# On the construction of Tanner graphs

Jesús Martínez Mateo

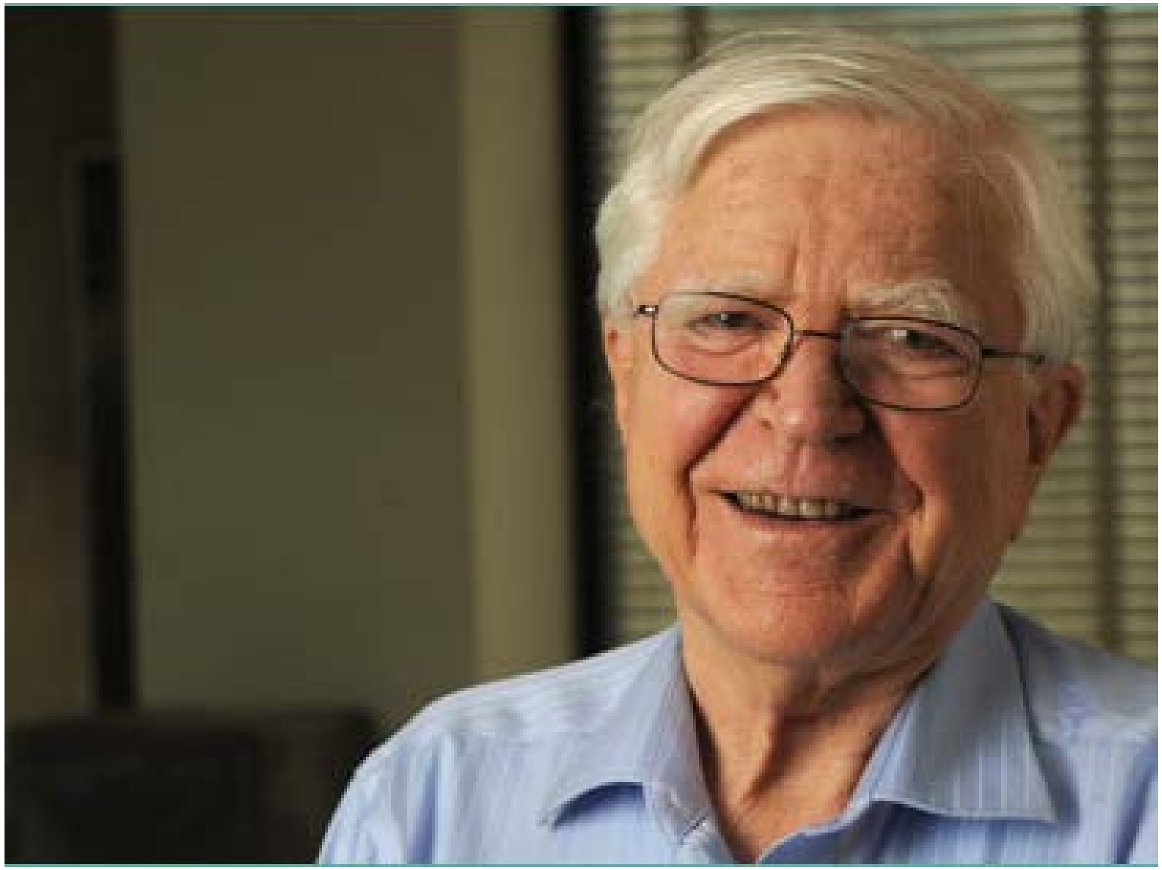
Universidad Politécnica de Madrid

# Outline

- Introduction
  - Low-density parity-check (LDPC) codes
- LDPC decoding
  - Belief propagation based algorithms
- Ensembles of LDPC codes
- LDPC code construction
  - PEG algorithm
  - Cycles, stopping and trapping sets
- Simulation results
- Summary

Low-density parity check codes

# INTRODUCTION



http://www.rle.mit.edu/rgallager/

## Low-density parity-check (LDPC) codes

Introduced by Robert G. Gallager in 1963, but neglected for years. Rediscovered in 90s by MacKay & Neal, and quickly showed that irregular LDPC codes easily outperform the best turbo codes.

# Linear codes (encoding)

- Error correction using parity-checks

  Multiple constraints (parity-check equations) often written in matrix form, *H*, **parity-check matrix**, a valid codeword *x* then satisfies

$$H\boldsymbol{x}^t = \boldsymbol{0}$$

$$\underbrace{\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}}_{H} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{cases} x_3 = x_1 \\ x_4 = x_2 \\ x_5 = x_1 + x_2 \end{cases}$$

$$(x_1\ x_2\ x_3\ x_4\ x_5) = (x_1\ x_2) \underbrace{\begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}}_{G}$$

# Linear codes (decoding I)

- Maximum likelihood decoder

  Knowing the binary received string, *y*, the best decoder will choose the codeword closest in **Hamming distance** to *y* (or randomly one of them)

