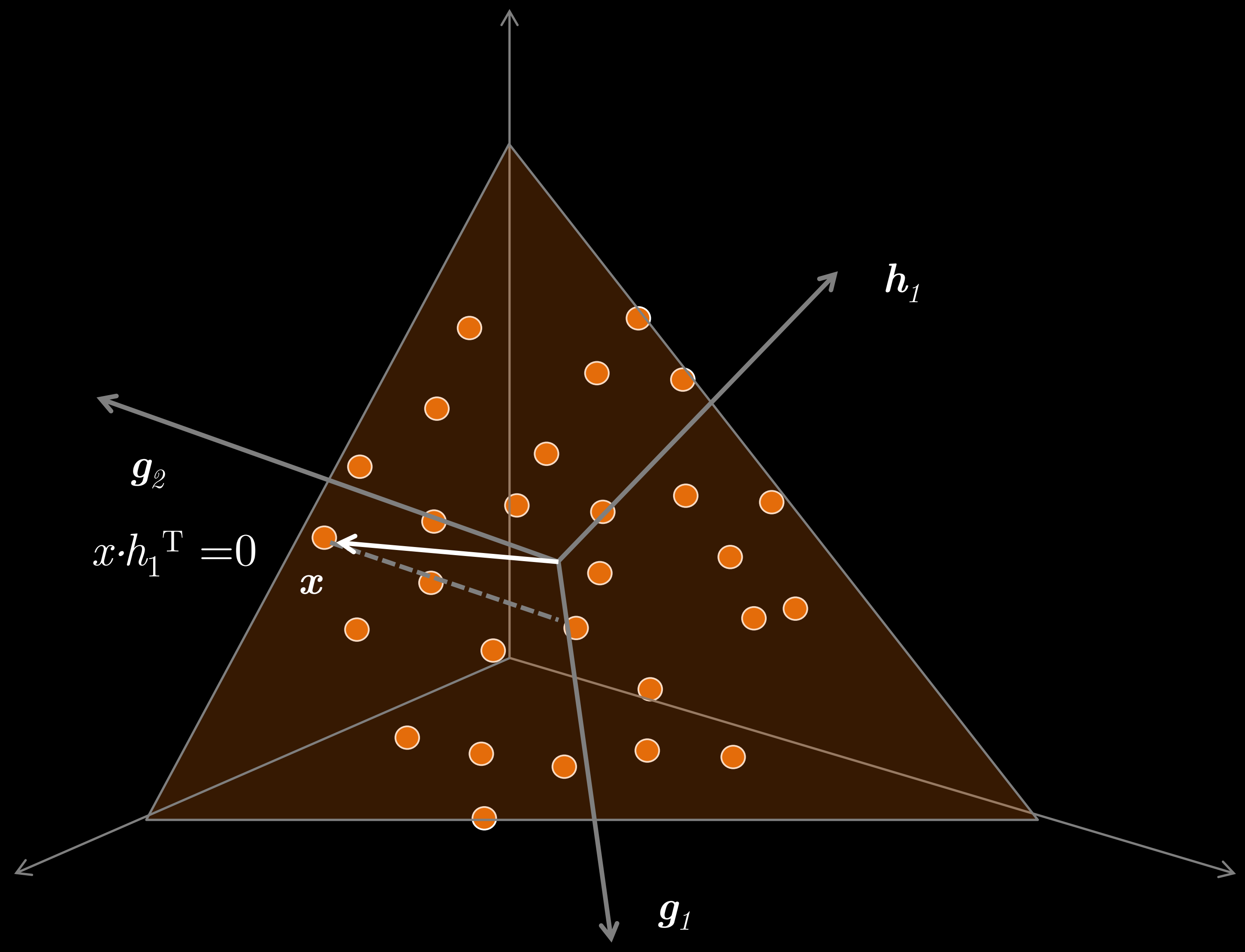
a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	0

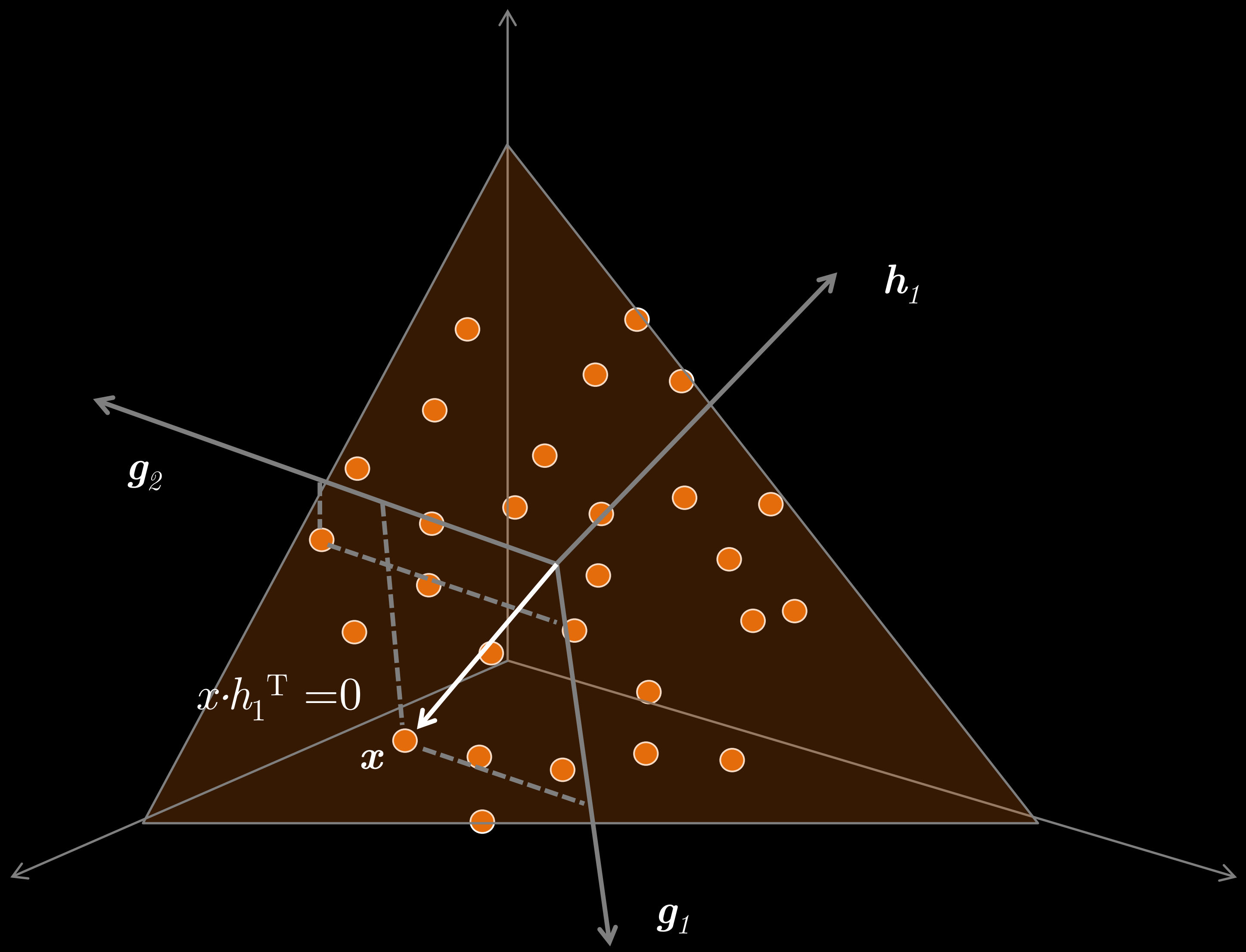
Linear block codes as subspaces

- Given a GF(2) (ground field), we define the vector space - the *n*-tuple **v**=($v_1, v_2, \ldots v_n$) of elements from the ground filed is a type of vector.
- Elias and Golay: A binary linear (n,k) code *C* is a k -dimensional subspace of a vector space Galois Field, GF(2).

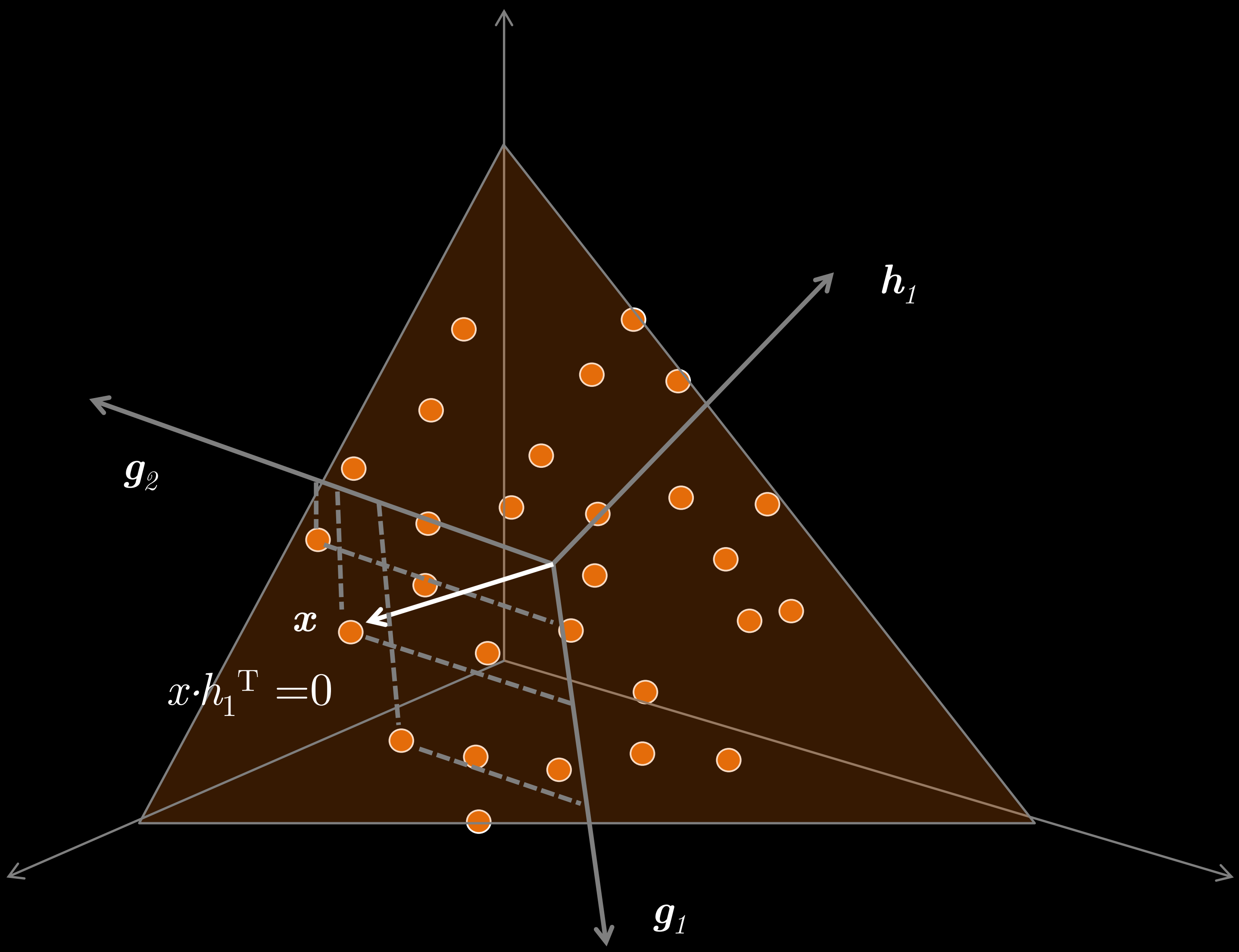
Parity check



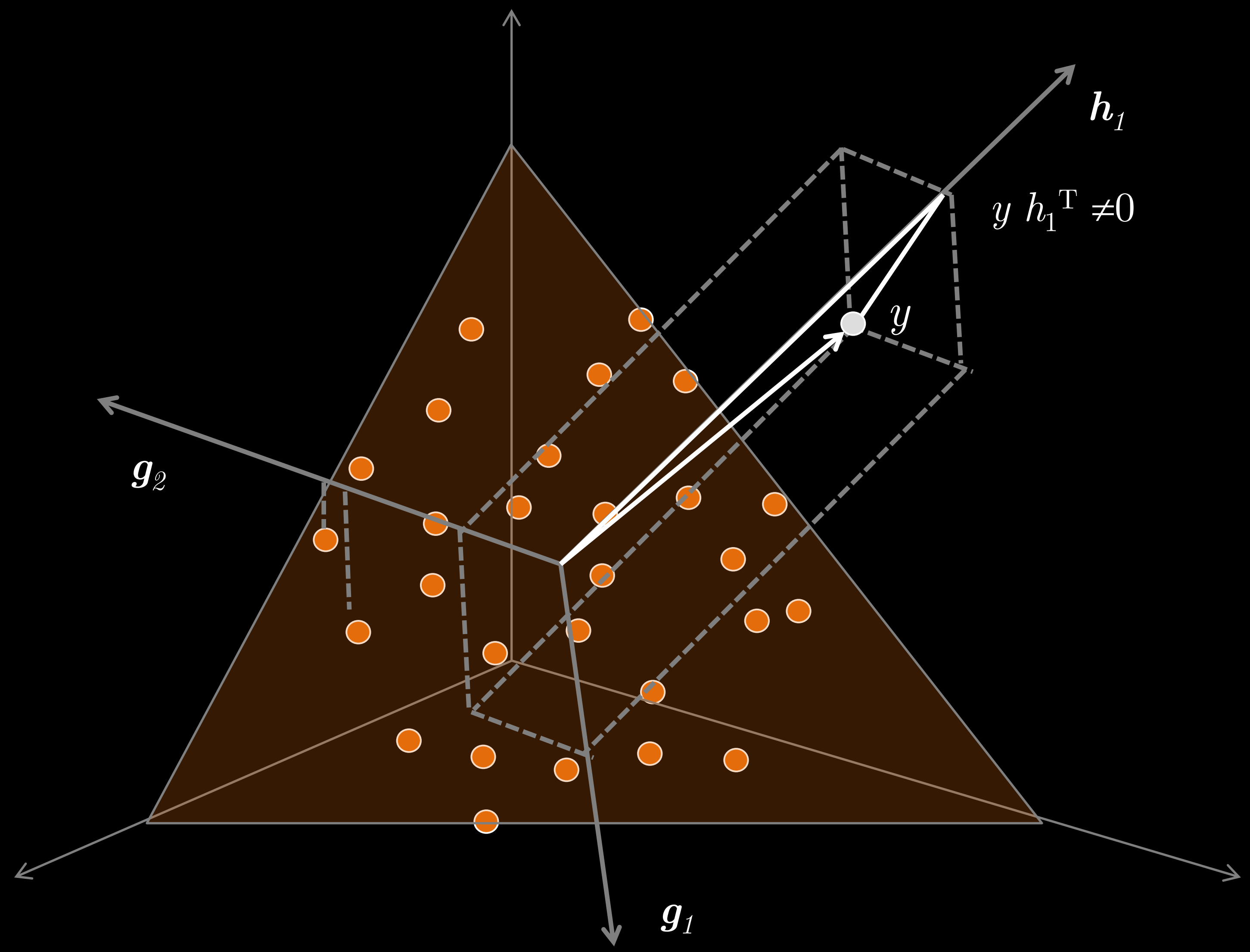
Parity check



Parity check



Syndrome


$\boldsymbol{h}_1$
$y\ h_1^{\mathrm{T}} \neq 0$
y
$\boldsymbol{g}_2$
$\boldsymbol{g}_1$

Dual code $C^\perp$

- Let x be a codeword

$$xh_1^{\mathrm{T}} = 0 \quad xh_2^{\mathrm{T}} = 0 \qquad xh_{n-k}^{\mathrm{T}} = 0$$

$$H = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{n-k} \end{bmatrix}$$ parity check matrix

$$xH^{\mathrm{T}} = 0$$

- A received vector which is not a codeword results in a nonzero <u>*syndrome.*</u>

$$y \neq x \Rightarrow yH^{\mathrm{T}} \neq 0$$

Linear constraints

- A codeword x satisfies $v \cdot H^T = 0$
- n-k equations in n variables
- Example:

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$\begin{aligned} c_1 &: \quad x_1 + x_4 + x_6 + x_7 = 0 \\ c_2 &: \quad x_2 + x_4 + x_5 + x_6 = 0 \\ c_3 &: \quad x_3 + x_5 + x_6 + x_7 = 0 \end{aligned}$$

Side observations

- Since $xH^{\mathrm{T}} = 0$ for any codeword x.
- and since $x = mG$ it follows $GH^{\mathrm{T}} = 0$
  - H can be found from G.
- For any $a, b \in \{0,1\}$ $x(ah_i^{\mathrm{T}} + bh_j^{\mathrm{T}}) = 0$

$$H' = \begin{bmatrix} H \\ ah_i + bh_j \end{bmatrix}$$

$$xH'^{\mathrm{T}} = 0$$

- The parity check matrix can be modified by adding linear combinations of its rows.
- The ranks of any such new parity matrix is still n-k.

LDPC code basics

Applications of LDPC codes

- Wireless networks, satellite communications, deep-space communications, power line communications are among applications where the low-density parity check (LDPC) codes are the standardized. Standards include: Digital video broadcast over satellite (DVB-S2 Standard) and over cable (DVB-C2 Standard), terrestrial television broadcasting (DVB-T2, DVB-T2-Lite Standards), GEO-Mobile Radio (GMR) satellite telephony (GMR-1 Standard), local and metropolitan area networks (LAN/MAN) (IEEE 802.11 (WiFi)), wireless personal area networks (WPAN) (IEEE 802.15.3c (60 GHz PHY)), wireless local and metropolitan area networks (WLAN/WMAN) (IEEE 802.16 (Mobile WiMAX), near-earth and deep space communications (CCSDS), wire and power line communications (ITU-T G.hn (G.9960)), utra-wide band technologies (WiMedia 1.5 UWB), magnetic hard disk drives, optical communications, flash memories.

Outline

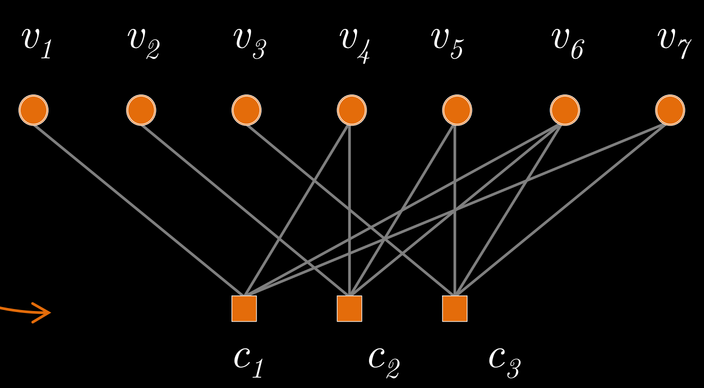
- Basics
  - Error correction codes, linear block codes, parity check matrices, code graphs
  - Decoding using local information, iterative decoders
  - Decoders as finite-state dynamical systems, basins of attraction and decoding failures
- Failures of iterative decoders
  - Correcting number of errors linear in code length
  - Finite length analysis
  - Trapping sets
- Code design
  - Combinatorial designs and codes
  - Quasi-cyclic codes designed from group-theoretic transforms, Latin squares, difference families, finite geometries

Graphical model for a linear block code

$$
\begin{aligned}
c_1 &: \quad v_1 + v_4 + v_6 + v_7 = 0 \\
c_2 &: \quad v_2 + v_4 + v_5 + v_6 = 0 \\
c_3 &: \quad v_3 + v_5 + v_6 + v_7 = 0
\end{aligned}
$$