  – Optimal, but too computational expensive

  The received string has to be compared to every other codeword in the code

**Classical block codes**

✓usually short, and

✓algebraically designed

# Linea codes (decoding II)

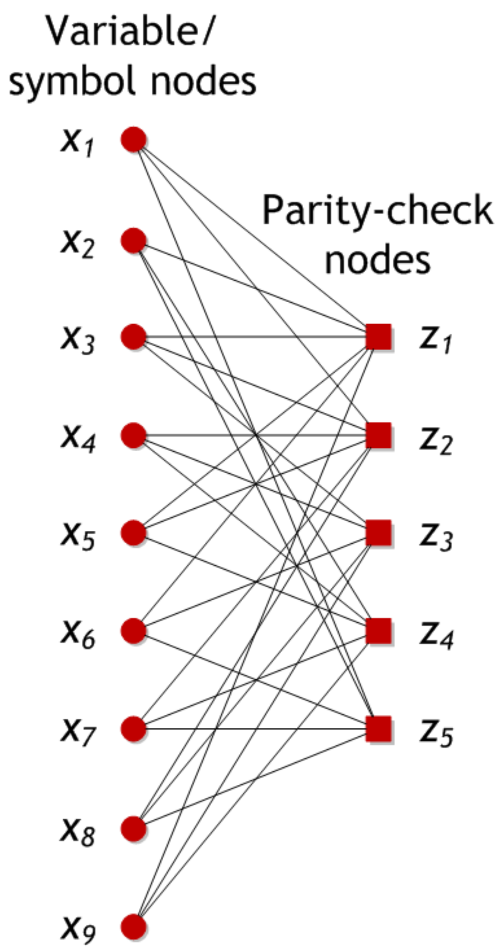
- Alternatives:
  - Iterative decoding

Using a graphical representation of the parity-check matrix.

e.g. message passing algorithms

Operate by passing messages along the edges of the Tanner graph.

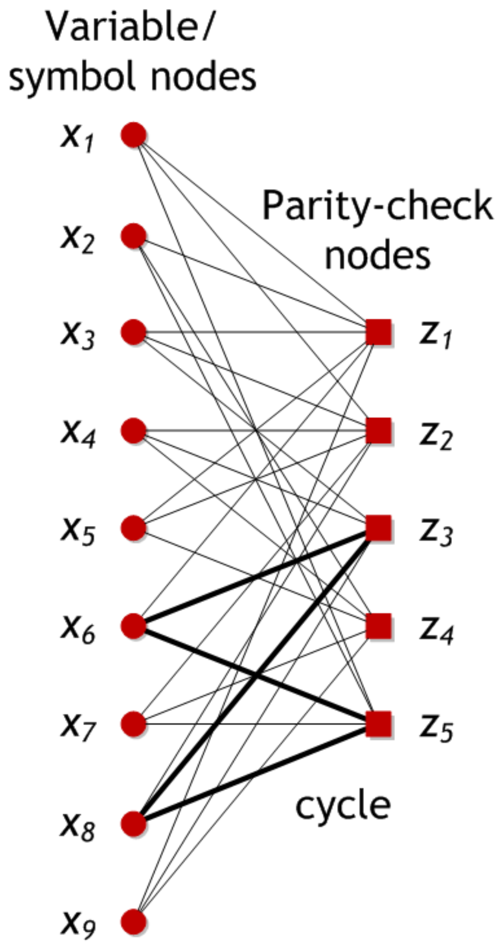
# Tanner or bipartite graphs




- The graph consists of two sets of nodes commonly referred to as:
  - Variable, bit or symbol nodes (for the codewords)
  - Check or parity-check nodes (for the parity-check equations)

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

# Cycles, Lollipops and Girth




- A **cycle** in a Tanner graph is a sequence of connected nodes that start and end at the same node, and contain other vertices no more than once
  - The length of a cycle is the number of edges it contains
- The girth is the size of the smallest cycle in the graph

Message-passing algorithms

Belief propagation decoding

Sum-product algorithm

# LDPC DECODING

# Message passing algorithms

- Operate by passing messages along the edges of the Tanner graph
  - the decoding performance depends on the number of edges
- Also known as **iterative decoding** algorithms:
  - messages from symbol (check) nodes to check (symbol) nodes are exchanged iteratively until a result is achieved
- Obey the extrinsic information principle: only extrinsic information is passed along

  i.e. the **outgoing message** on an edge *e* is a function of all the incoming messages except the message received on *e*

  and the received value in the case of messages from symbol to check nodes

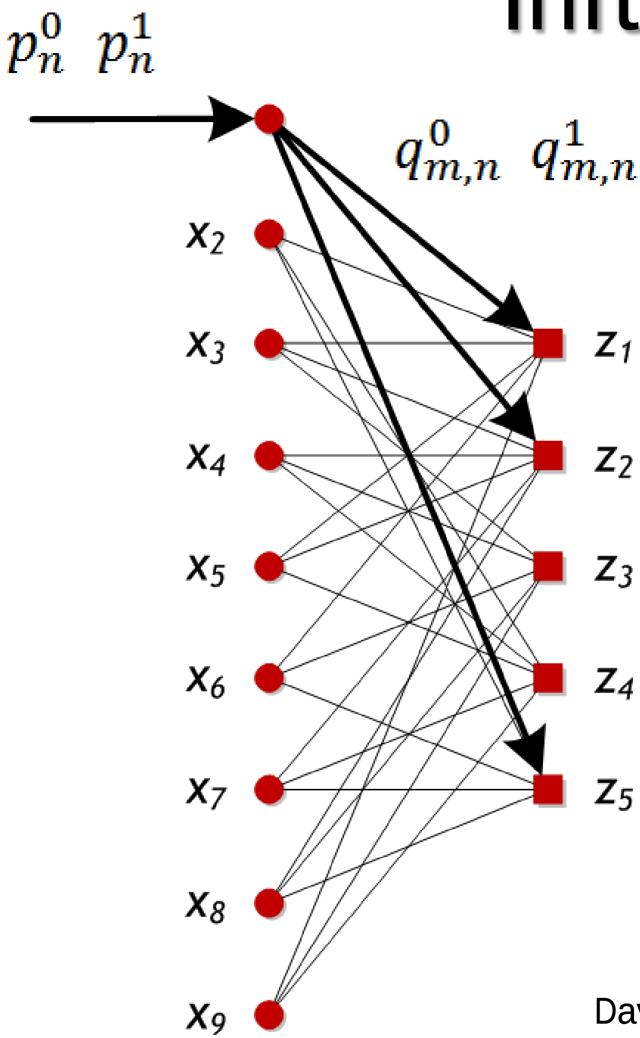
# Belief propagation decoding

- Different algorithms are considered depending on the information exchanged in the passed message:
  - Bit flipping and belief propagation decoding are well known message passing algorithms