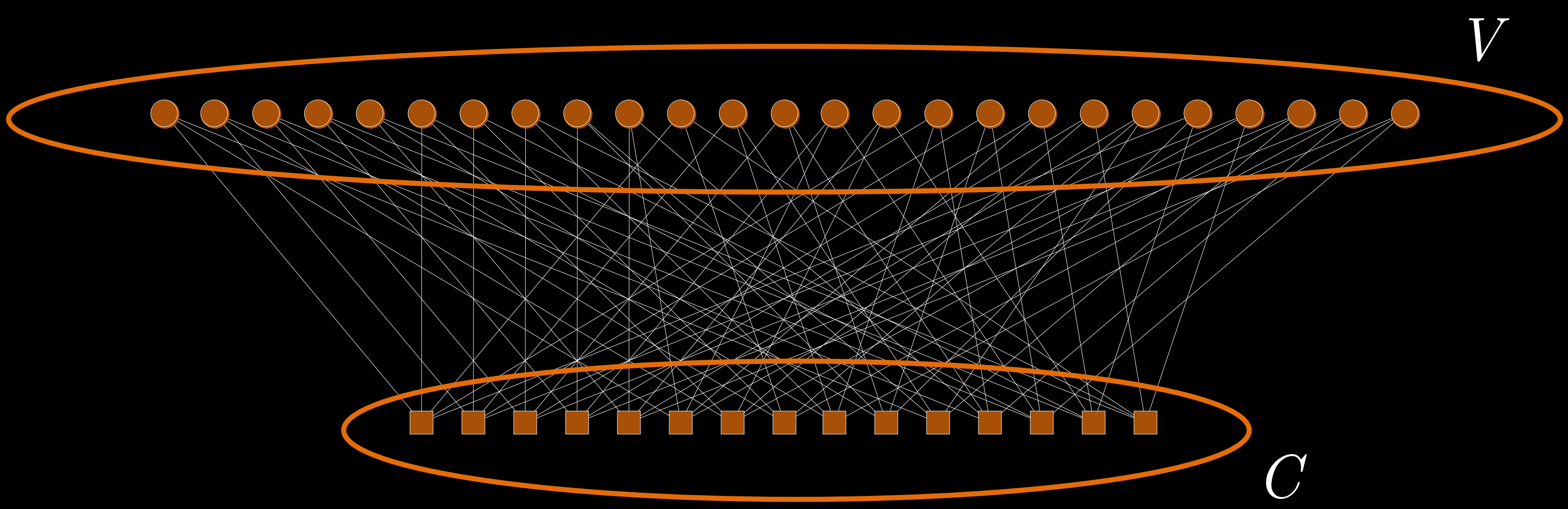
Checks

Variables



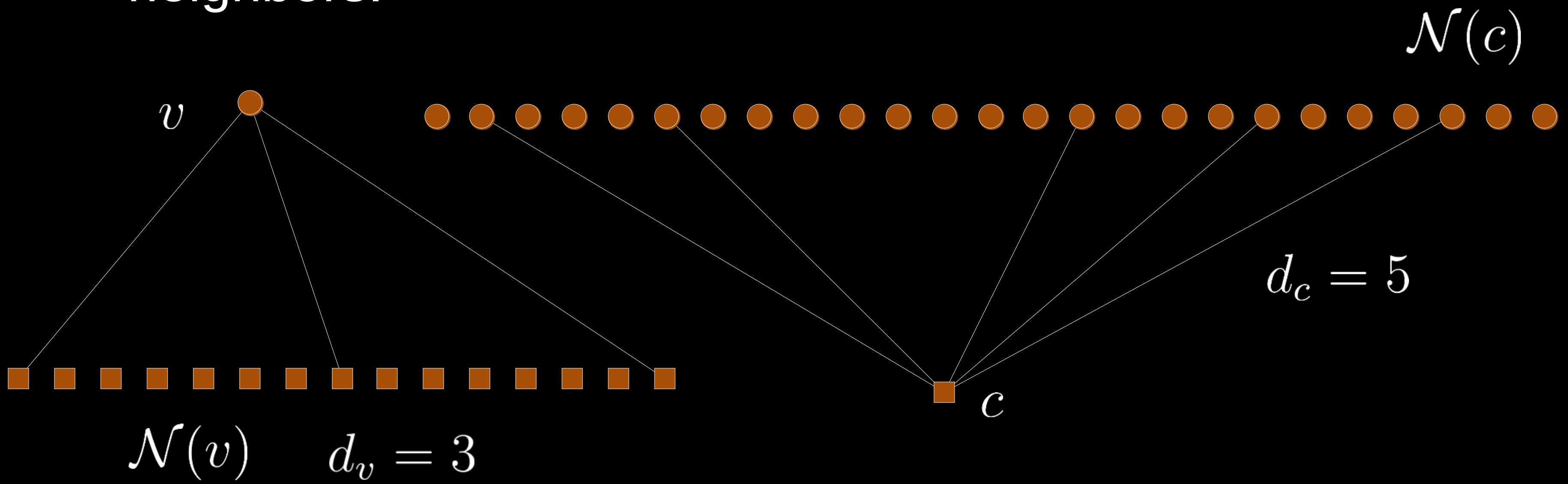
Definitions

- LDPC codes belong to the class of linear block codes which can be defined by sparse bipartite graphs.
- The *Tanner* graph of an LDPC code $\mathcal{C}$ is a bipartite graph G with two sets of nodes:
  - the set of variable nodes $V = \{1, 2, \ldots, n\}$
  - and the set of check nodes $C = \{1, 2, \ldots, m\}$



Definitions

- The check nodes (variable nodes resp.) connected to a variable node (check node resp.) are referred to as its neighbors.
- The set of neighbors of a node u is denoted by $\mathcal{N}(u)$
- The degree d_u of a node u is the number of its neighbors.

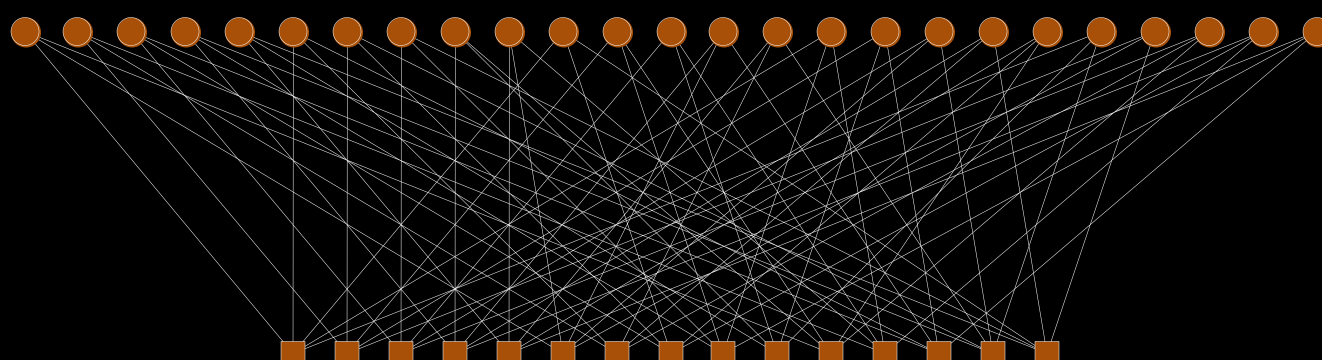


Definitions

- A vector $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ is a codeword if and only if for each check node, the modulo two sum of its neighbors is zero.
- An (n, γ, ρ) regular LDPC code has a Tanner graph with n variable nodes each of degree γ and $n\gamma/\rho$ check nodes each of degree ρ .
- This code has length n rate $r \geq 1 - \gamma/\rho$
- The Tanner graph is not uniquely defined by the code and when we say the Tanner graph of an LDPC code, we only mean one possible graphical representation.

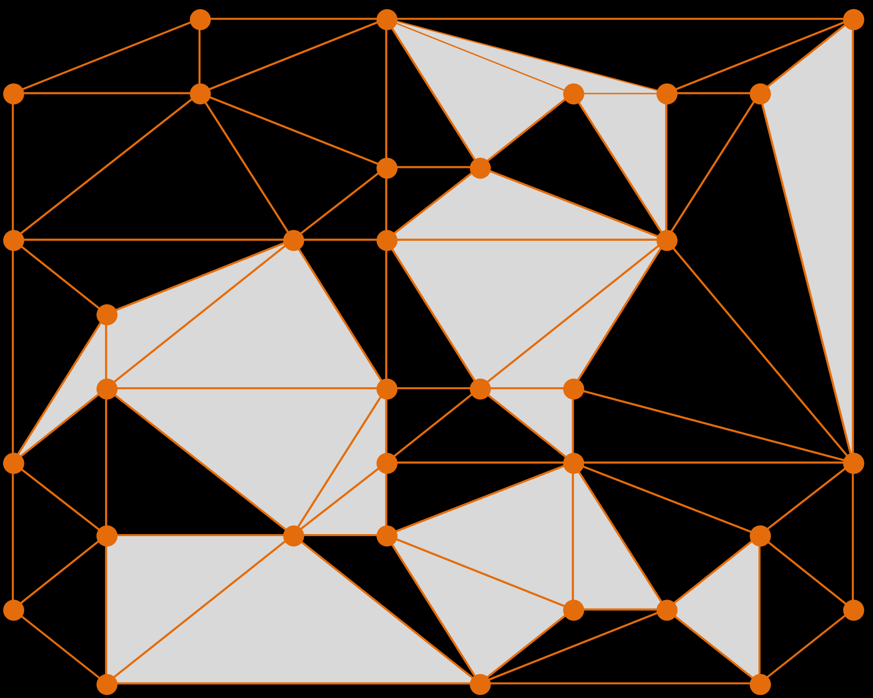
An example of a regular n=25 γ=3, ρ=5 code

$$
H = \left[\begin{array}{ccccc:ccccc:ccccc:ccccc:ccccc}
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
\hdashline
1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0\\
\hdashline
1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0\\
0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&0
\end{array}\right]
$$

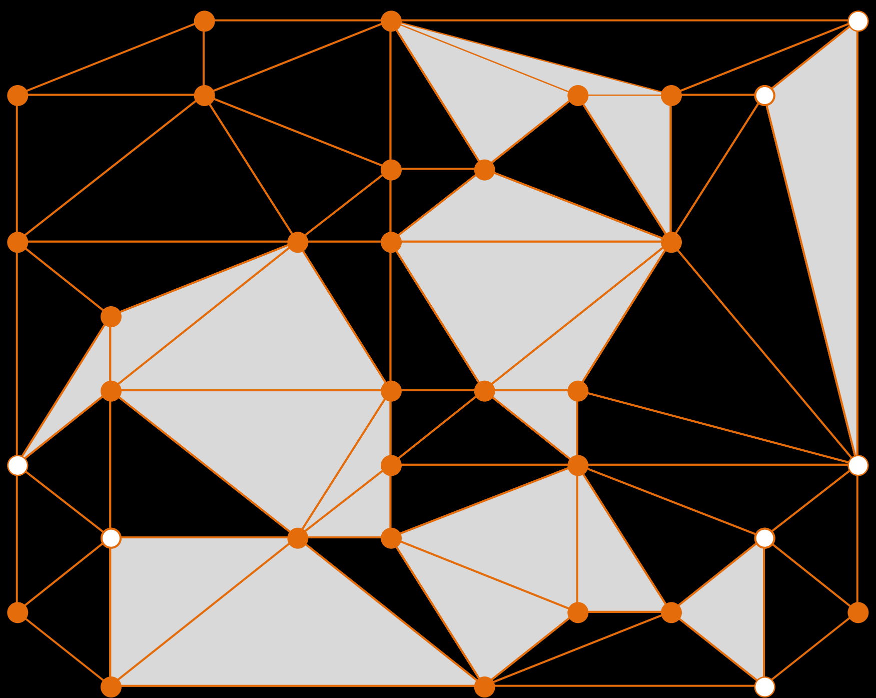


Iterative decoding

Message Passing Example: 1

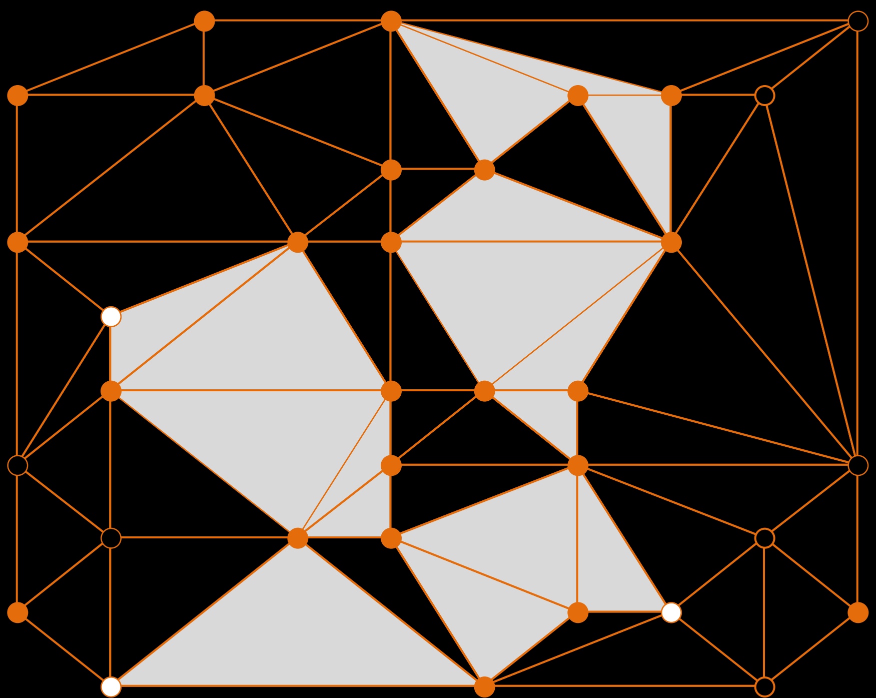


Message Passing Example: 1

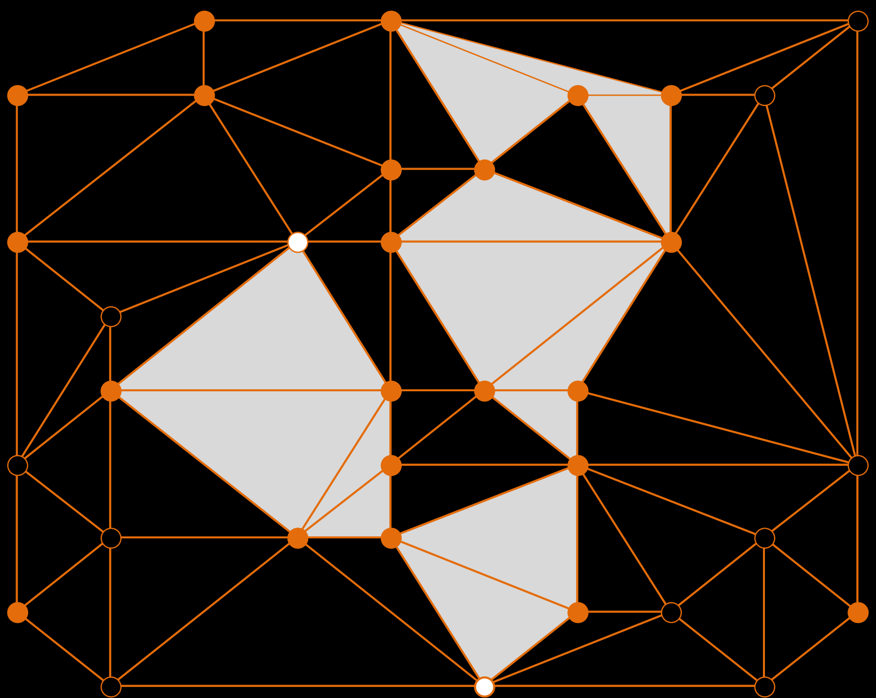


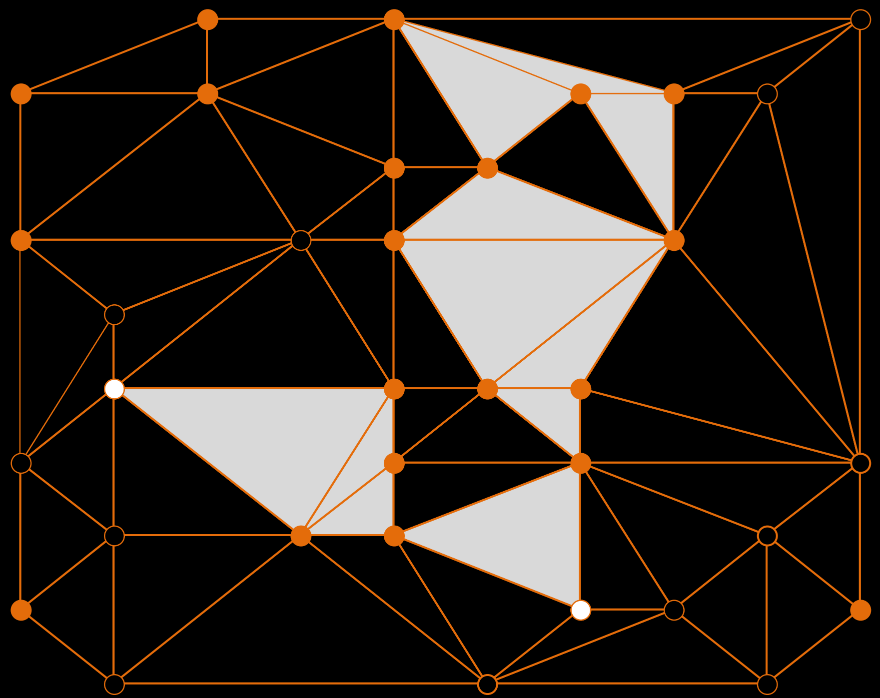
THE UNIVERSITY OF ARIZONA
TUCSON ARIZONA