- Belief propagation decoding:
  - Messages are probabilities which represent the level of belief on a codeword bit value
  - Variants: sum-product and min-sum (Viterbi) algorithms

T. Richardson, R. Urbanke, IEEE Trans. Inf. Theory, vol. 47, no. 2, pp. 599-618 (2001)

# Sum-product algorithm: Initialization




Compute the prior probabilities:

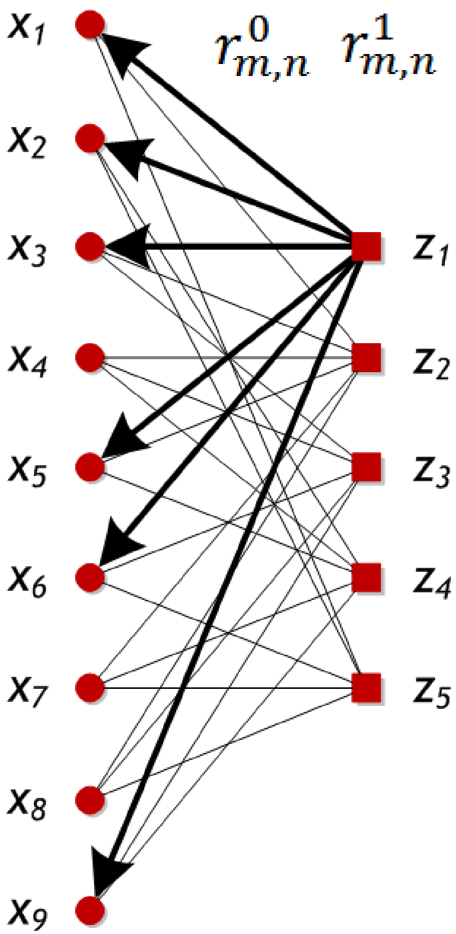
$$p_n^0 = \Pr(x_n = 0)$$
$$p_n^1 = \Pr(x_n = 1) = 1 - p_n^0$$

Initialize messages from symbols:

$$q_{m,n}^0 = p_n^0$$
$$q_{m,n}^1 = p_n^1$$

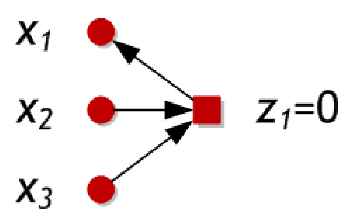
David J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press (2003)

# Sum-product algorithm: Horizontal step (1)




$$r^0_{m,n} = \sum \Pr(z_n | x_n = 0) \prod q^{x_{n'}}_{m,n'}$$

$$r^1_{m,n} = \sum \Pr(z_n | x_n = 1) \prod q^{x_{n'}}_{m,n'}$$

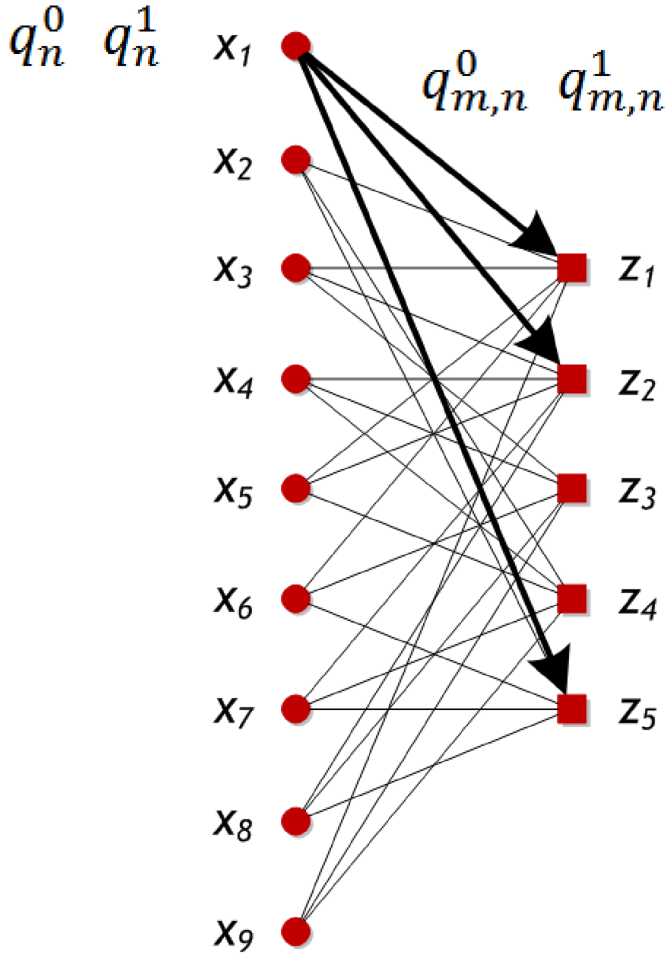



$$r^0_{1,1} = q^0_{2,1} q^0_{3,1} + q^1_{2,1} q^1_{3,1}$$

$$r^1_{1,1} = q^0_{2,1} q^1_{3,1} + q^1_{2,1} q^0_{3,1}$$


David J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press (2003)

# Sum-product algorithm: Vertical step (2)




$$q_{m,n}^0 = \alpha_{m,n}\, p_n^0 \prod_{m' \in \mathcal{M}(n) \backslash n} r_{m',n}^0$$

$$q_{m,n}^1 = \alpha_{m,n}\, p_n^1 \prod_{m' \in \mathcal{M}(n) \backslash n} r_{m',n}^1$$

Pseudo posterior probabilities:

$$q_n^0 = \alpha_n\, p_n^0 \prod_{m \in \mathcal{M}(n)} r_{m,n}^0$$

$$q_n^1 = \alpha_n\, p_n^1 \prod_{m \in \mathcal{M}(n)} r_{m,n}^1$$

David J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press (2003)

Generating polynomials

Symbol and check node degree distributions

# ENSEMBLES OF LDPC CODES

# Ensembles of LDPC codes

- Usually used the ensemble of all possible codes with certain parameters (e.g. the degree distribution of symbol and check nodes) instead of a particular parity-check matrix

- ## Generating polynomials

Two polynomials *$\lambda(x)$* and *$\rho(x)$* representing symbol and check node degree distributions, respectively

$$\lambda(x) = \sum_i \lambda_i x^{i-1} \qquad s.t. \sum_i \lambda_i = 1, \qquad \sum_j \rho_j = 1$$

$$\rho(x) = \sum_j \rho_i x^{j-1}$$

i.e. $\lambda_i$ ($\rho_j$) denote the fraction of edges connected to *i*-degree (*j*-degree) symbol (check) nodes

# Density and differential evolution

- The asymptotic performance (capacity) of a family or ensemble of LDPC codes can be determined using the density evolution [1]
  - The algorithm analyze the convergence of a particular degree distribution for the cycle-free case, i.e. assuming there is no cycles in the graph what never happens with finite-length codes
- Two common variants:
  - Gaussian approximation
  - Discretized density evolution [2]
- Good families or ensembles of LDPC codes can be determined using the differential evolution [3]


1. T. Richardson, R. Urbanke, IEEE Trans. Inf. Theory, vol. 47, no. 2, pp. 599-618 (2001)
2. S.-Y. Chung, G. Forney, T. Richardson, R. Urbanke, IEEE Commun. Lett., vol. 5, no. 2, pp. 58-60 (2001)
3. A. Shokrollahi, R. Storn, in Proc. IEEE Int. Symp. Inf. Theory (2000)

Original LDPC code construction proposed by Gallager

MacKay and Neal construction

Progressive edge-growth (PEG) algorithm

# LDPC CODE CONSTRUCTION

# Original LDPC code construction proposed by Gallager

- Valid only for the construction of regular LDPC codes
- Rows are divided into a number of sets
  - First set: rows contains a number of consecutive 1’s (ordered from left to right)
  - Next: rows are chosen randomly from a column permutation of the first set

$$H = \left(\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
\hline
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1
\end{array}\right)$$

# MacKay and Neal construction

- Columns are filled from left to right
  - Number of 1's chosen per column (edges) according to a degree distribution
  - Rows are chosen randomly from those that are **not full**

    i.e. the algorithm look for a regular check node degree distribution
- Valid for regular and irregular LDPC codes
- The algorithm can be easily adapted to avoid 4-cycles

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

# Progressive edge-growth (PEG) algorithm

- A Tanner graph is constructed connecting symbol and check nodes in an edge-by-edge manner
  - Symbols are processed sequentially
  - New check nodes are connected to the current symbol till the number of edges (symbol degree) is reached
- The algorithm consists of two basic procedures:
  - a **local graph expansion** (used to detect and avoid short cycles)
  - and, a **check node selection procedure**

**The PEG algorithm construct codes having a large girth (i.e. avoiding short cycles)**

# PEG algorithm (I)

**for** $j = 1$ to $n$ **do**
  **for** $k = 1$ to $deg(x_j)$ **do**
    **if** $k = 1$ **then**
      $E^1_{x_j} \leftarrow (z_i, x_j)$, where $E^1_{x_j}$ is the first edge incident to $x_j$, and $z_i$ is a check node such that it has the *lowest check-node degree* under the current graph setting $E_{x_1} \cup E_{x_2} \cup \cdots \cup E_{x_{j-1}}$
    **else**
      expand a subgraph from symbol node $x_j$ up to depth $\ell$ under the current graph setting, such that $\mathcal{N}^{\ell}_{x_j} = \mathcal{N}^{\ell+1}_{x_j}$, or $\overline{\mathcal{N}}^{\ell+1}_{x_j} = \emptyset$
      $E^k_{x_j} \leftarrow (z_i, x_j)$, where $E^k_{x_j}$ is the $k$th edge incident to $x_j$ and $z_i$ is a check node from the set $\overline{\mathcal{N}}^{\ell}_{x_j}$ having the *lowest check-node degree*.
    **end if**
  **end for**
**end for**

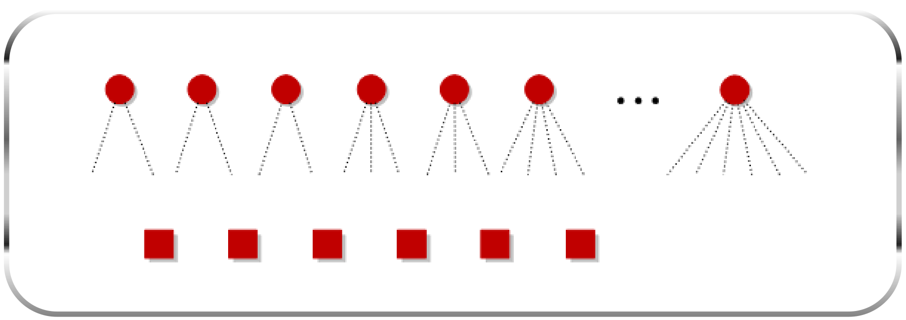
X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386-398 (2005)