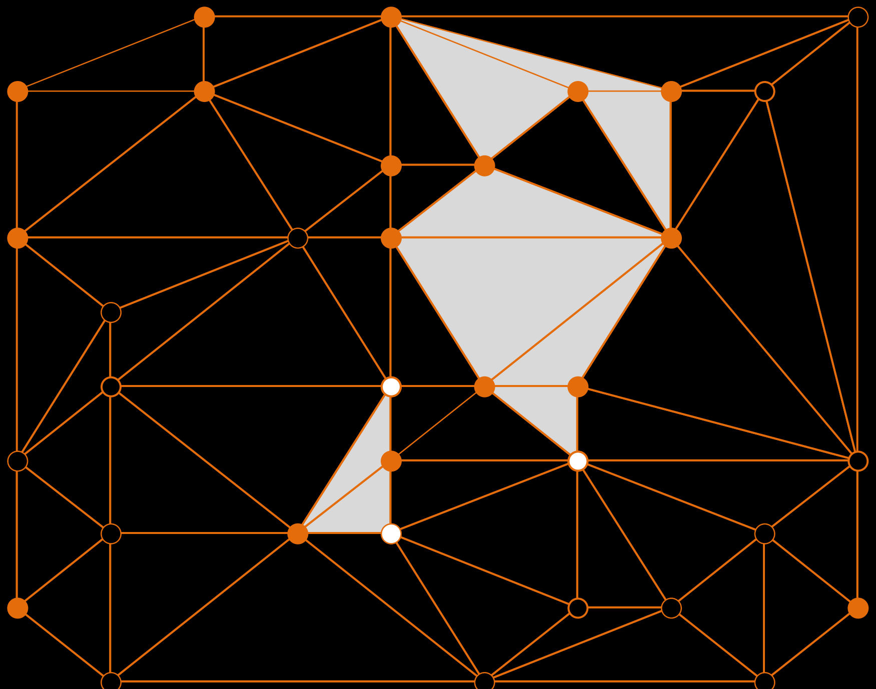
2



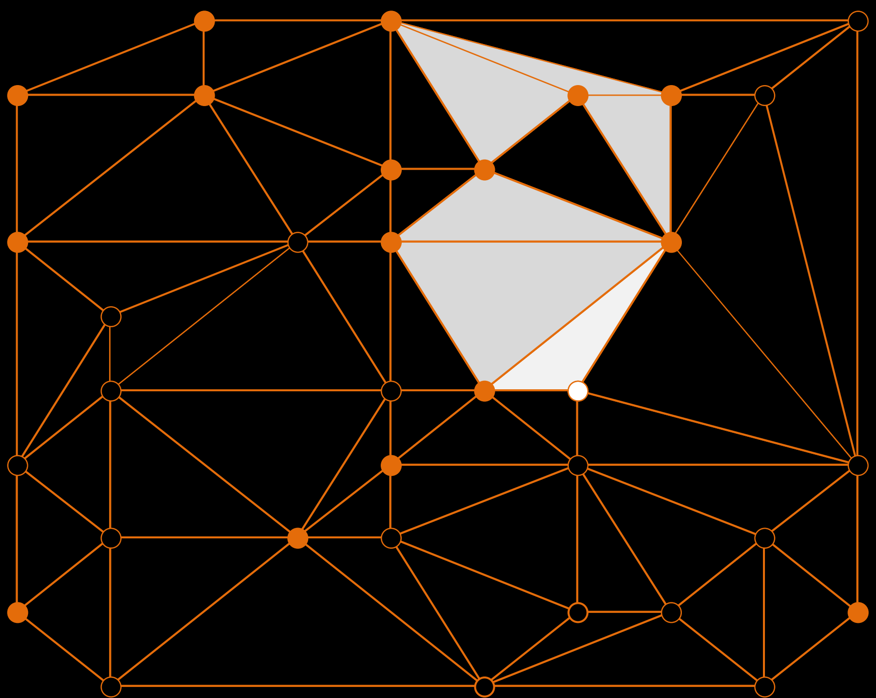
3



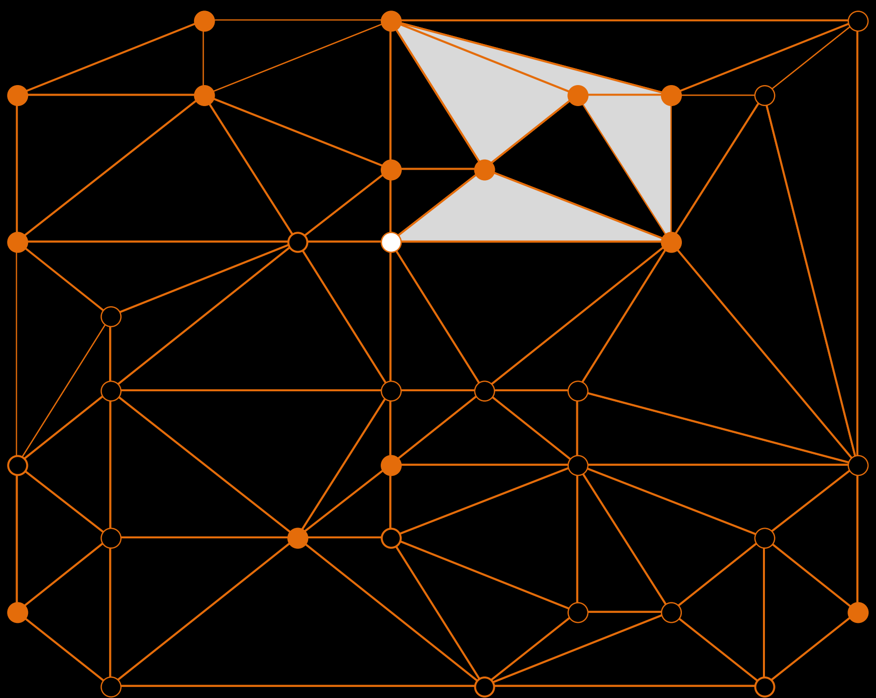




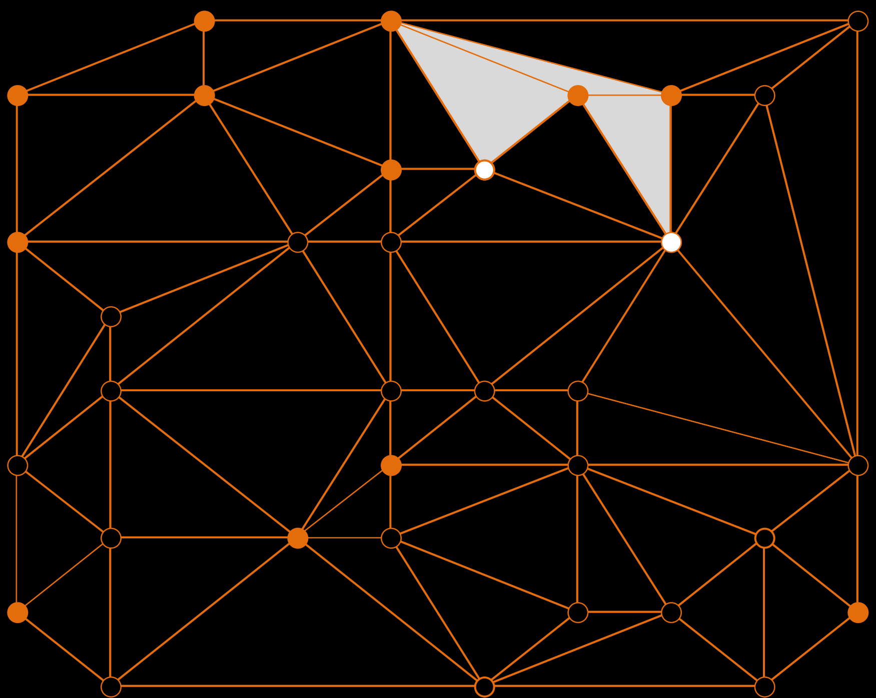
6



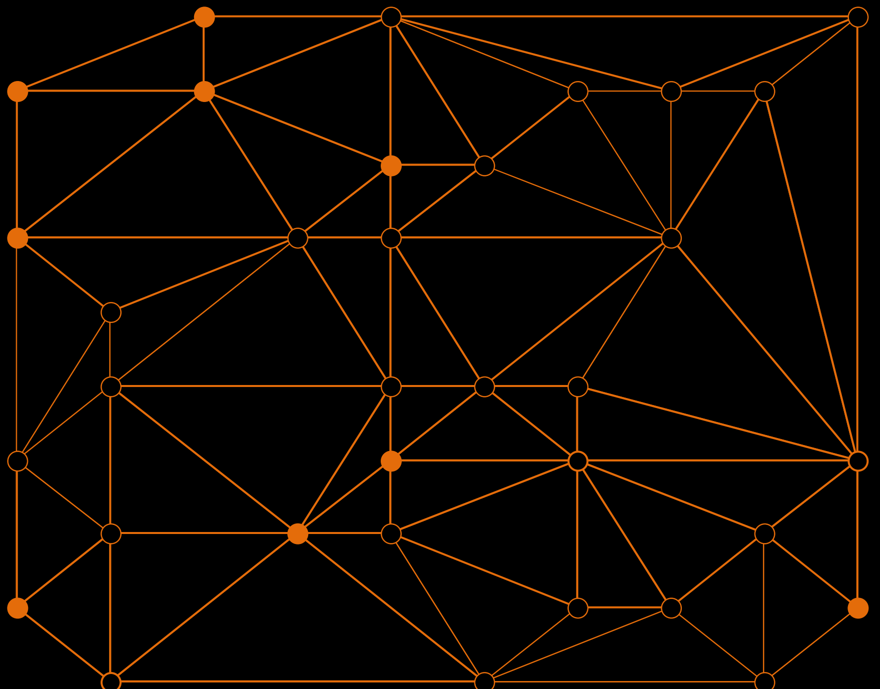
7



8

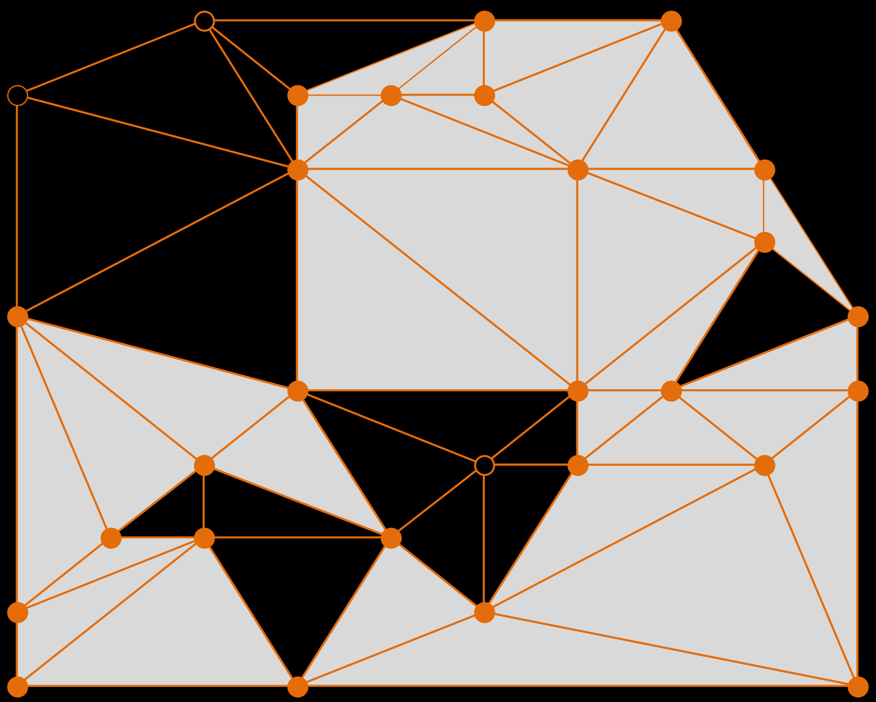


9



Done !

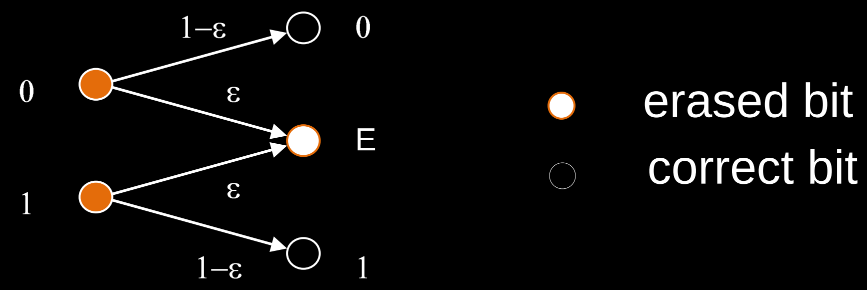
An unresolvable configuration

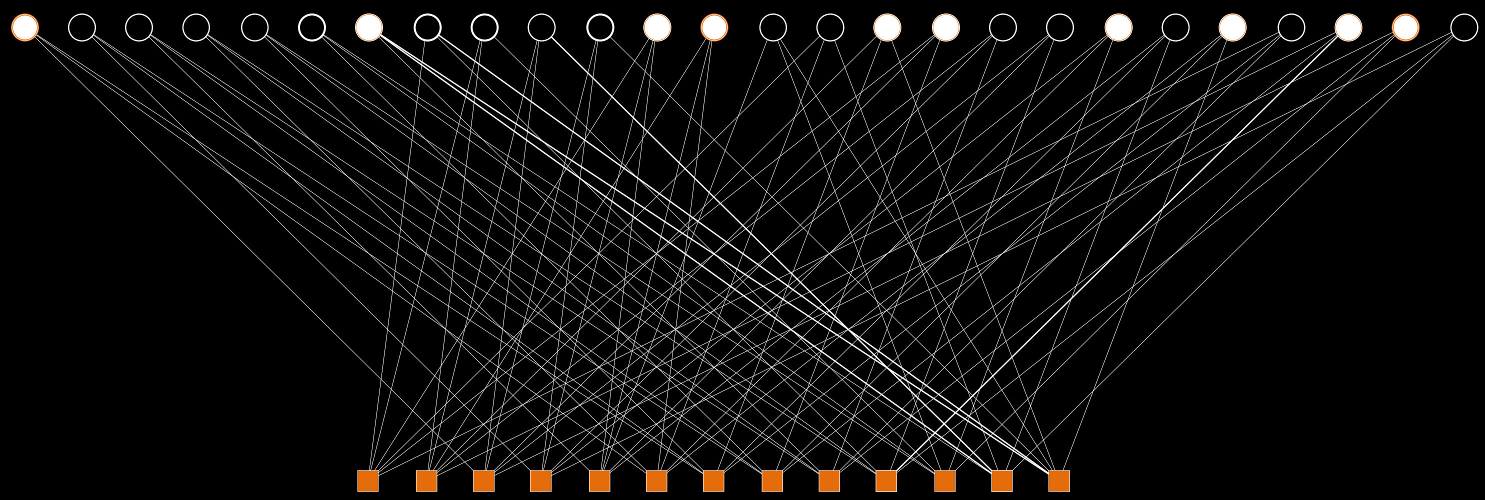


Stucked !

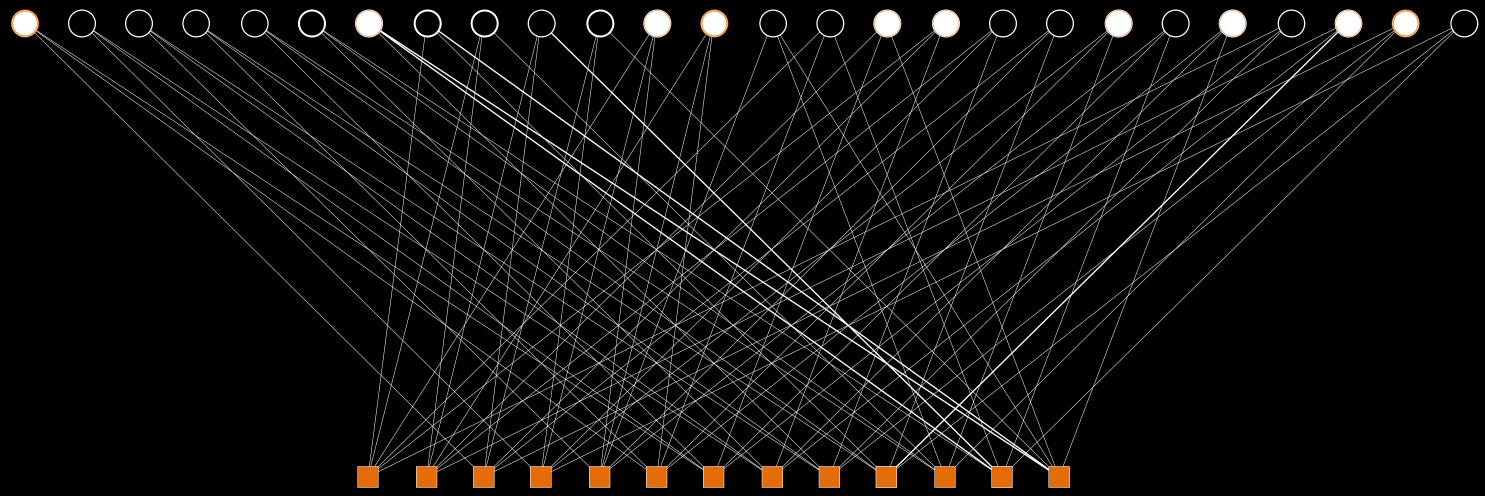
Iterative decoders for BEC

Iterative decoding on BEC

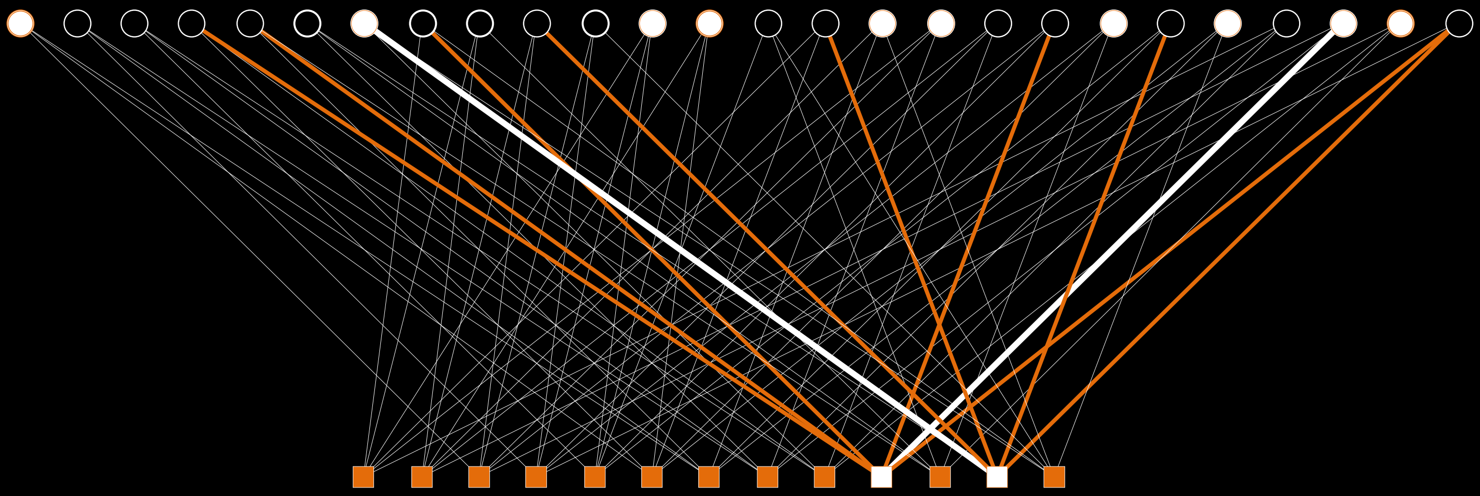




Decoding simulation

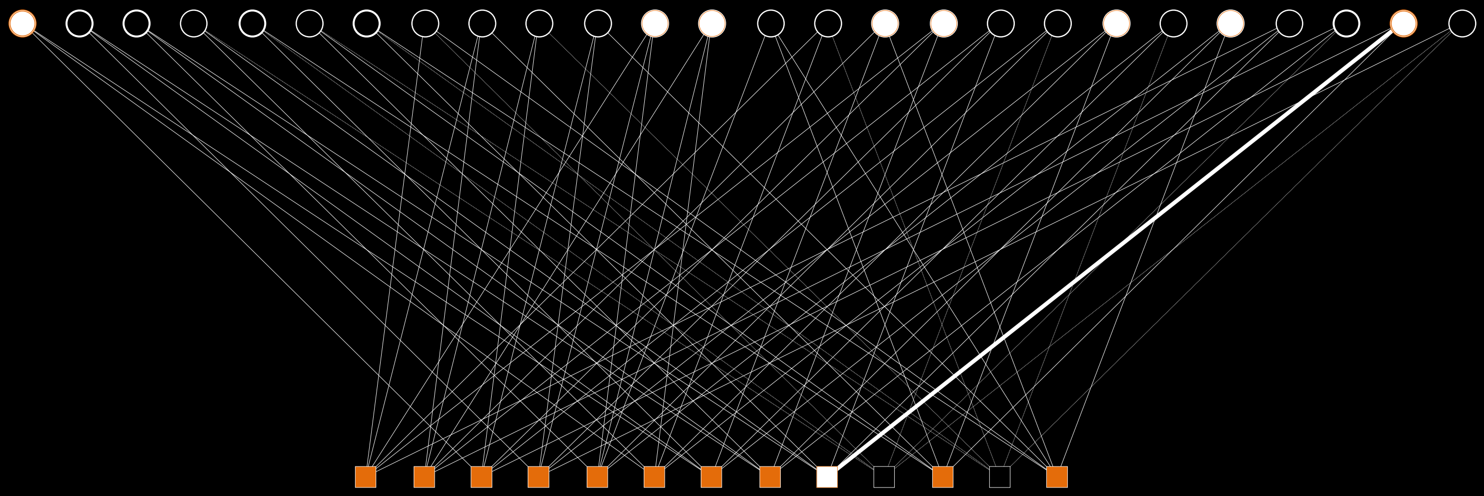


BEC decoding simulation



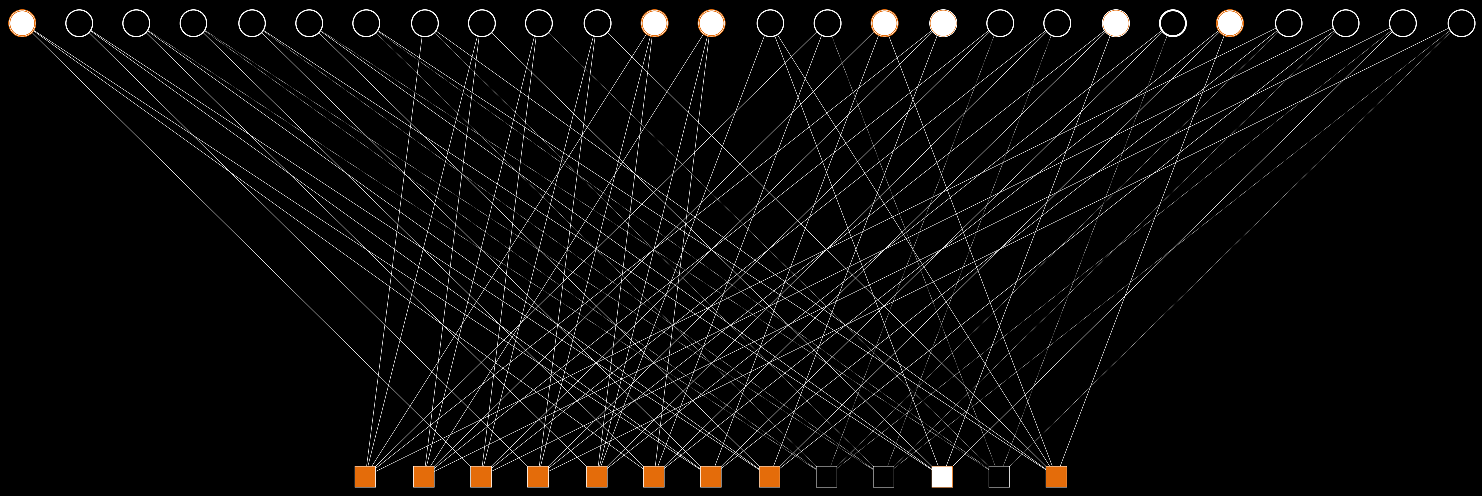
- a check involving a <u>single</u> erased bit
- other check