# PEG algorithm (II)

- Input parameters:
  - Parity-check matrix dimension (i.e. num. symbols & check nodes)
  - Symbol node degree distribution

Note that the check node degree distribution is not considered in the original PEG algorithm

- Given the generating polynomial $\lambda(x)$ and the codeword length $n$, we calculate the number of edges per symbol node, $deg(x_i)$
  - While $\lambda_i$ denote the fraction of edges connected to $i$-degree symbol nodes
  - $\lambda^*_i$ denote the fraction of symbols with degree $i$



$$\lambda_i^* = \frac{\frac{\lambda_i}{i}}{\sum_i \frac{\lambda_i}{i}}$$

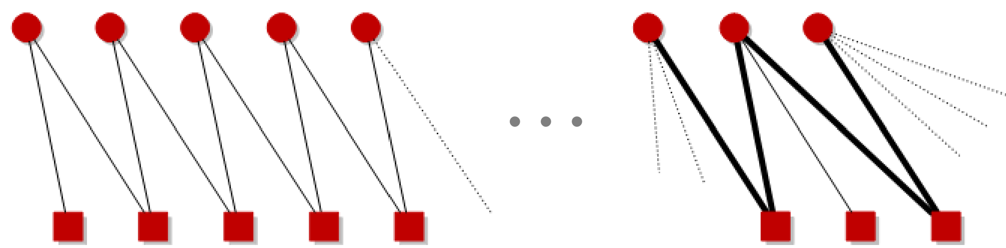
# PEG algorithm (III)

- According to the first condition:
  - Every symbol node is firstly connected to the current graph

In particular, 2-degree symbol nodes are connected in zigzag

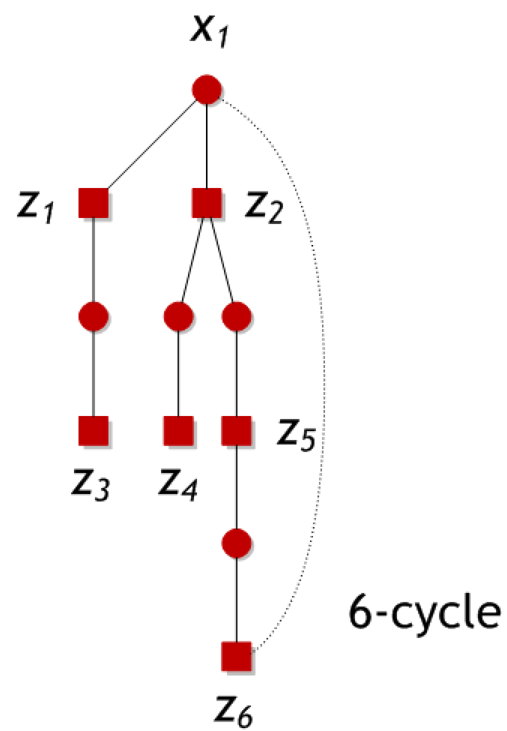
Note that 0-degree check nodes (i.e. those check nodes that are not currently connected to any symbol node) are not considered in the current graph

2-degree zigzag



X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386-398 (2005)

# PEG algorithm (IV)




$$\mathcal{N}_{x_1}^1 = \{z_1, z_2\}$$
$$\mathcal{N}_{x_1}^2 = \{z_1, z_2, z_3, z_4, z_5\}$$
$$\bar{\mathcal{N}}_{x_1}^2 = \{z_6\}$$

- Check node selection:
  - Lowest check node degree criterium over the set of candidate checks
- Candidate check nodes:
  - unvisited (non expanded) check nodes (including those that are not in the current graph)

    no cycle is produced
  - the set of expanded check nodes with highest depth, otherwise

    it produces the longest (possible) cycle

X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386-398 (2005)

# PEG algorithm (proposed optimizations)

- Two proposed optimizations in the original PEG algorithm:
  - Nongreedy version:

    for long-block codes or low-rate codes (in which the minimum distance is –in principle- large), it may be favourable to limit the maximum depth $\ell$

  - Look-ahead enhanced version:

    when several choices exist for placing the *k*th edge, we look one step ahead and choose the one (check node) having the maximum possible depth $\ell$ in the expanded subgraph

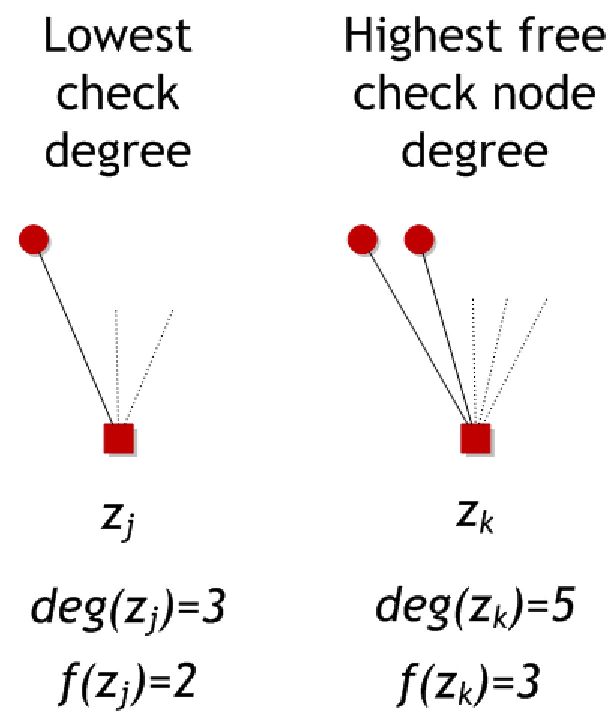
X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386-398 (2005)