BEC simulation - 1

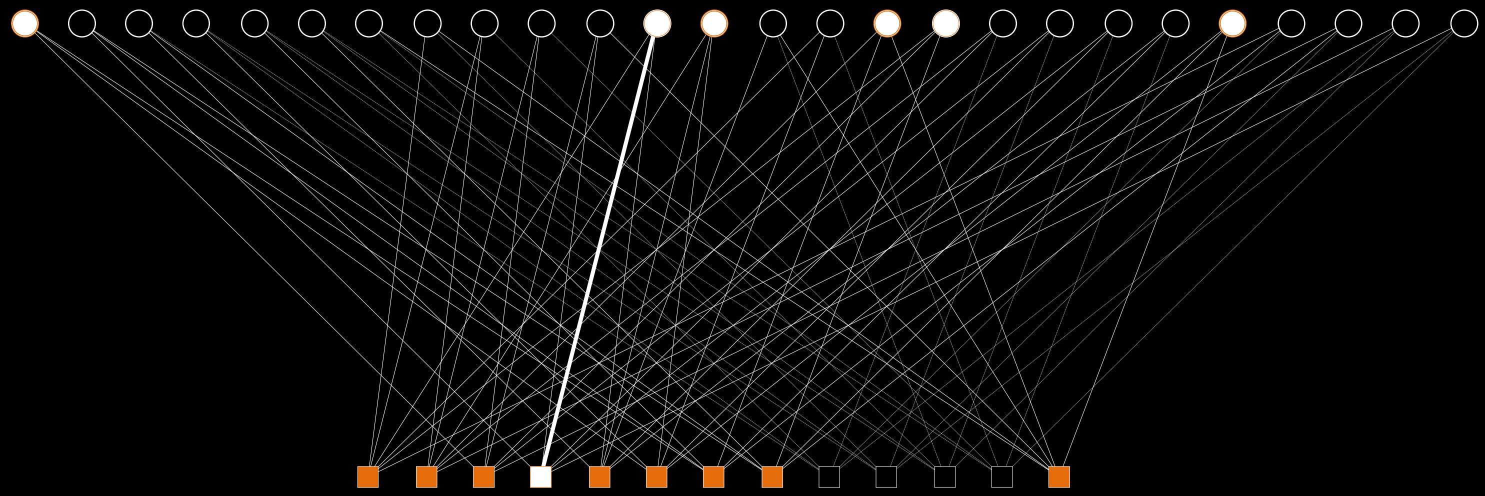


- a check satisfied after correction

BEC simulation - 2

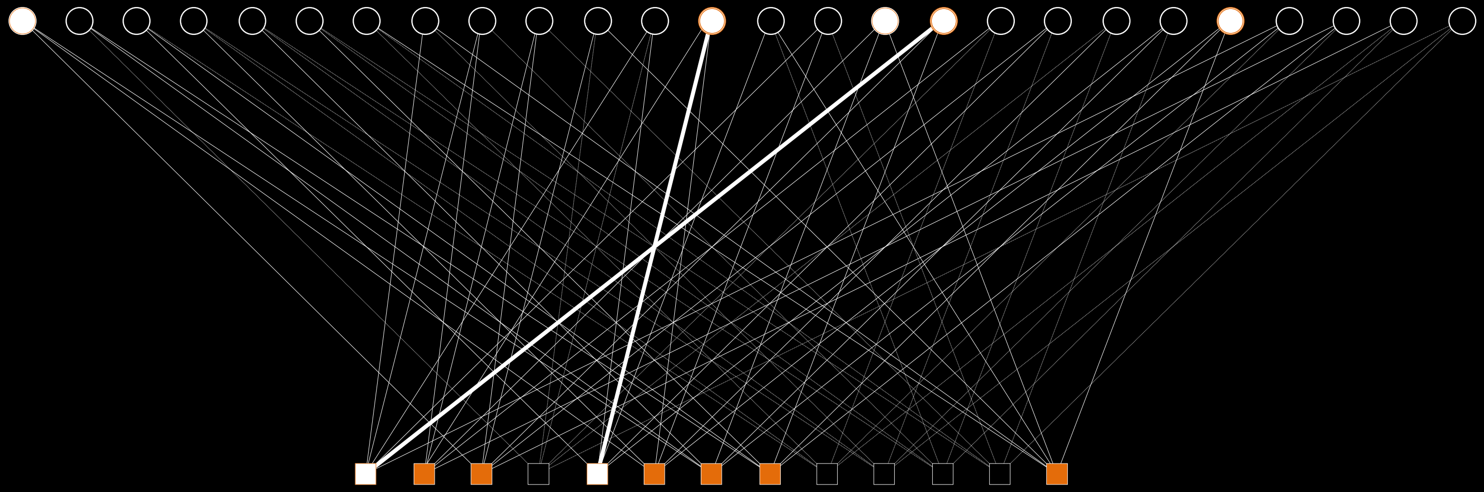


BEC simulation - 3

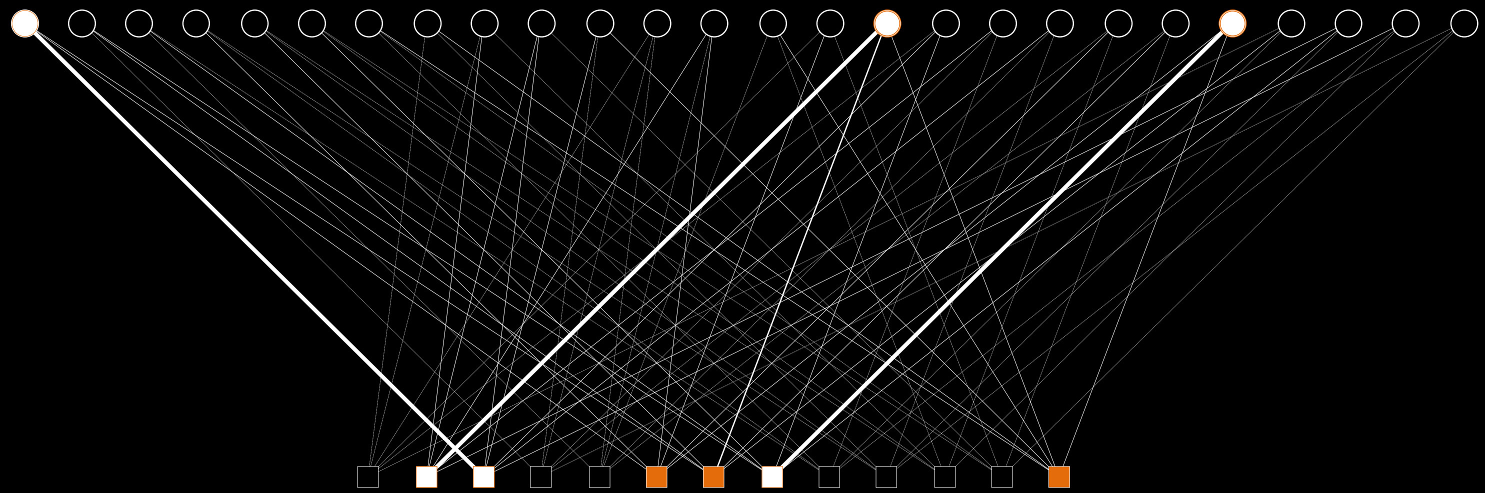


THE UNIVERSITY OF ARIZONA
TUCSON ARIZONA

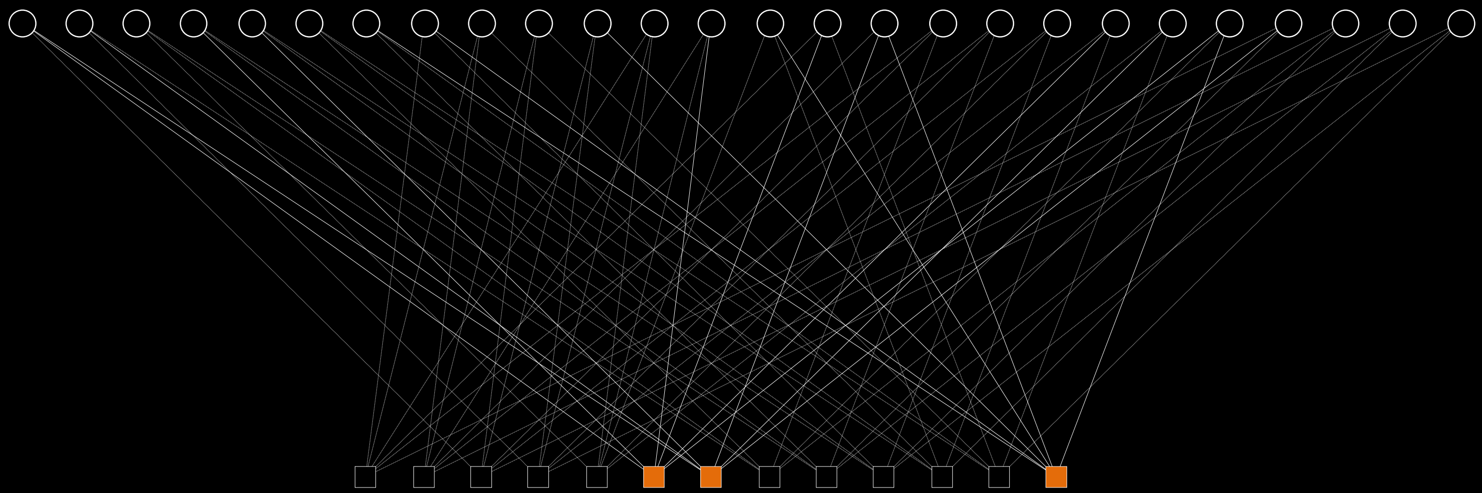
BEC simulation - 4



BEC simulation - 5



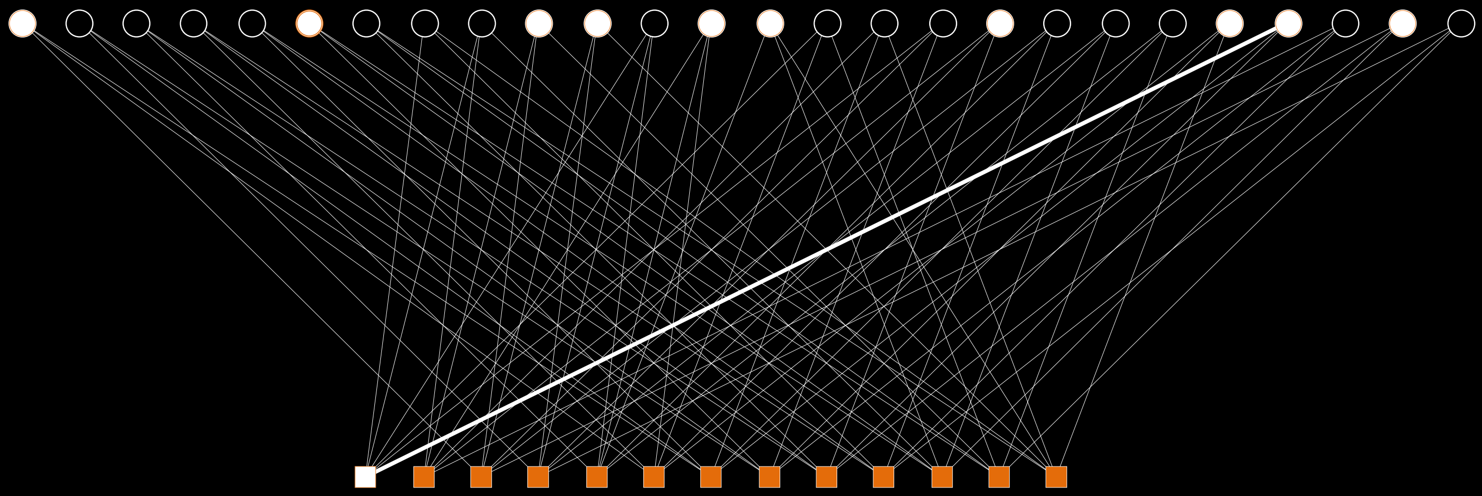
BEC simulation - 6



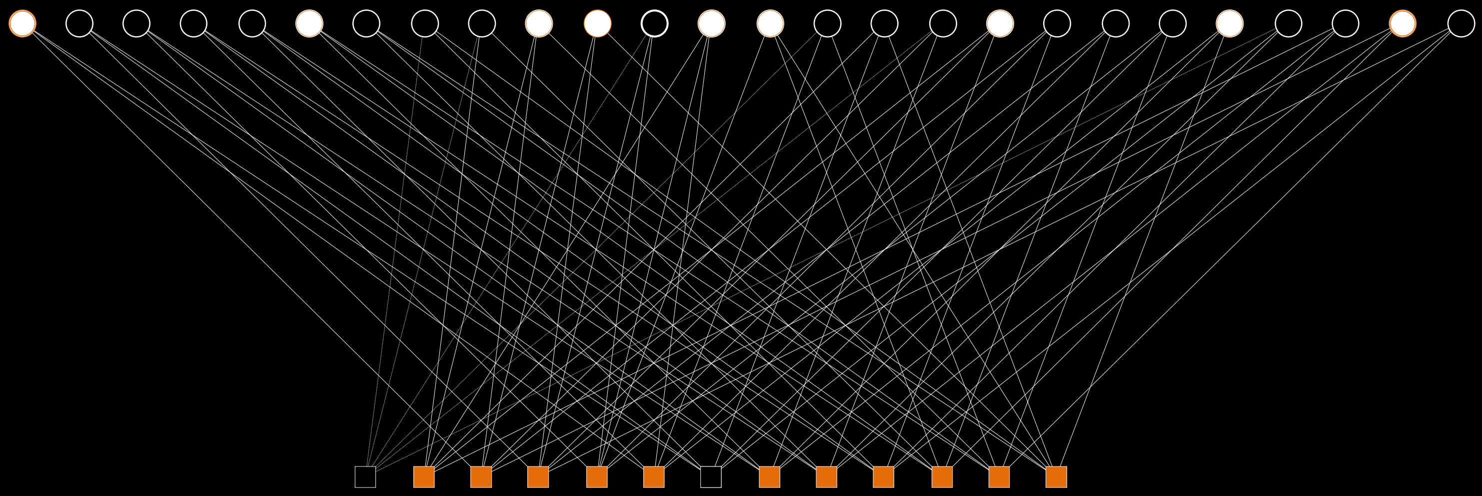
Success !

THE UNIVERSITY OF ARIZONA®
TUCSON ARIZONA

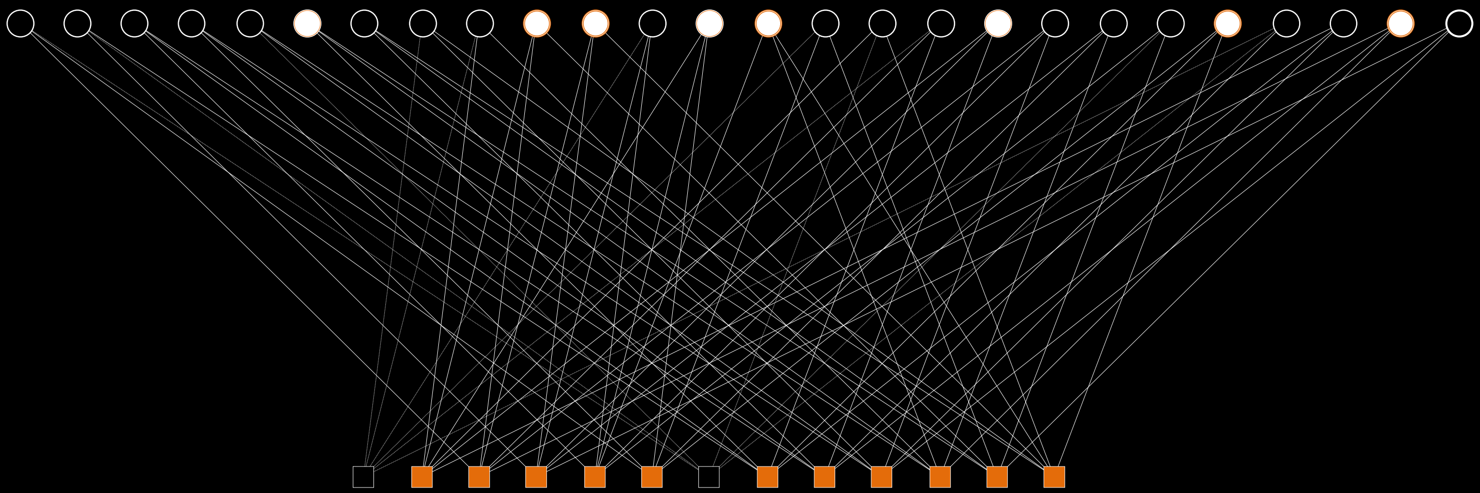
Another example BEC simulation - 1



Another example BEC simulation - 2



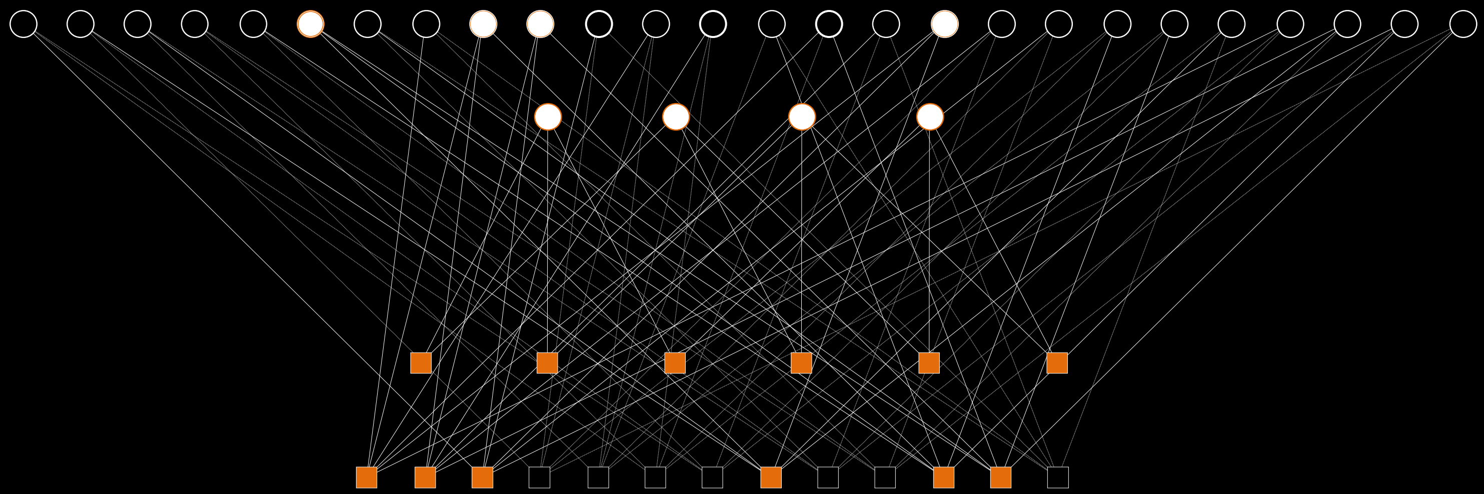
BEC simulation -final



Stuck !

Decoding failures

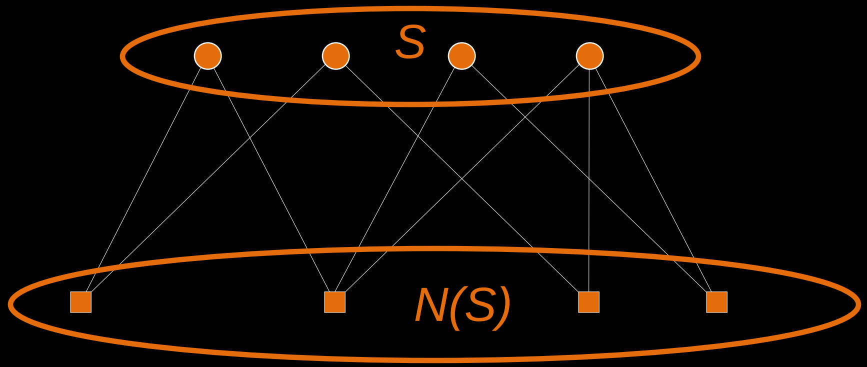
- A BEC iterative decoder fails to converge to a codeword (correct or wrong) if at any iteration there is no check node connected to less than one erased variable node.



- A graph induced by such set of check nodes is called a <u>stopping set</u>.

Combinatorial definition of a stopping set

- Consider a set *S* of variable nodes.
- Let *N(S)* be a set of all checks nodes connected to *S*.
- If smallest outdegree of nodes in *N(S)* is two, then *S* is a stopping set.



- Other channels such as BSC, AWGN do not have such combinatorial definition of a decoding failure.

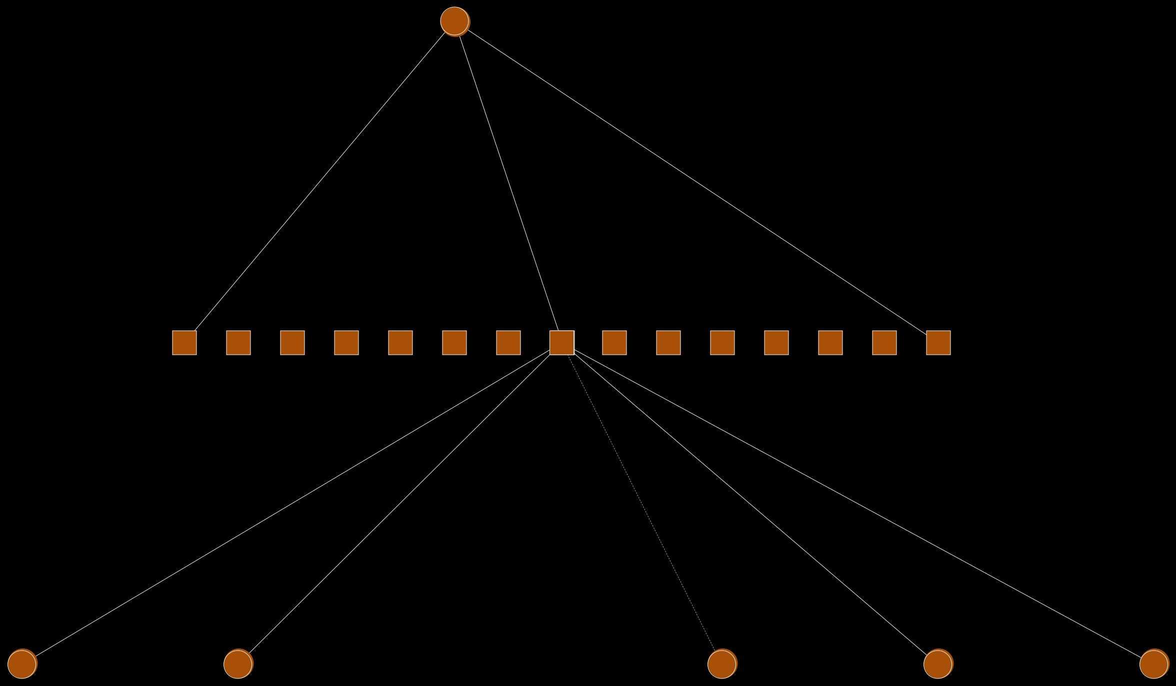
Iterative decoders for BSC

Decoding on graphs on BSC

- Two basic types of algorithms:
  - Bit flipping
  - Message passing

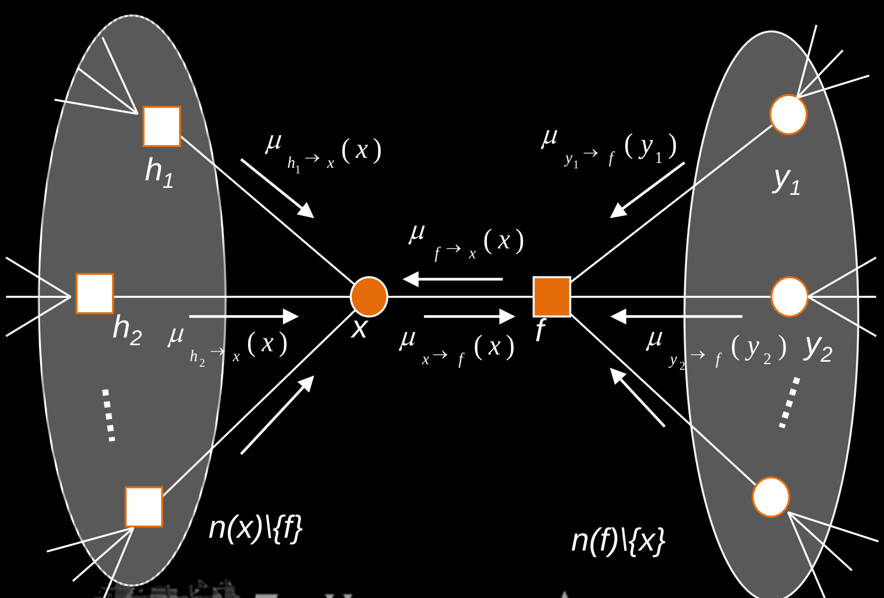
Bit flipping

- If more checks are unsatisfied than satisfied, flip the bit.
- Continue until all checks are satisfied



Message passing

- Steps:
  - A variable node sends his value to all neighboring checks.
  - A check computes XOR of all incoming messages and sends this along the edges, but it excludes the message on the edge the result is send along!
  - Variable takes a majority vote of incoming messages and sends this along, if tie, sends its original value



Gallager A/B algorithm

- The Gallager A/B algorithms are hard decision decoding algorithms in which all the messages are binary.
- With a slight abuse of the notation, let $|\varpi_{*\to i} = m|$ denote the number of incoming messages to i which are equal to $m \in \{0,1\}$. Associated with every decoding round k and variable degree d_i is a threshold b_{k,d_i}.
- The Gallager B algorithm is defined as follows.

$$\begin{aligned}
\omega_{i\to\alpha}^{(0)} &= y_i \\
\varpi_{\alpha\to i}^{(k)} &= \left(\sum_{j\in\mathcal{N}(\alpha)\setminus i} \omega_{j\to\alpha}^{(k-1)}\right) \bmod 2 \\
\omega_{i\to\alpha}^{(k)} &= \begin{cases} 1, & \text{if } |\varpi_{*\setminus\alpha\to i}^{(k)} = 1| \geq b_{k,d_i} \\ 0, & \text{if } |\varpi_{*\setminus\alpha\to i}^{(k)} = 0| \geq b_{k,d_i} \\ y_i, & \text{otherwise} \end{cases}
\end{aligned}$$

Gallager A/B algorithm

- The Gallager A algorithm is a special case of the Gallager B algorithm with $b_{k,d_i} = d_i - 1$ for all k .
- At the end of each iteration, a decision on the value of each variable node is made based on all the incoming messages and possibly the received value.