# Improved PEG algorithm

- Both degree distributions, for symbol and check nodes, are considered when changing the edge-selection criterion
  - Instead of the node with the lowest check degree we select the one with highest free check node degree,

    i.e. the difference between the number of currently assigned edges and the total number of edges to be assigned

Note that 2-degree symbol nodes are no longer connected in zigzag

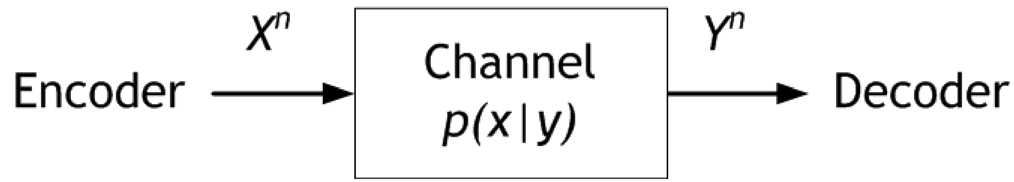



J. Martinez-Mateo, D. Elkouss, V. Martin, IEEE Commun. Lett., vol. 14, no. 12, pp. 1155-1957 (2010)

Performance, Frame/Bit error rate

# SIMULATION RESULTS

# Error model for noisy channels



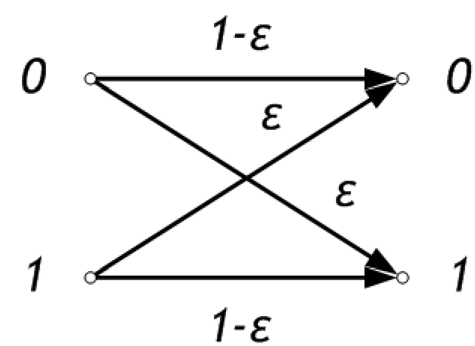

- Discrete channel

  A system consisting of

  - An input and an output alphabets
  - And a probability transition matrix *p(x|y)*