General iterative decoders

- An iterative decode D is defined as a 4-tuple given by

$$\mathrm{D} = (\mathcal{M}, \mathcal{Y}, \Phi_v, \Phi_c)$$

- $\mathcal{M}$ is a set the message values are confined to
- $\mathcal{Y}$ is the set of channel values
- The function $\Phi_c : \mathcal{M}^{d_c - 1} \to \mathcal{M}$ used for update at a check node with degree d_c .
- The function $\Phi_v : \mathcal{Y} \times \mathcal{M}^{d_v - 1} \to \mathcal{M}$ is the update function used at a variable node with degree d_v.

Decoders as dynamical systems

- Let $\mathbf{v}^{(k)}$ be the vector of messages along all edges in the Tanner graph in the k-th iteration, and $\mathbf{y}$ the received vector, then an iterative decoder D on the Tanner graph G can be seen as a dynamical system

$$\mathbf{v}^{(k)} = F(\mathbf{v}^{(k-1)}, \mathbf{y})$$

- Such dynamical system may have a chaotic behavior
- When alphabets are finite, a decoder is a finite state machine, with a very large state space.
- The trajectory $\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \mathbf{v}^{(2)} \ldots$ converge either to a fixed point or exhibits oscillations around attractor points in the state space.
- The attractor structure is defined by G and D.