  – Channel capacity

$$C = \max_{p(x)} I(X;Y)$$



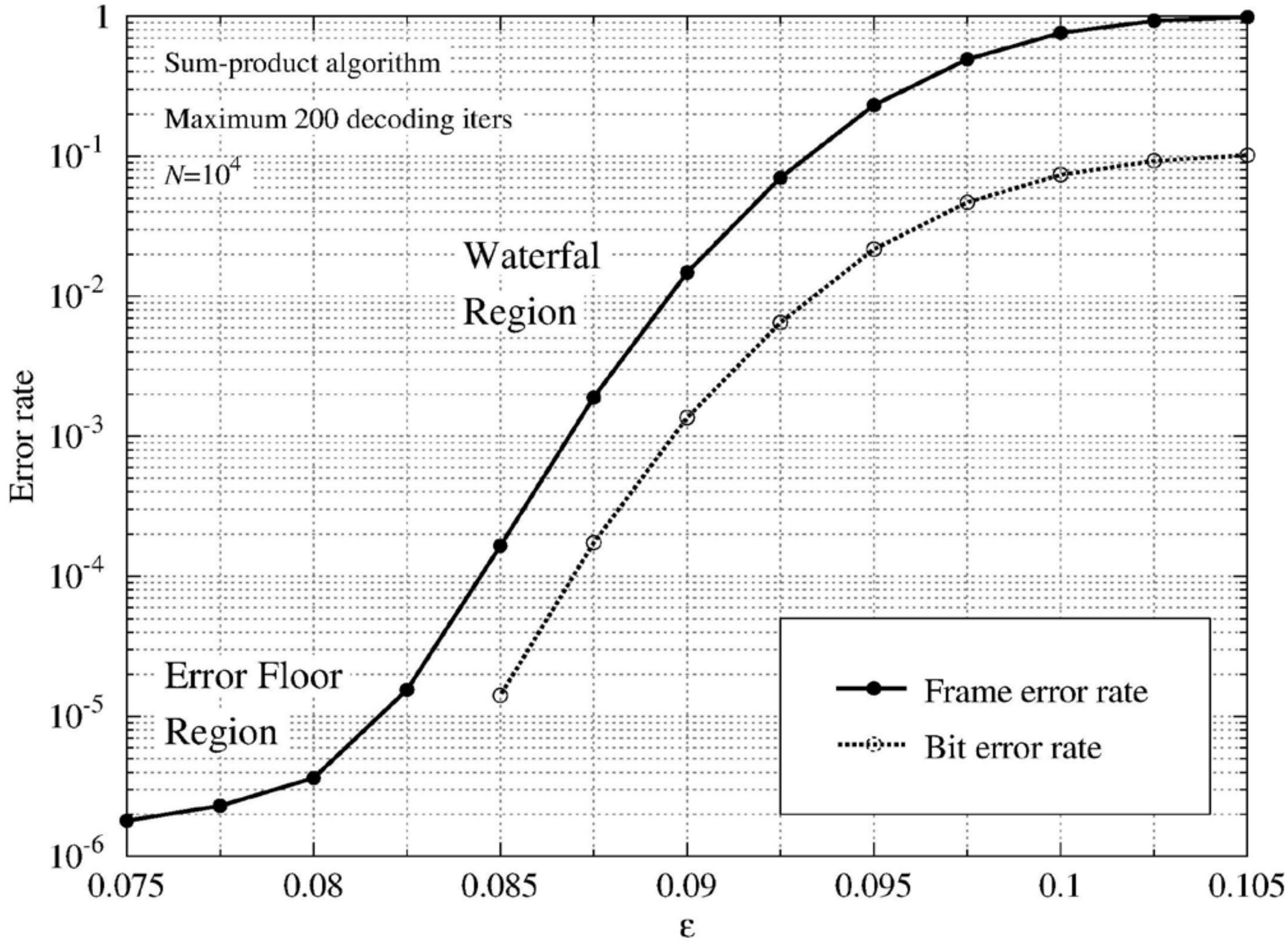

- Binary symmetric channel

  with crossover probability *ε*

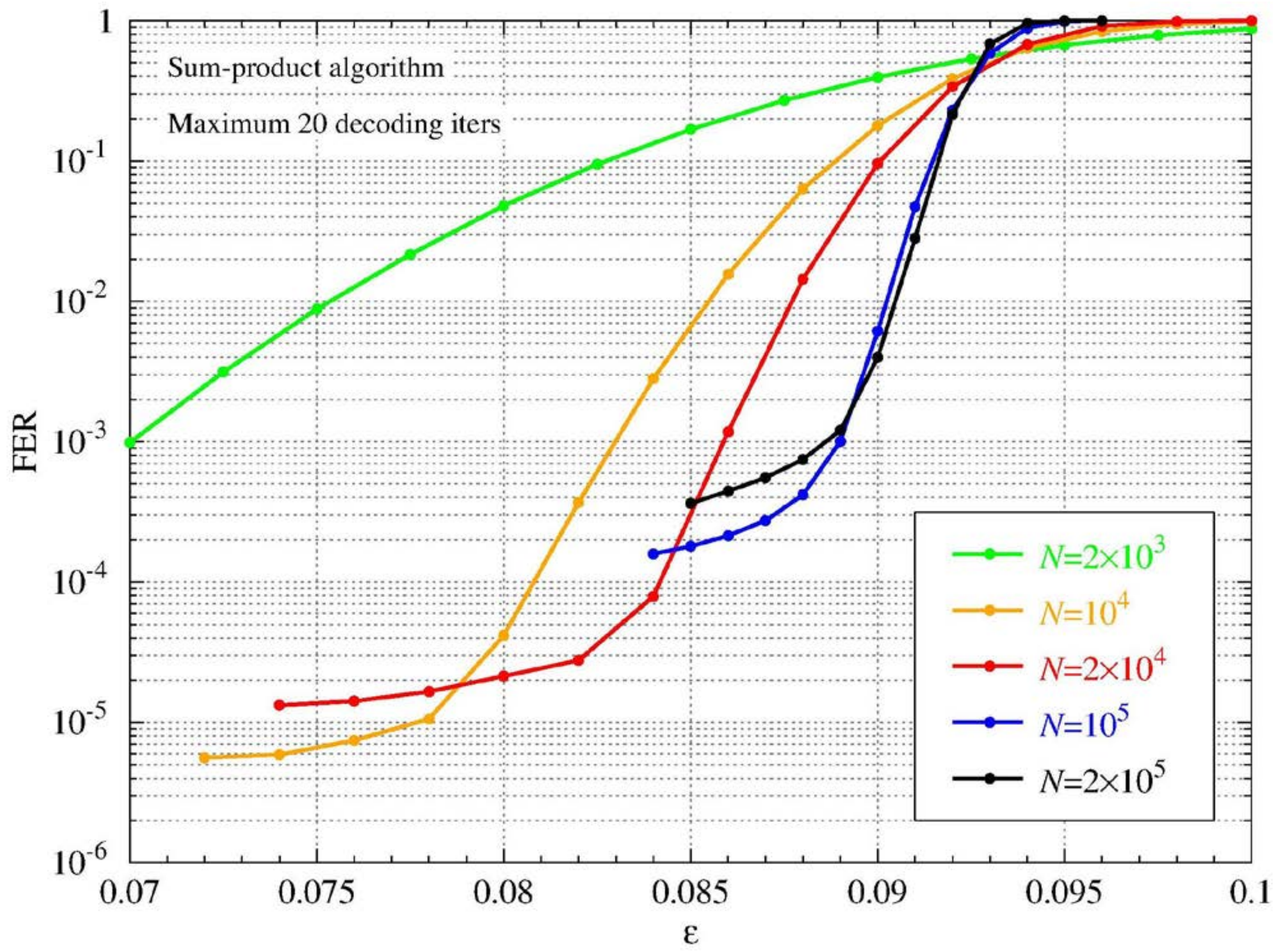
$$C_{BSC(\varepsilon)} = 1 - H(\varepsilon)$$

T. M. Cover, J. A. Thomas, *Elements of Information Theory*, Wiley-Interscience (1991)

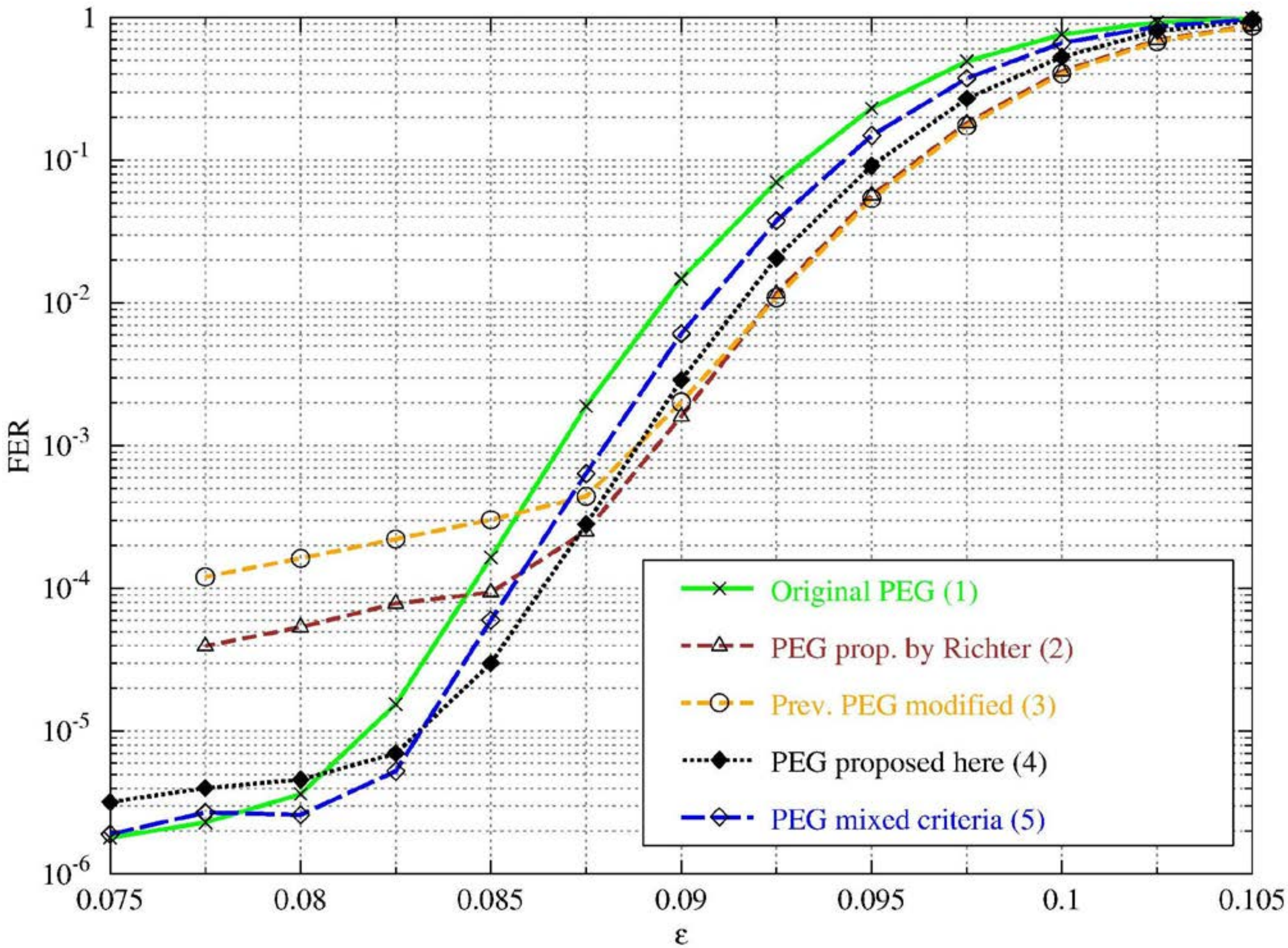
# Performance of LDPC codes (I) Frame and bit error rate



# Performance of LDPC codes (II) FER vs. codeword length



# Performance of LDPC codes (III) Improved PEG



# Performance of LDPC codes (III) Improved PEG [Legend]

1) Original PEG algorithm
2) Improved PEG algorithm proposed by Richter
   in *Proc. Int. Conference on Computer as a Tool*, pp. 1044–1047 (2005)
3) Modified (2): a check node in the current graph is selected when adding the first edge to a symbol node
4) Improved PEG algorithm proposed here
5) Mixed version, the lowest check node degree criterion is used to connect the first edge to a symbol node (not only to 2-degree symbol nodes as proposed here) and the highest free check node degree criterion for the remaining edges

Summary and suggested bibliography

# CONCLUDING REMARKS

# Summary

- Modern coding techniques:
  - Also based on linear codes
  - Iterative decoding using message passing algorithms:
    - Messages along the edges of a code graph
    - Local calculations ('divide and conquer' strategy)
- Ensembles of codes:
  - Designed using density and differential evolution
- Instances of codes:
  - Constructed using a progressive edge-growth algorithm

# Suggested bibliography


**Books and notebooks**

- David J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press (2003)
- Sara J. Johnson, *Introducing Low-Density Parity-Check Codes* (2006)
- T. Richardson, R. Urbanke, *Modern Coding Theory*, Cambridge U. Press (2008)
- T. M. Cover, J. A. Thomas, *Elements of Information Theory*, Wiley (1991)

**Articles in international journals and conferences**

Design of LDPC codes: density and differential evolution

- T. Richardson, R. Urbanke, IEEE Trans. Inf. Theory 47, 599 (2001)
- S.-Y. Chung, G. Forney, T. Richardson, R. Urbanke, IEEE Commun. Lett. 5, 58 (2001)
- A. Shokrollahi and R. Storn, in Proc. IEEE Int. Symp. Inf. Theory (2000)

Construction of LDPC codes: PEG algorithm

- X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, IEEE Trans. Inf. Theory 51, 386 (2005)
- T. Tian, C. Jones, J. Villasenor, R. Wesel, in IEEE Int. Conf. Commun. 5, 3125 (2003)
- S.-H. Kim, J.-S. Kim, D.-S. Kim, H.-Y. Song, IEEE Commun. Lett. 11, 607 (2007)
- J. Martinez-Mateo, D. Elkouss, V. Martin, IEEE Commun. Lett. 14, 1155 (2010)

# Thank you!

Questions?