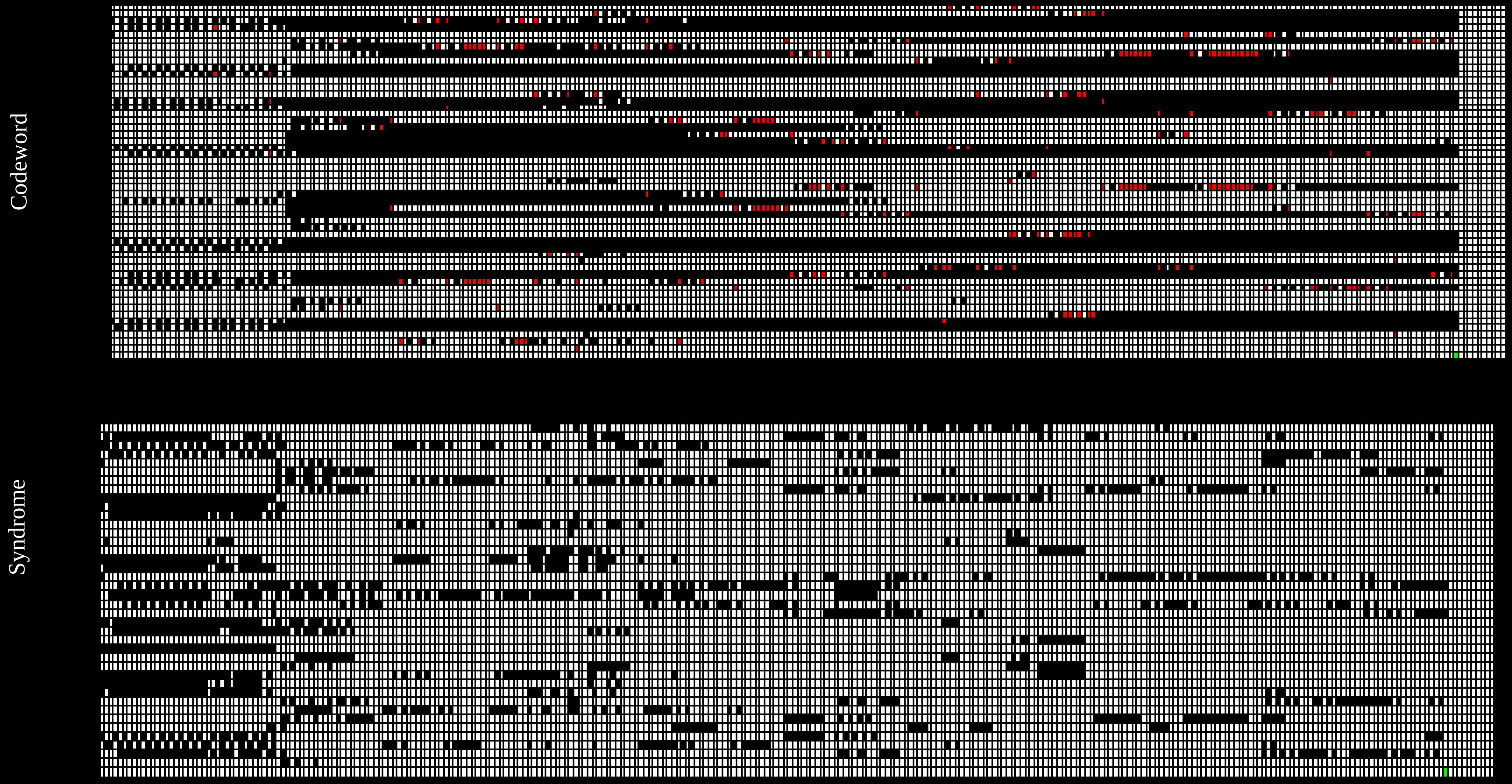
Attractors of iterative decoders

THE UNIVERSITY OF ARIZONA.
TUCSON ARIZONA

adapted using www.sussex.ac.uk and www.metafysica.nl

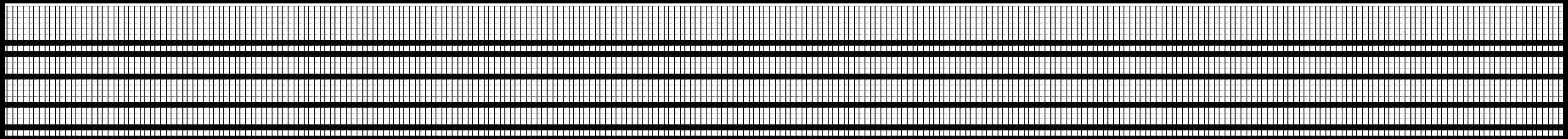
Trajectory examples

- Bit flipping decoder

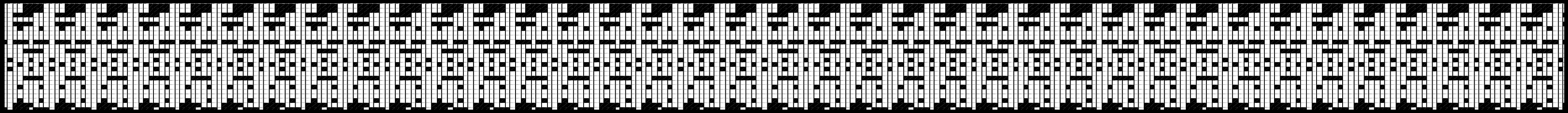


Trajectory types

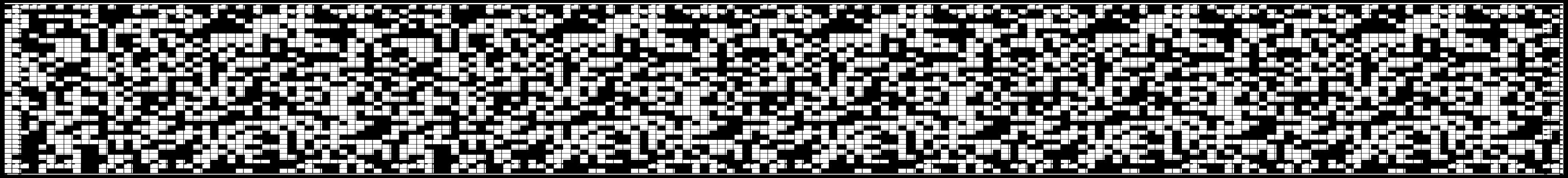
- Fixed point



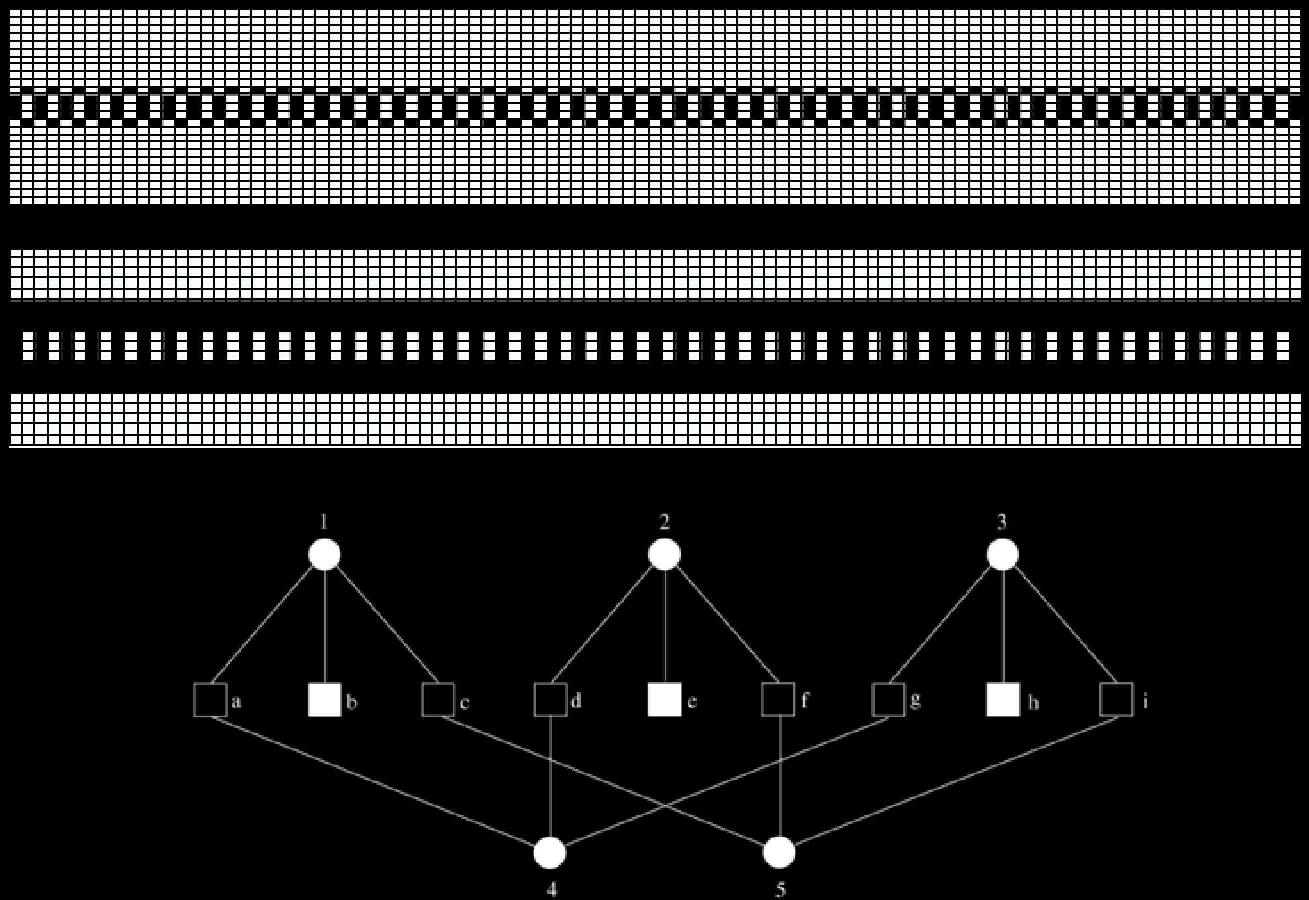
- Cyclic



- Cyclic with a large period

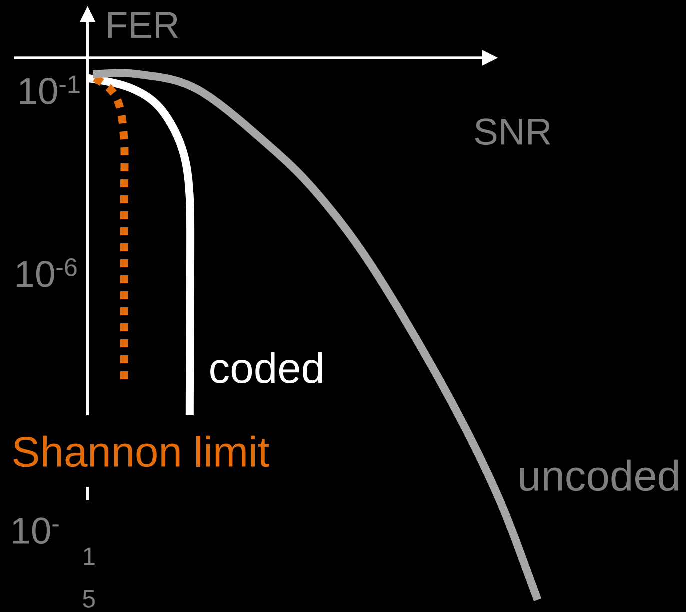


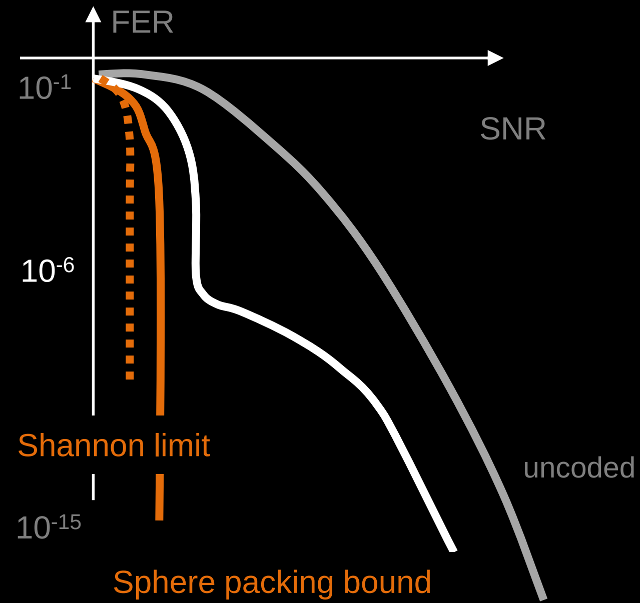
An example of a trajectory



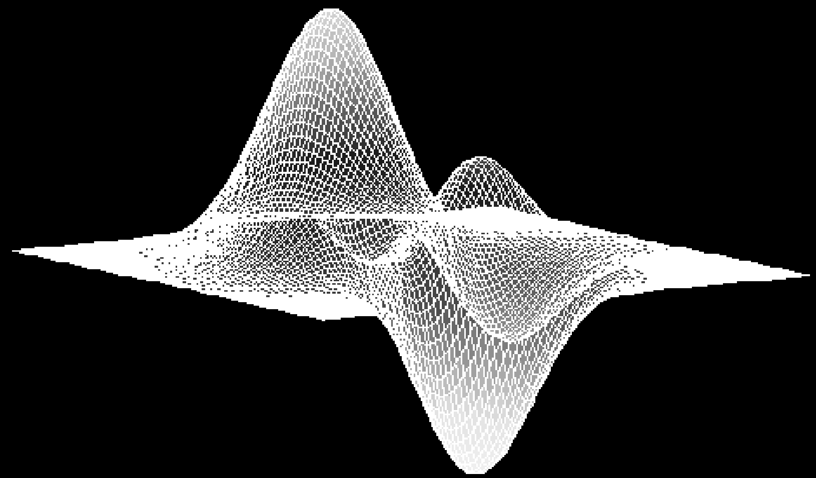
Failures of iterative decoders

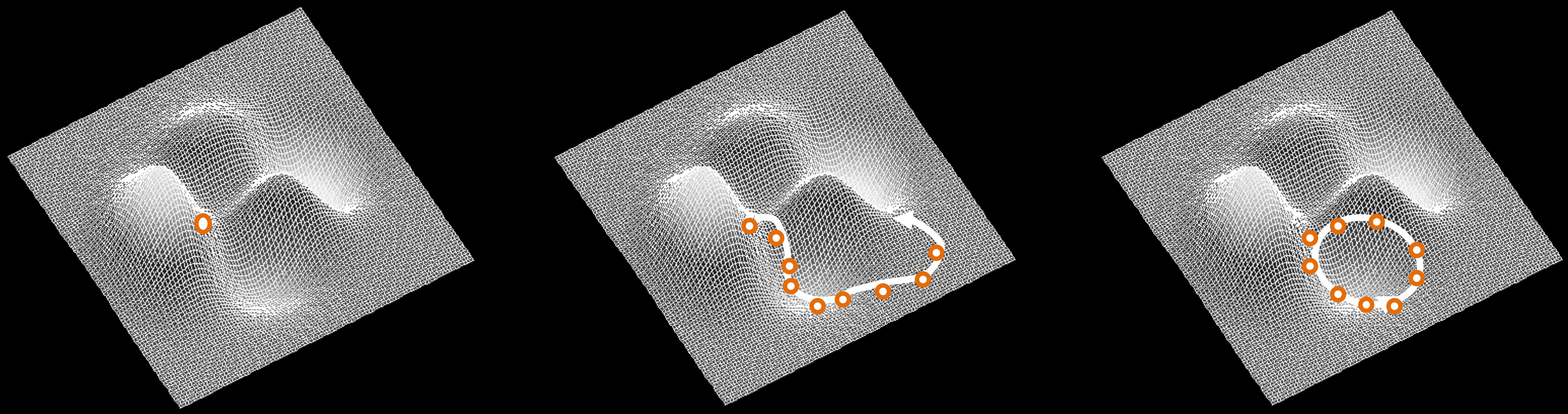
Error floor


FER
SNR
10^{-1}
10^{-6}
coded
Shannon limit
uncoded
10^{-15}


FER
SNR
10^{-1}
10^{-6}
Shannon limit
uncoded
10^{-15}
Sphere packing bound

Locality of decoding

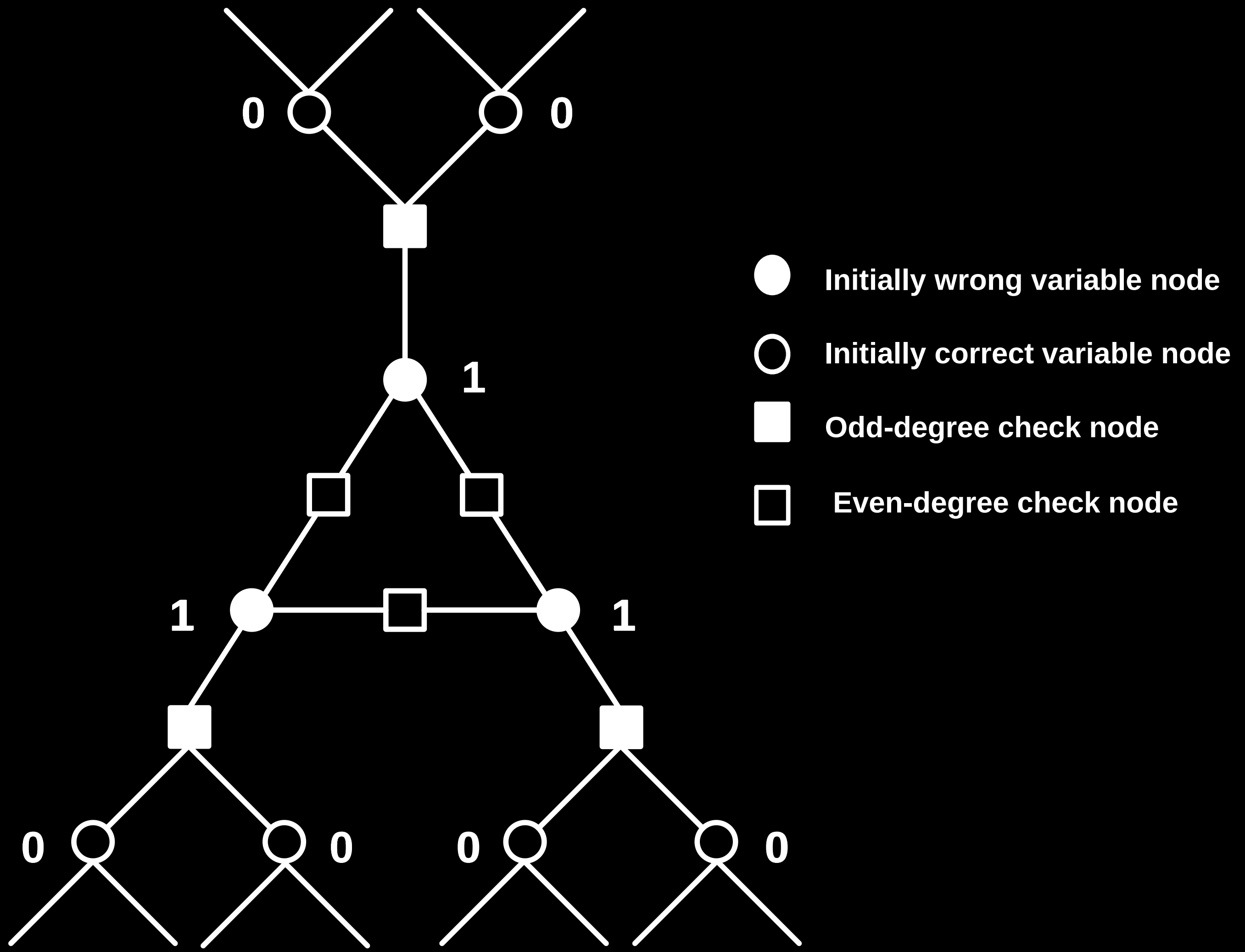




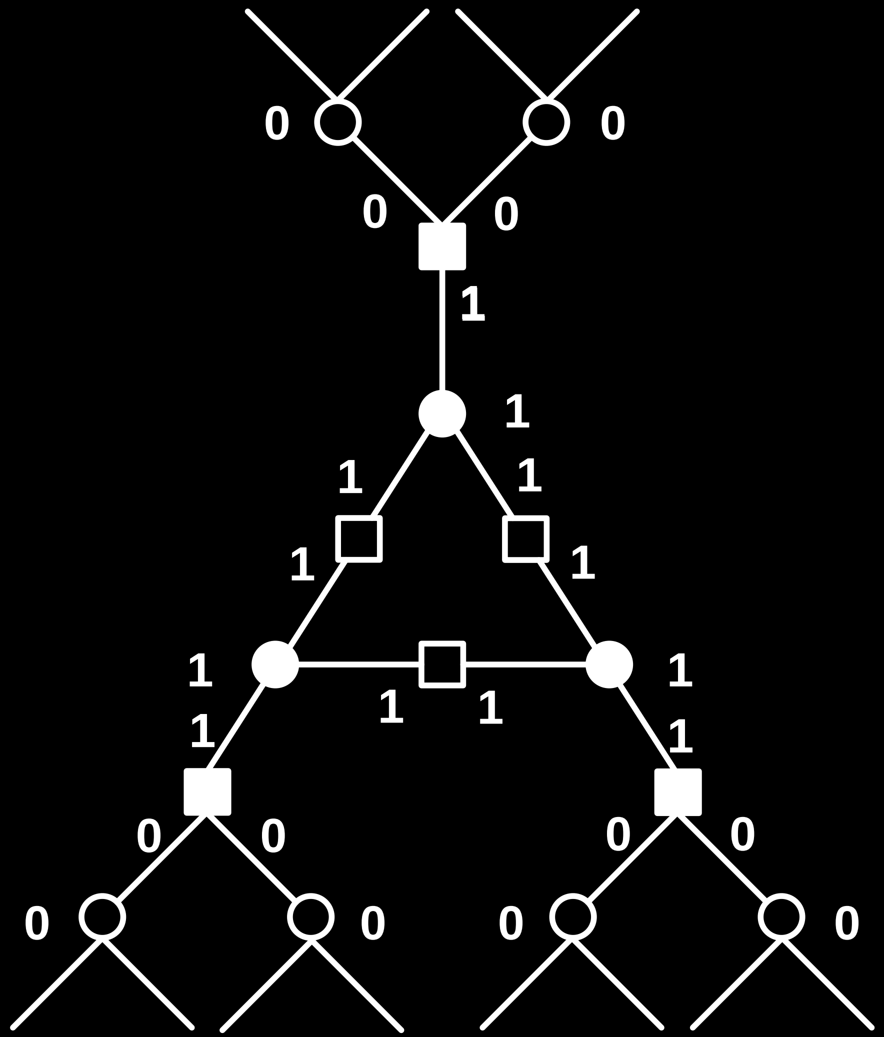
A motivating example

- Consider a six cycle in a 3-variable regular Tanner Graph.
- Assume the channel introduces three errors exactly on the variable nodes in the cycle.
- Also the assume that the neighborhood of the subgraph does not influence the messages propagated within the subgraph (condition to be explained later)
- Gallager – A fails for such error pattern.
- By adding an extra bit in the message, the decoder can succeed.

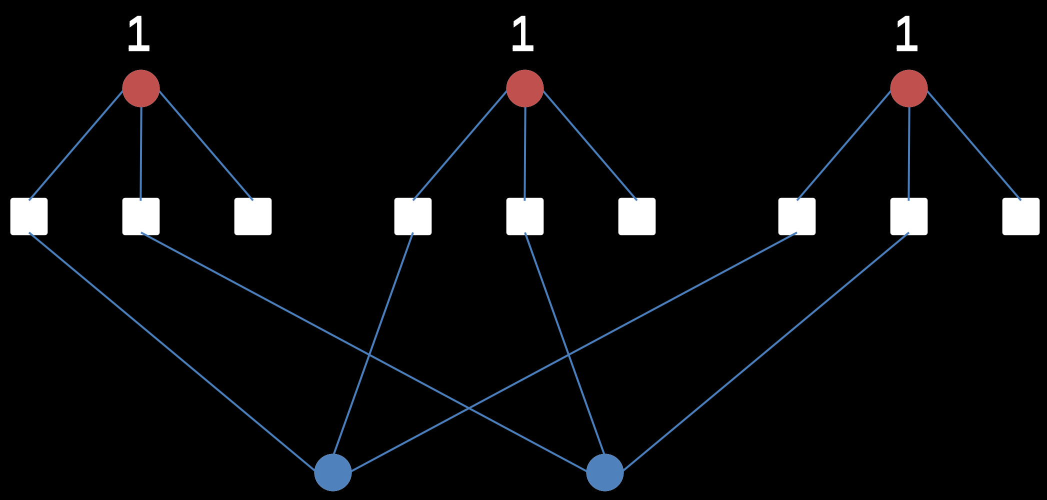
Gallager – A iteration 1



Gallager – A iteration 2

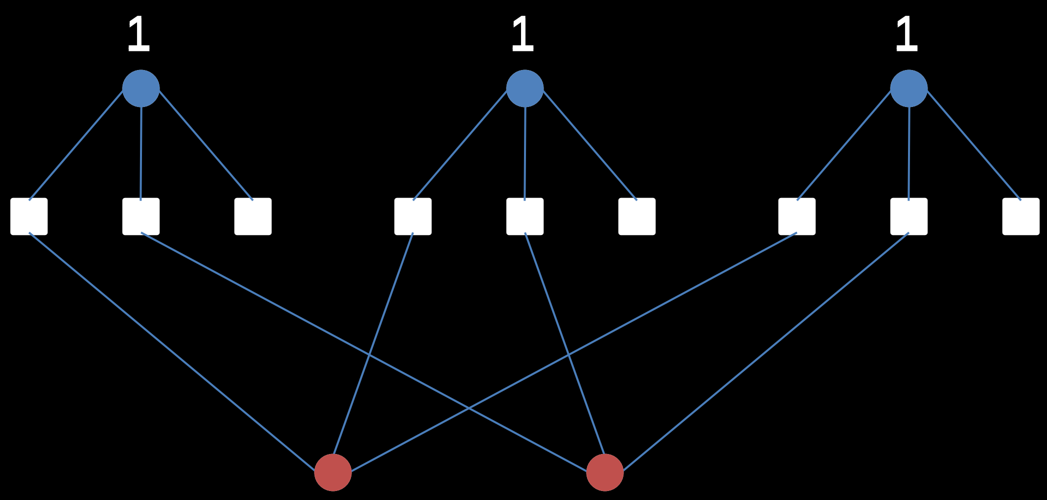


A trapping set illustration



- Corrupt variable
- Correct variable
- Variable decoded correctly
- Variable decoded wrongly

A trapping set illustration



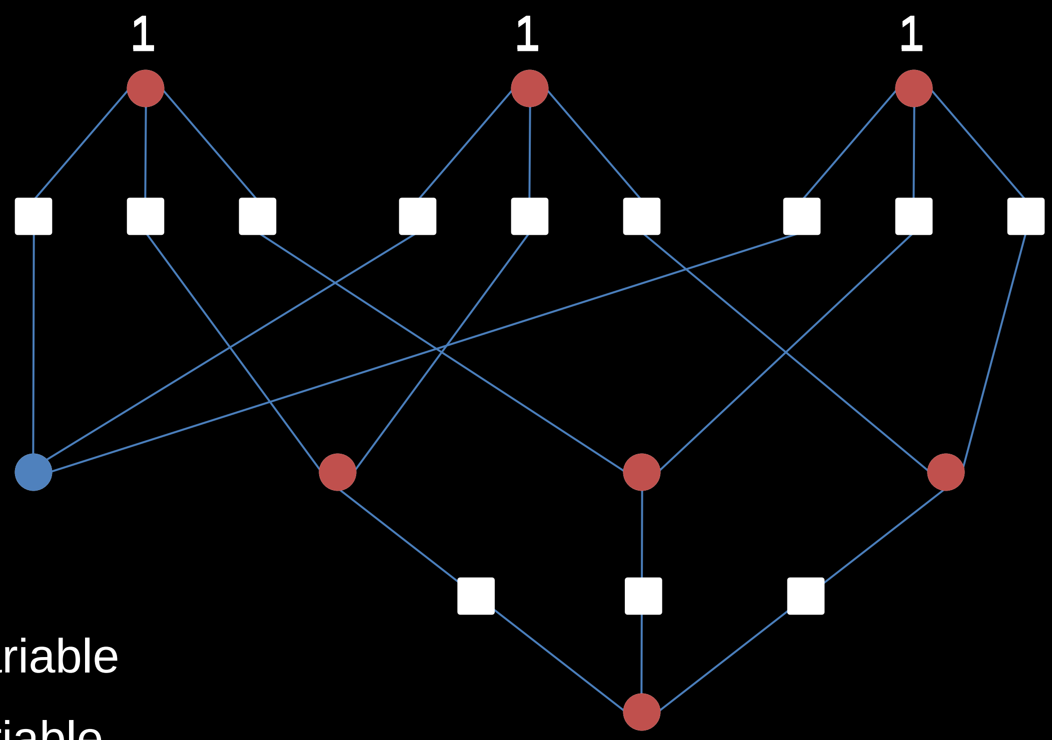
Corrupt variable

Correct variable

Variable decoded correctly

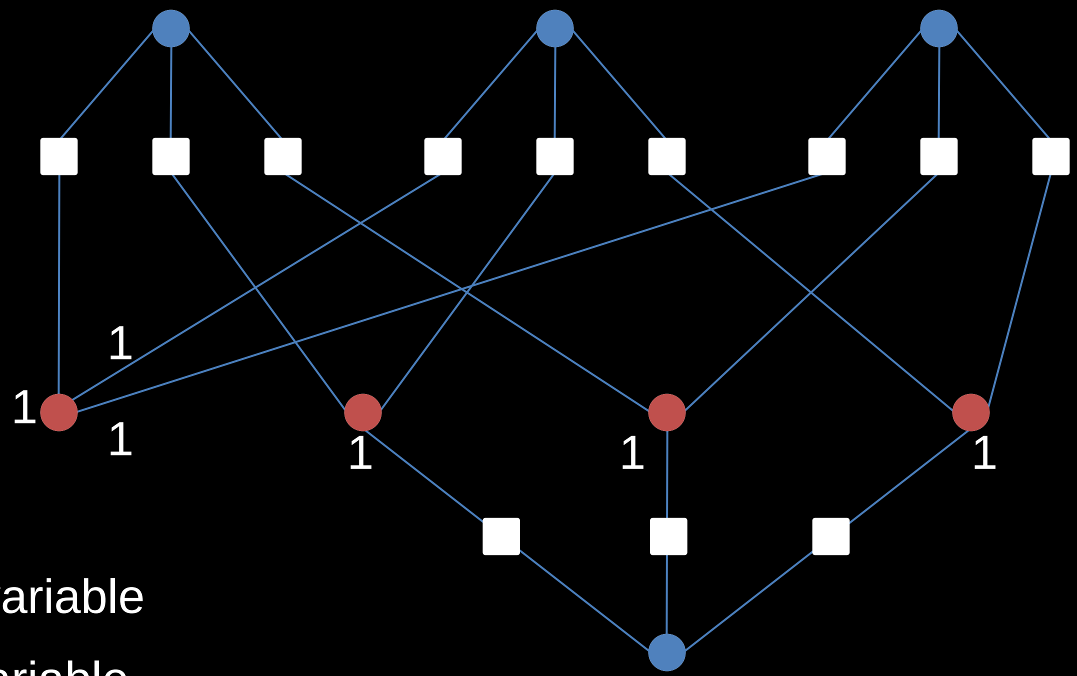
Variable decoded wrongly

Oscillations in the decoder



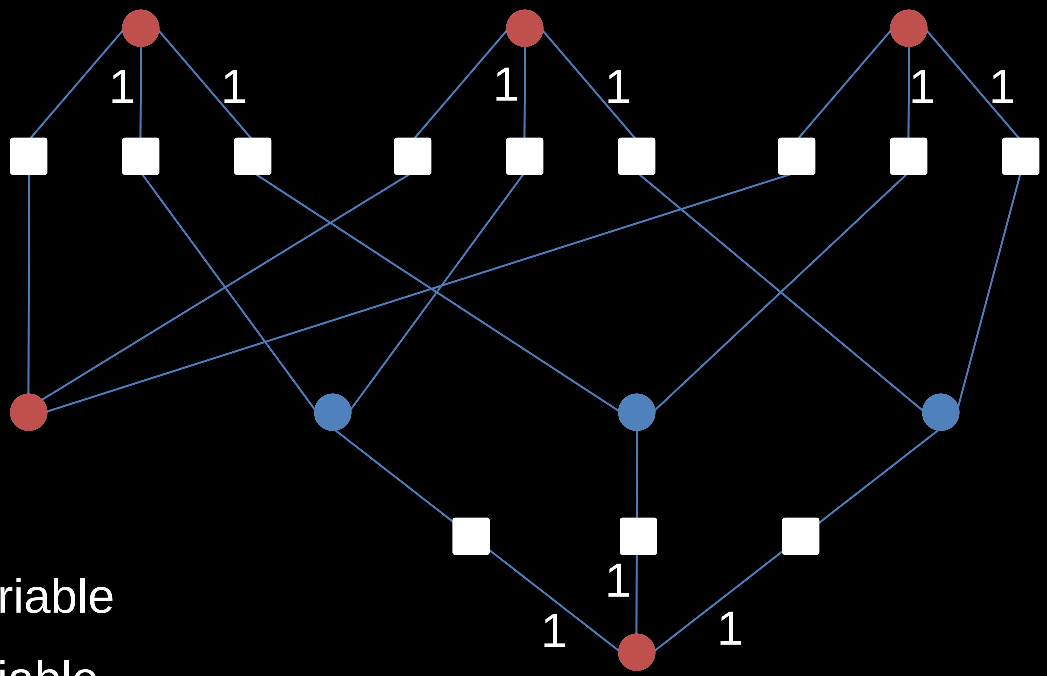
- Corrupt variable
- Correct variable
- Variable decoded correctly
- Variable decoded wrongly

Oscillations in the decoder



- Corrupt variable
- Correct variable
- Variable decoded correctly
- Variable decoded wrongly

Oscillations in the decoder



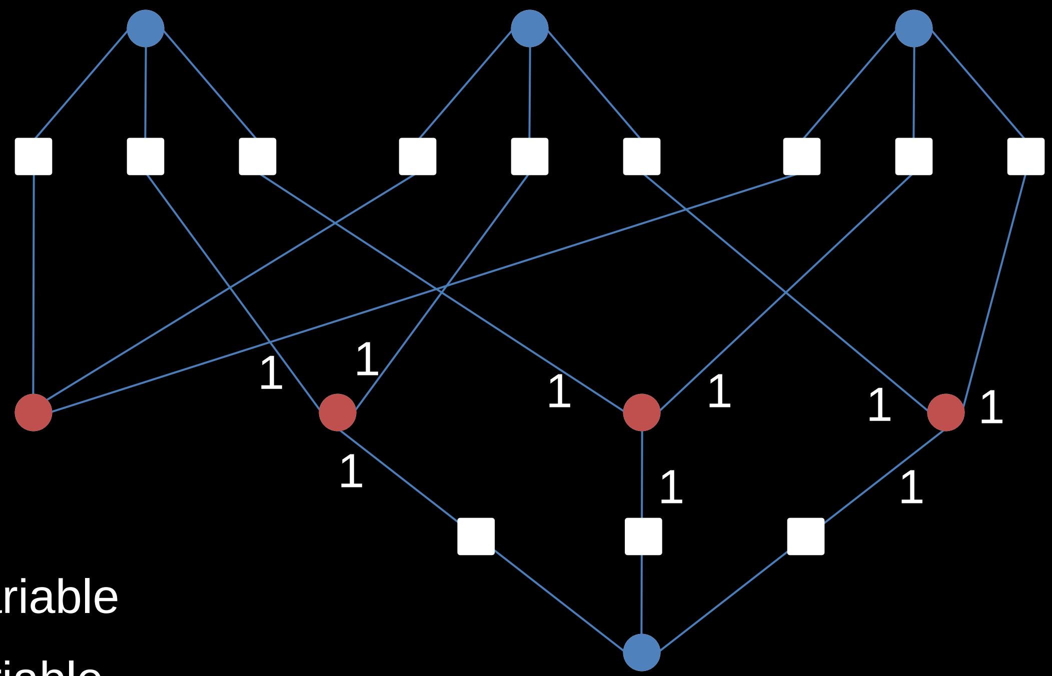
Corrupt variable

Correct variable

Variable decoded correctly

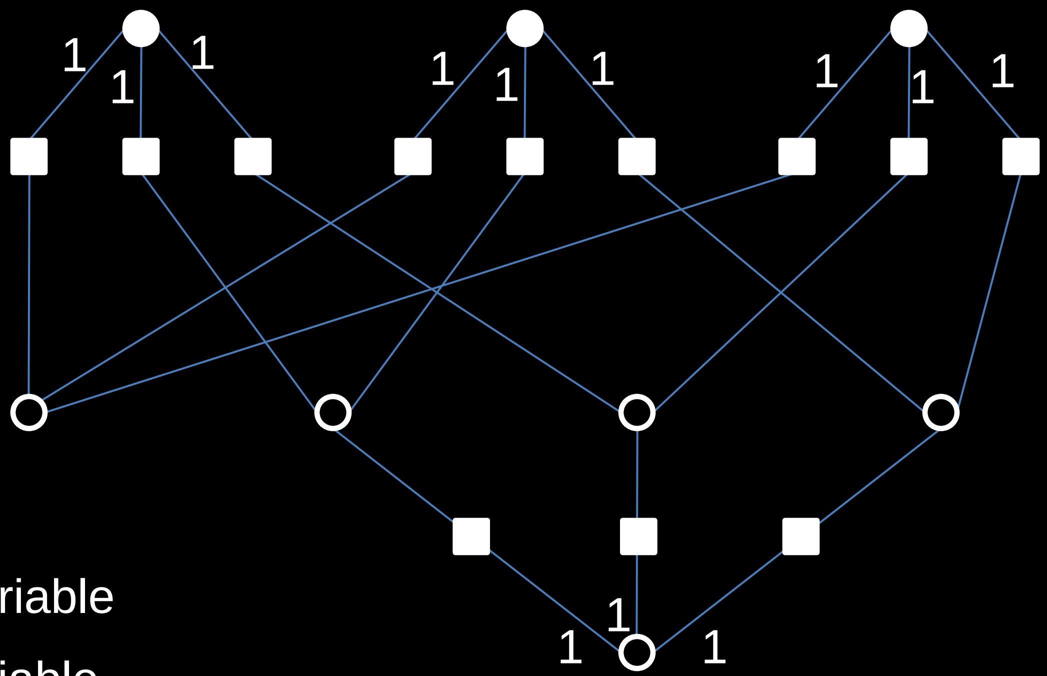
Variable decoded wrongly

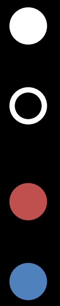
Oscillations in the decoder



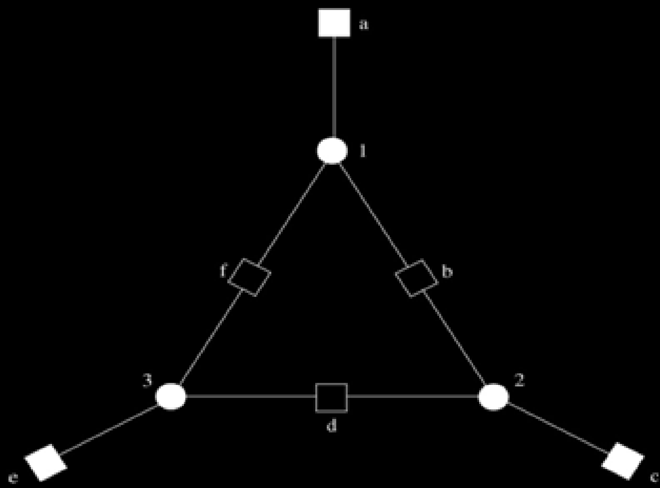
- Corrupt variable
- Correct variable
- Variable decoded correctly
- Variable decoded wrongly

Oscillations in the decoder

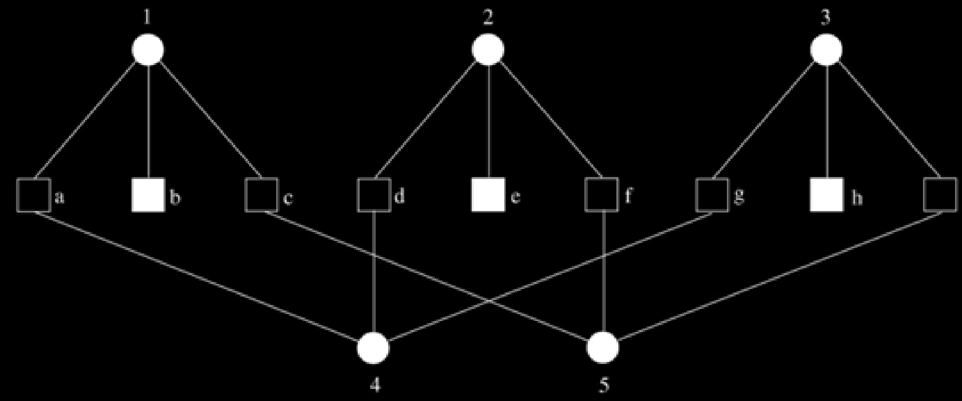




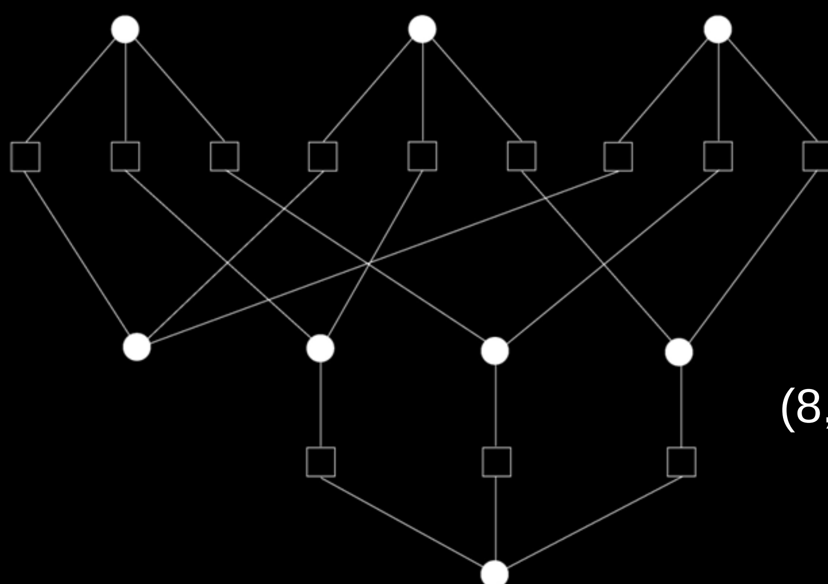
Concept of a trapping set



(3,3) trapping set



(5,3) trapping set



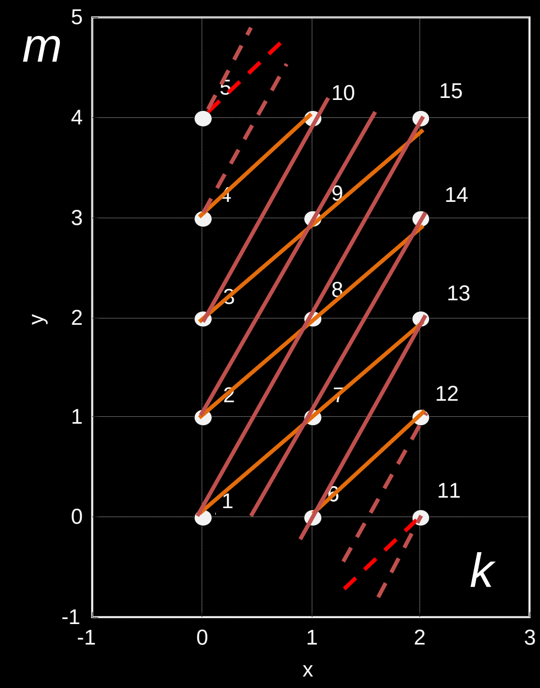
(8,0) Trapping Set

Some ways to construct LDPC codes

LDPC codes - combinatorial designs

- Affine partial geometry $L = \{(x, y) : 0 \leq x \leq k-1,\ 0 \leq y \leq m-1\}$
- *m - a prime*
- Blocks: the lines starting at points (0,*a*) with slopes *s*
  - *(0 ≤a,s ≤m-1)*
  - each point incident with exactly *m* blocks
  - m^2 blocks
- Example: *k=3, m=5*

s=0			s=1			s=2			s=3			s=4		
1	6	11	1	7	13	1	8	15	1	9	12	1	10	14
2	7	12	2	8	14	2	9	11	2	10	13	2	6	15
3	8	13	3	9	15	3	10	12	3	6	14	3	7	11
4	9	14	4	10	11	4	6	13	4	7	15	4	8	12
5	10	15	5	6	12	5	7	14	5	8	11	5	9	13



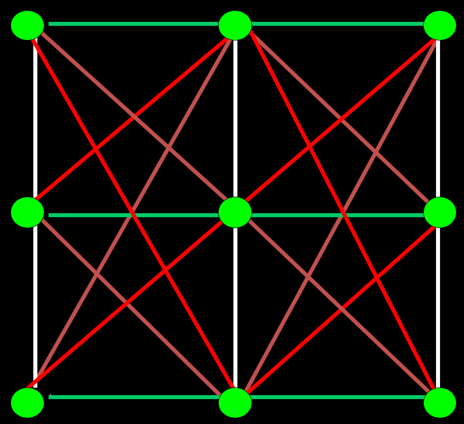
Integer lattice codes

s=0			*s=1*			*s=2*			*s=3*			*s=4*		
1	6	11	1	7	13	1	8	15	1	9	12	1	10	14
2	7	12	2	8	14	2	9	11	2	10	13	2	6	15
3	8	13	3	9	15	3	10	12	3	6	14	3	7	11
4	9	14	4	10	11	4	6	13	4	7	15	4	8	12
5	10	15	5	6	12	5	7	14	5	8	11	5	9	13

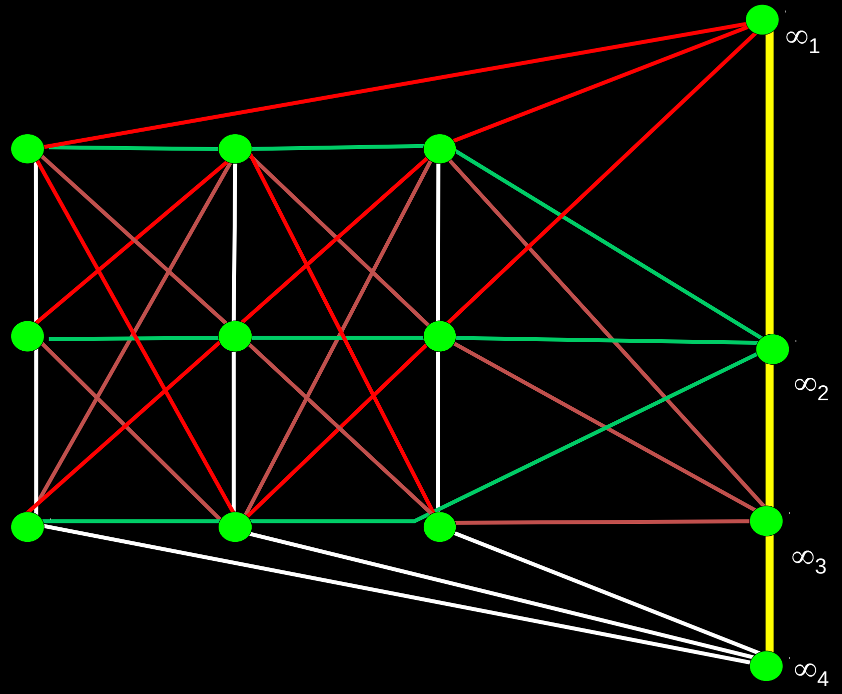
$$H = \left[\begin{array}{ccccc:ccccc:ccccc:ccccc:ccccc}
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
\hdashline
1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0\\
\hdashline
1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0\\
0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&0
\end{array}\right]$$

Affine and projective planes-example

Affine Plane



Projective Plane



Cyclic difference families

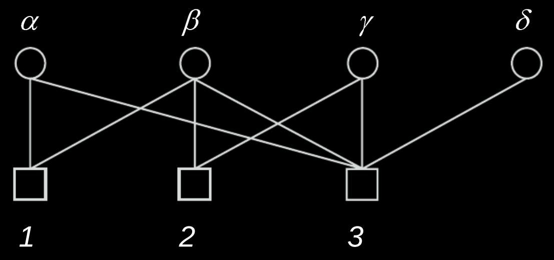
- We can think of the actions of the group V as a partitioning B into classes or orbits.
- Example: (13,3,1) CDF, Z_{13}
- Base blocks B_1={0,1,4} and B_2={0,2,7}

B_1 orbits			B_2 orbits		
$b_{11}+g$	$b_{12}+g$	$b_{13}+g$	$b_{21}+g$	$b_{22}+g$	$b_{23}+g$
0	1	4	0	2	7
1	2	5	1	3	8
2	3	6	2	4	9
3	4	7	3	5	10
4	5	8	4	6	11
5	6	9	5	7	12
6	7	10	6	8	0
7	8	11	7	9	1
8	9	12	8	10	2
9	10	0	9	11	3
10	11	1	10	12	4
11	12	2	11	0	5
12	0	3	12	1	6

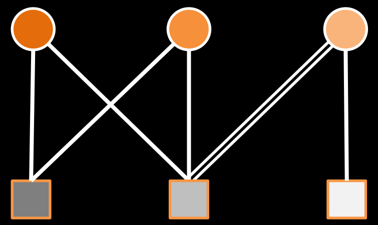
$$H = \left[\begin{array}{ccccccccccccc|ccccccccccccc}
1&0&0&0&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&1&0&0&0&0&1&0\\
1&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1\\
0&1&1&0&0&0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&1&0&0&0&0\\
0&0&1&1&0&0&0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&1&0&0&0\\
1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0\\
0&0&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&1\\
0&0&0&1&0&0&1&1&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&0&1&0&0&1&1&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0\\
0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0\\
0&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&1&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&1&0&0&0&0&1&0&1
\end{array}\right]$$

Protograph based codes

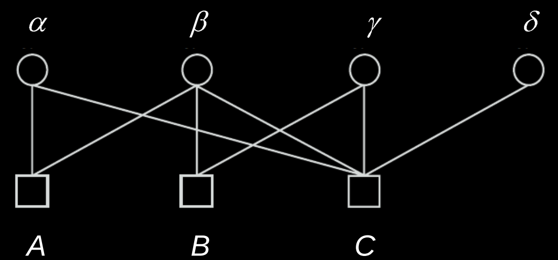
- A protograph is a small Tanner graph.
- Example (Thorpe):
  - $|V|$ = 4 variable nodes and $|C|$ = 3 check nodes, connected by $|E|$ = 8 edges.



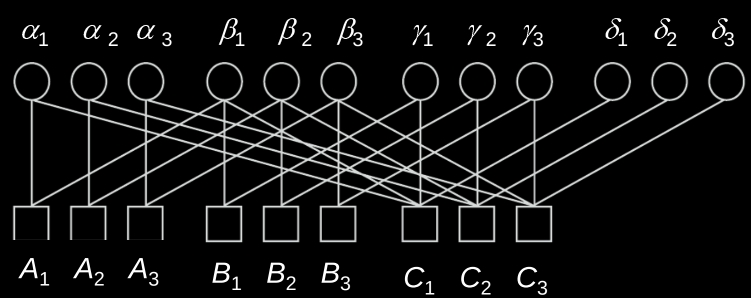
  - In this case Tanner graph of an (n = 4, k = 1) LDPC code (in this case, a repetition code).
- Double edges are allowed



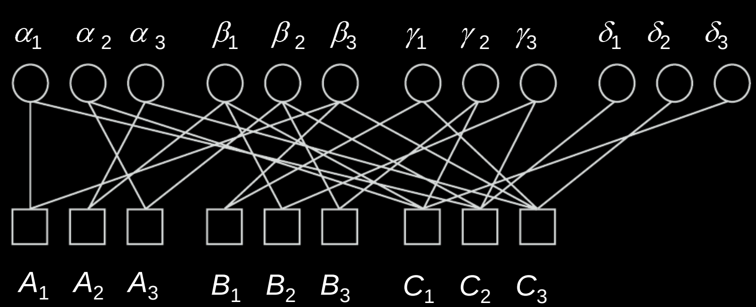
Protograph codes



$$H = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$



$$H = \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ B_1 \\ B_2 \\ B_3 \\ C_1 \\ C_2 \\ C_3 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} \alpha_1 & \alpha_2 & \alpha_3 & \beta_1 & \beta_2 & \beta_3 & \gamma_1 & \gamma_2 & \gamma_3 & \delta_1 & \delta_2 & \delta_3 \end{array} \\ \left[\begin{array}{ccc|ccc|ccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & & & & & & \\ 0 & 0 & 1 & 1 & 0 & 0 & & 0 & & & 0 & \\ 0 & 1 & 0 & 0 & 1 & 0 & & & & & & \\ \hline & & & 0 & 0 & 1 & 1 & 0 & 0 & & & \\ & 0 & & 1 & 0 & 0 & 0 & 0 & 1 & & 0 & \\ & & & 0 & 1 & 0 & 0 & 1 & 0 & & & \\ \hline 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right] \end{array}$$



Parity check masking

- Start from a quasi-cyclic code and force some blocks to be zeros (in the Tanner graph, disconnect groups of checks and variables)

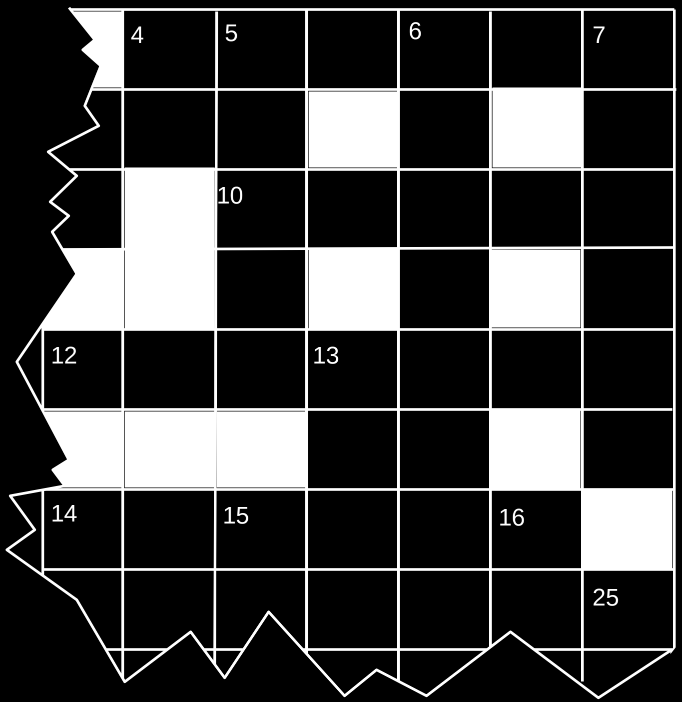
$$H = \left[\begin{array}{ccccc|ccccc|ccccc|ccccc|ccccc}
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
\hline
1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0\\
\hline
1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0\\
0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&0
\end{array}\right]$$

$$H = \left[\begin{array}{ccccc|ccccc|ccccc|ccccc|ccccc}
1&0&0&0&0&1&0&0&0&0&&&&&&1&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&1&0&0&0&&&&&&0&1&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&1&0&0&&&0&&&0&0&1&0&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&1&0&&&&&&0&0&0&1&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&0&1&&&&&&0&0&0&0&1&0&0&0&0&1\\
\hline
&&&&&0&0&0&0&1&0&0&0&1&0&&&&&&0&1&0&0&0\\
&&&&&1&0&0&0&0&0&0&0&0&1&&&&&&0&0&1&0&0\\
&&0&&&0&1&0&0&0&1&0&0&0&0&&&0&&&0&0&0&1&0\\
&&&&&0&0&1&0&0&0&1&0&0&0&&&&&&0&0&0&0&1\\
&&&&&0&0&0&1&0&0&0&1&0&0&&&&&&1&0&0&0&0\\
\hline
1&0&0&0&0&&&&&&0&1&0&0&0&0&0&0&0&1&&&&&\\
0&1&0&0&0&&&&&&0&0&1&0&0&1&0&0&0&0&&&&&\\
0&0&1&0&0&&&0&&&0&0&0&1&0&0&1&0&0&0&&&0&&\\
0&0&0&1&0&&&&&&0&0&0&0&1&0&0&1&0&0&&&&&\\
0&0&0&0&1&&&&&&1&0&0&0&0&0&0&0&1&0&&&&&
\end{array}\right]$$

Decoding by belief propagation

Crossword puzzles

- Iterate!



Across:

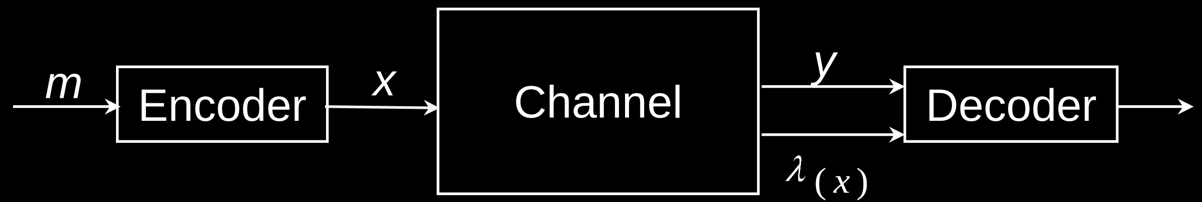
4 Animal with long ears and a short tail.
10 Person who is in charge of a country.
12 In no place.

Down:

5 Pointer, weapon fired from a bow.
6 Accept as true.
7 A place to shoot at; objective.

Decoders for channels with soft outputs

- In addition to the channel value, a measure of bit reliability is also provided



- Bit log-likelihood ratio given y_i.

$$\lambda(x_i) = \log \frac{P(x_i = 0 \mid y_i)}{P(x_i = 1 \mid y_i)}$$

$$= \log \frac{\dfrac{p(y_i \mid x_i = 0) P(x_i = 0)}{p(y_i)}}{\dfrac{p(y_i \mid x_i = 1) P(x_i = 1)}{p(y_i)}} = \log \frac{p(y_i \mid x_i = 0) P(x_i = 0)}{p(y_i \mid x_i = 1) P(x_i = 1)}$$

$$= \log \frac{p(y_i \mid x_i = 0)}{p(y_i \mid x_i = 1)} + \log \frac{P(x_i = 0)}{P(x_i = 1)}$$

Log-likelihood ratio

- Without prior knowledge on x_i

$$\gamma_i = \lambda(x_i) = \log \frac{p(y_i \mid x_i = 0)}{p(y_i \mid x_i = 1)}$$

- For AWGN ($y_i = x_i + n_i,\ n_i \sim N(\ 0,\ \sigma^2\ )$)

$$\gamma_i = \log \frac{p(y_i \mid x_i = 0)}{p(y_i \mid x_i = 1)} = \frac{1}{2\sigma^2} - (y_i - 1)^2 + (y_i + 1)^2 = \frac{y_i}{2\sigma^2}$$

- For BSC with parameter α

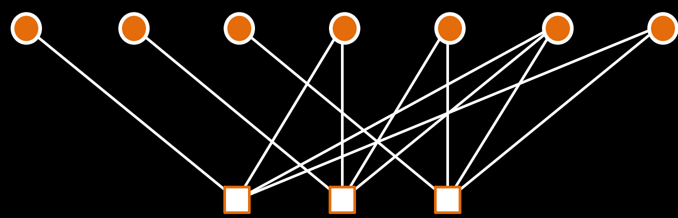
$$\gamma_i = \begin{cases} \log \dfrac{1-\alpha}{\alpha} & \text{if } y_i = 0 \\ \log \dfrac{\alpha}{1-\alpha} & \text{if } y_i = 1 \end{cases}$$

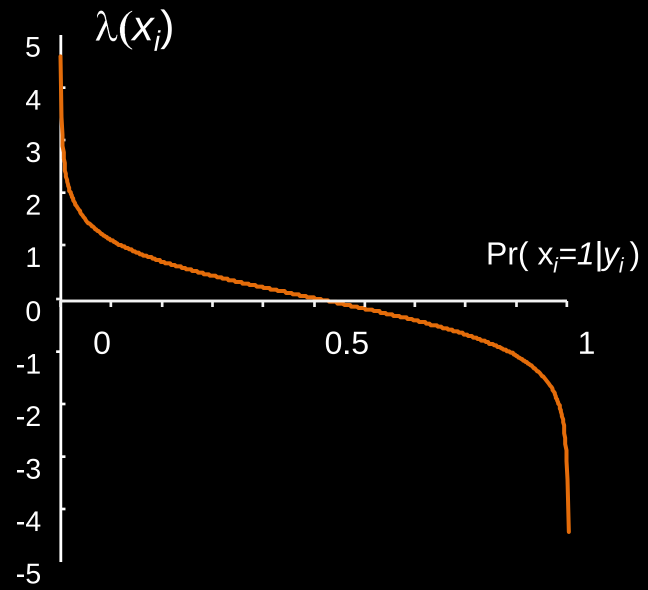
Message-passing

- Soft outputs (x_i, λ_i)
  - x_i – an estimate of the i[th] bit
  - λ_i- belief, reliability, likelihood, likelihood ratio

Example:

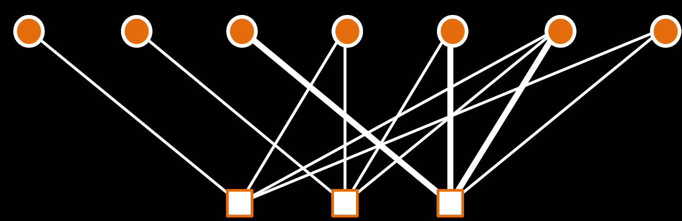
x :	1	1	0	0	1	0	1
$\hat{x}$:	1	1	0	0	1	**1**	1
$\mu(\hat{x})$:	- 6	-1	+10	+2	-1	**-2**	-5





Soft decoding example

x :	1	1	0	0	1	0	1
$\hat{x}$:	1	1	0	0	1	**1**	1
$\mu(\hat{x})$:	- 6	-1	+10	+2	-1	**-2**	-5

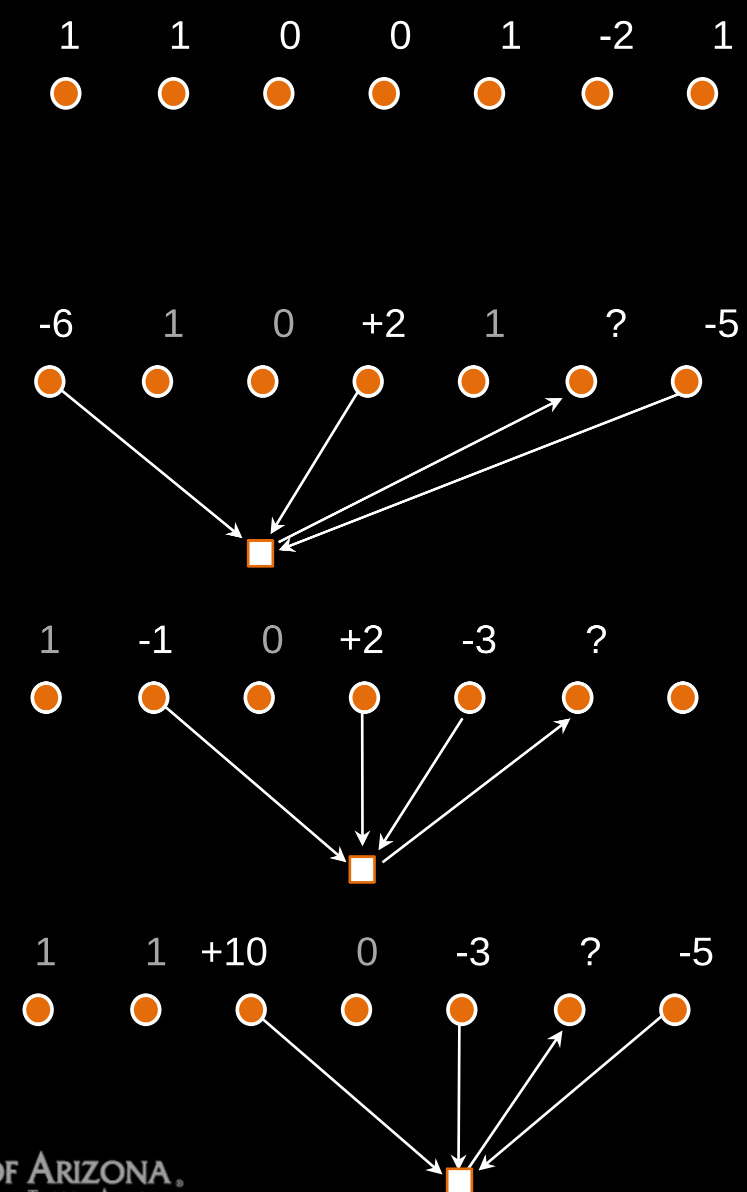


$$M_1 = \min(|-6|, |+2|, |-5|) = 2$$

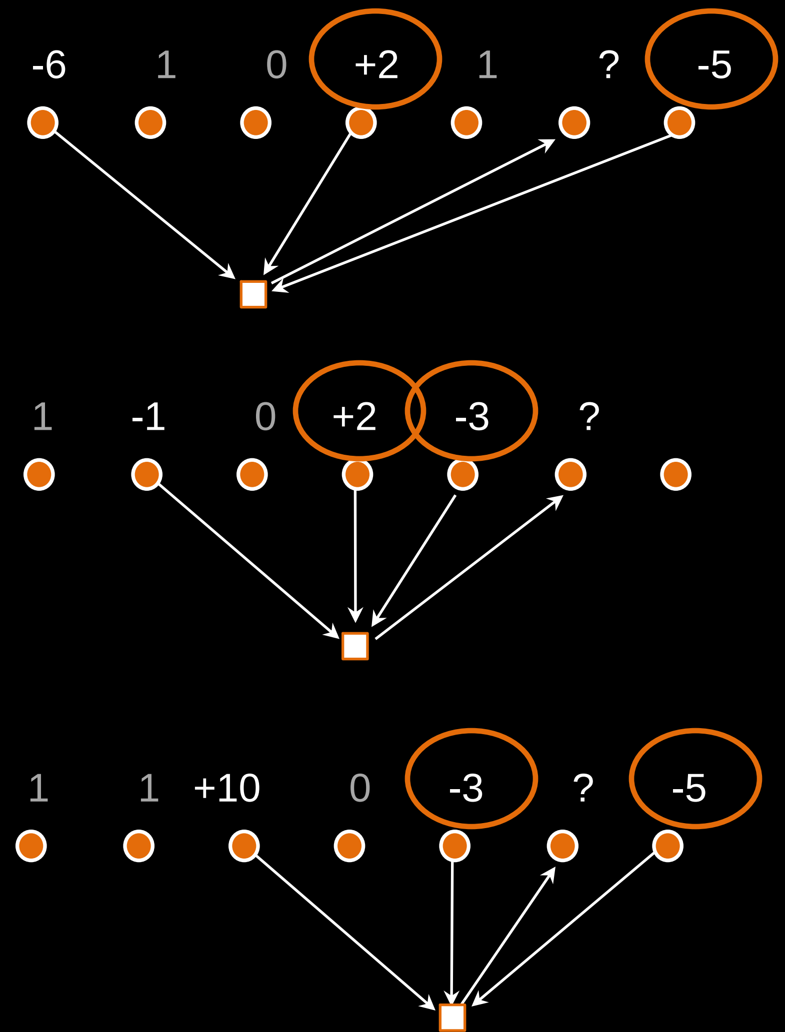
$$S_1 = sign(-6) \cdot sign(+2) \cdot sign(-5) = +1$$

$$A_1 = S_1 \cdot M_1$$

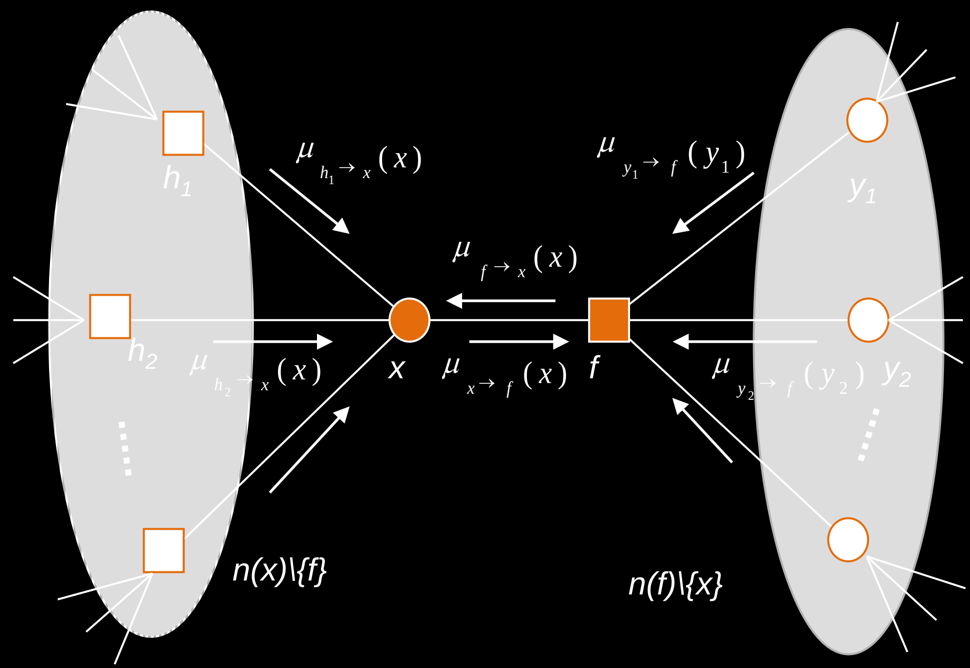
$$A_0 = A_0 + A_1 + A_2 + A_3$$



Side remark: some bits “voted” twice



The min-sum update rule



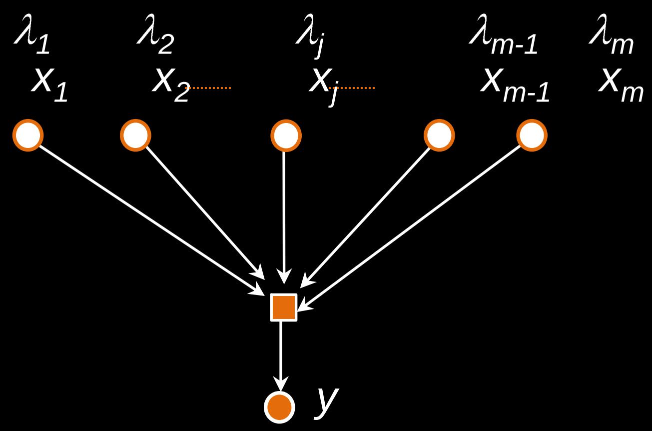
$$\mu_{x \to f} = \lambda_x + \sum_{h \in n(x) \setminus \{f\}} \mu_{h \to f}$$

$$\mu_{f \to x}(x) = \prod_{y \in n(f) \setminus \{x\}} \operatorname{sgn}(\mu_{y \to f}) \min_{y \in n(f) \setminus \{x\}} |\mu_{y \to f}|$$

$$g_i(x_i) = \lambda(x_i) + \sum_{h \in n(x_i)} \mu_{h \to x_i}$$

Derivation of the check update rule

- Given the log-likelihoods of $(x_j)_{1\le j\le m}$ find the log-likelihood of y, $L(y)$.



$$L(y) = \log \frac{\Pr\{y = 0\}}{\Pr\{y = 1\}} = \log \frac{\Pr\{\#\text{"1" } in\ x\ is\ even\}}{\Pr\{\#\text{"1" } in\ x\ is\ odd\}}$$

$$L(y) \simeq \prod_{1\le j\le m} \operatorname{sgn}(\lambda_j) \cdot \sum_{1\le j\le m} |\lambda_j|$$

Derivation of the check update rule

$$\textit{Bernoulli trials:}\quad \Pr\{x = 0\} = q, \quad \Pr\{x = 1\} = p$$

$$(q + p)^m = \sum_{0 \le j \le m} \binom{m}{j} p^j \cdot q^{m-j}$$

$$(q - p)^m = \sum_{0 \le j \le m} (-1)^j \binom{m}{j} p^j \cdot q^{m-j}$$

$$\Pr\{\#\text{"1" } in\ x\ is\ even\} = \frac{1}{2}\left[(q + p)^m + (q - p)^m\right]$$

$$\Pr\{\#\text{"1" } in\ x\ is\ odd\} = \frac{1}{2}\left[(q + p)^m - (q - p)^m\right]$$

Generalization:

$$\Pr\{x_j = 0\} = q_j, \quad \Pr\{x_j = 0\} = p_j, \quad 0 \le j \le m$$

$$\Pr\{\#\text{"1" } in\ x\ is\ even\} = \frac{1}{2}\left(\prod_{1 \le j \le m}(q_j + p_j) + \prod_{1 \le j \le m}(q_j - p_j)\right) = \frac{1}{2}\left(1 + \prod_{1 \le j \le m}(q_j - p_j)\right)$$

$$\Pr\{\#\text{"1" } in\ x\ is\ odd\} = \frac{1}{2}\left(\prod_{1 \le j \le m}(q_j + p_j) - \prod_{1 \le j \le m}(q_j - p_j)\right) = \frac{1}{2}\left(1 - \prod_{1 \le j \le m}(q_j - p_j)\right)$$

Derivation of the check update rule

$$L(y) = \log \frac{\Pr\{y=0\}}{\Pr\{y=1\}} = \log \frac{\Pr\{\#\text{"1"} \ in\ x\ is\ even\}}{\Pr\{\#\text{"1"} \ in\ x\ is\ odd\}}$$

$$= \log \frac{1 + \prod_{1\le j\le m}\left(\frac{e^{\lambda_j}}{1+e^{\lambda_j}} - \frac{1}{1+e^{\lambda_j}}\right)}{1 - \prod_{1\le j\le m}\left(\frac{e^{\lambda_j}}{1+e^{\lambda_j}} - \frac{1}{1+e^{\lambda_j}}\right)}$$

$$= \log \frac{1 + \prod_{1\le j\le m}\frac{e^{\lambda_j}-1}{e^{\lambda_j}+1}}{1 - \prod_{1\le j\le m}\frac{e^{\lambda_j}-1}{e^{\lambda_j}+1}}$$

$$L(y) = \log \frac{1 + \prod_{1\le j\le m}\frac{e^{\lambda_j/2}-e^{-\lambda_j/2}}{e^{\lambda_j/2}+e^{-\lambda_j/2}}}{1 - \prod_{1\le j\le m}\frac{e^{\lambda_j/2}-e^{-\lambda_j/2}}{e^{\lambda_j/2}+e^{-\lambda_j/2}}}$$

$$= \log \frac{1 + \prod_{1\le j\le m}\tanh\left(\frac{\lambda_j}{2}\right)}{1 - \prod_{1\le j\le m}\tanh\left(\frac{\lambda_j}{2}\right)}$$

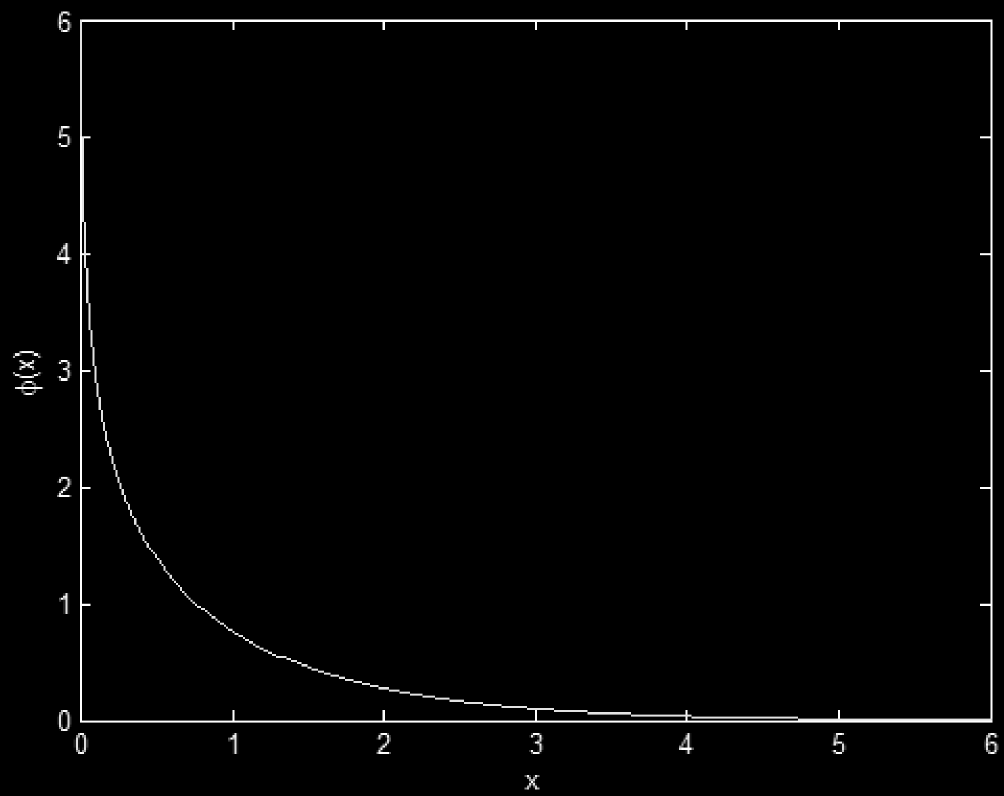
$$= 2\cdot\frac{1}{2}\cdot\log \frac{1 + \prod_{1\le j\le m}\tanh\left(\frac{\lambda_j}{2}\right)}{1 - \prod_{1\le j\le m}\tanh\left(\frac{\lambda_j}{2}\right)}$$

$$= 2\,\mathrm{artanh}\left(\prod_{1\le j\le m}\tanh\left(\frac{\lambda_j}{2}\right)\right)$$

$$L(y) = 2\,\mathrm{artanh}\left(\prod_{1\le j\le m}\tanh\left(\frac{\lambda_j}{2}\right)\right)$$

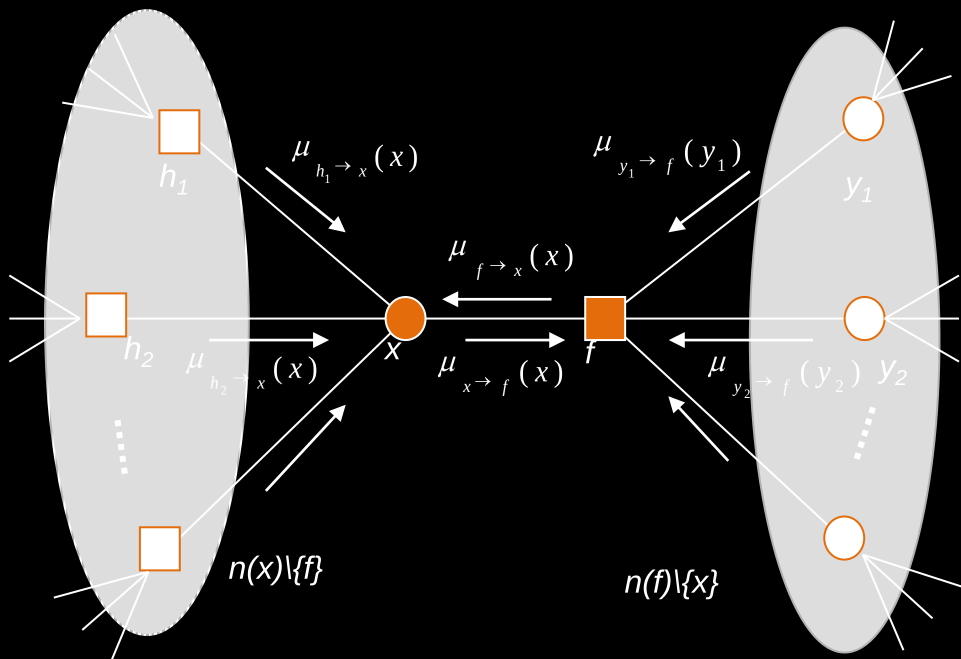
Min-sum approximation

- $\phi(x) = -\log \tanh(x/2) = \log((e^x+1)/(e^x-1)) = \phi^{-1}(x)$



- $$\phi\left(\sum_i \phi\left(|\mu_{i\to f}|\right)\right) \approx \phi\left(\phi\left(\min_i |\mu_{i\to f}|\right)\right) = \min_{i'} |\mu_{i\to f}|$$

Sum-product algorithm (Kschischang et. al.)



$$\mu_{x\to f}(x) = \prod_{h\in n(x)\backslash\{f\}} \mu_{h\to x}(x)$$

$$\mu_{f\to x}(x) = \sum_{\sim\{x\}}\left(f(X) \prod_{h\in n(f)\backslash\{x\}} \mu_{y\to f}(y)\right)$$

$$g_i(x_i) = \prod_{h\in n(x_i)} \mu_{h\to x_i}(x_i)$$

The sum-product algorithm

- The update rule

$$\begin{aligned} \omega_{i\to\alpha}^{(0)} &= \gamma_i \\ \varpi_{\alpha\to i}^{(k)} &= 2\tanh^{-1}\left(\prod_{j\in\mathcal{N}(\alpha)\setminus i} \tanh\left(\frac{1}{2}\omega_{j\to\alpha}^{(k-1)}\right)\right) \\ \omega_{i\to\alpha}^{(k)} &= \gamma_i + \sum_{\delta\in\mathcal{N}(i)\setminus\alpha} \varpi_{\delta\to i}^{(k)} \end{aligned}$$

- The result of decoding after k iterations, denoted by $\mathbf{x}^{(k)}$
- is determined by the sign of

$$m_i^{(k)} = \gamma_i + \sum_{\alpha\in\mathcal{N}(i)} \varpi_{\alpha\to i}^{(k)}$$

- If $m_i^{(k)} > 0$ then $x_i^{(k)} = 0$ otherwise $x_i^{(k)} = 1$

The min-sum algorithm

- In the limit of high SNR, when the absolute value of the messages is large, the sum-product becomes the min-sum algorithm, where the message from the check β to the bit i looks like:

$$\varpi^{(k)}_{\beta\to i} \;=\; \min\left|\omega^{(k-1)}_{*\setminus i\to\beta}\right| \cdot \prod_{j\in\mathcal{N}(\beta)\setminus i} \operatorname{sign}\left(\omega^{(k-1)}_{j\to\beta}\right)$$