Channel Coding – Graph-based Codes

Implementation of LDPC Codes & Beyond

Prof. Dr.-Ing. Laurent Schmalen

Overview

- LDPC Codes – Encoding & Matrix Construction
  - Encoding LDPC Codes
  - Practical Code Constructions
- Spatially Coupled LDPC Codes
  - Motivation
  - The Regular Ensemble
  - Density Evolution
  - Burst Erasure

Overview

- LDPC Codes – Encoding & Matrix Construction
  - Encoding LDPC Codes
  - Practical Code Constructions
- Spatially Coupled LDPC Codes
  - Motivation
  - The Regular Ensemble
  - Density Evolution
  - Burst Erasure

Encoding – Standard Procedure

- In general, LDPC codes can be treated just as any other block code when it comes to encoding
- Generator matrix $\boldsymbol{G}$ must be orthogonal to $\boldsymbol{H}$, i.e. $\boldsymbol{G} \cdot \boldsymbol{H}^T = \boldsymbol{0}$
- If $\boldsymbol{H} = \left(\boldsymbol{P}^T \; \boldsymbol{I}_{n-k}\right)$ then $\boldsymbol{G} = (\boldsymbol{I}_k \; \boldsymbol{P})$
- Using Gauss-Jordan elimination, $\boldsymbol{H}$ can be converted to $\tilde{\boldsymbol{H}} = \left(\boldsymbol{P}^T \; \boldsymbol{I}_{n-k}\right)$ with $\boldsymbol{P}^T$ an $(n-k) \times k$ binary matrix and $\boldsymbol{I}_{n-k}$ the identity matrix of size $(n-k) \times (n-k)$
- The codeword then is $\boldsymbol{x} = \boldsymbol{u}\boldsymbol{G}$

Problems with Standard Encoding

- Obtaining $\boldsymbol{G}$ is a complex process
- $\boldsymbol{G}$ will most likely *not be sparse* as $\boldsymbol{P}^T$ will most likely not be sparse, thus the encoding complexity and the storage requirements will be $O\left(n^2\right)$.

Encoding – (Almost) Time Linear

- Instead of finding a generator matrix $\boldsymbol{G}$ for parity check matrix $\boldsymbol{H}$ an LDPC code can be encoded using full-rank $\boldsymbol{H}$ directly
- $\boldsymbol{H}$ must be transformed into approximate upper triangular form
- Using *only row and column permutations*, we obtain $\boldsymbol{H}'$ from $\boldsymbol{H}$ with

$$\boldsymbol{H}' = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{T} \\ \boldsymbol{C} & \boldsymbol{D} & \boldsymbol{E} \end{pmatrix}$$

  where $\boldsymbol{T}$ is a *lower triangular matrix* of size $(m-g)\times(m-g)$, $\boldsymbol{B}$ is of size $(m-g)\times g$, and $\boldsymbol{A}$ is of size $(m-g)\times k$ (if $\boldsymbol{H}'$ is full rank)
- g is the number of rows left in $\boldsymbol{C}$, $\boldsymbol{D}$ and $\boldsymbol{E}$ and is called *gap* of the approximate representation
- *The smaller g, the lower the encoding complexity!*
- If the permutation operations are well chosen, we have $g \ll m$

Encoding – (Almost) Time Linear (2)

- Starting from $\boldsymbol{H}'$, we can use Gauss-Jordan elimination to clear $\boldsymbol{E}$
- This is equivalent to the multiplication

$$\begin{pmatrix} \boldsymbol{I}_{m-g} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{T}^{-1} & \boldsymbol{I}_g \end{pmatrix} \boldsymbol{H}'$$

which results in

$$\widetilde{\boldsymbol{H}} = \begin{pmatrix} \boldsymbol{I}_{m-g} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{T}^{-1} & \boldsymbol{I}_g \end{pmatrix} \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{T} \\ \boldsymbol{C} & \boldsymbol{D} & \boldsymbol{E} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{T} \\ \widetilde{\boldsymbol{C}} & \widetilde{\boldsymbol{D}} & \boldsymbol{0} \end{pmatrix}$$

with

$$\widetilde{\boldsymbol{C}} = -\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A} + \boldsymbol{C}$$

and

$$\widetilde{\boldsymbol{D}} = -\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{B} + \boldsymbol{D}$$

Encoding – (Almost) Time Linear (3)

- Finally, the codeword $\boldsymbol{x}$ is divided into 3 parts

$$\boldsymbol{x} = (\boldsymbol{u} \;\; \boldsymbol{p}_1 \;\; \boldsymbol{p}_2)$$

where $\boldsymbol{u}$ is the k-bit message, $\boldsymbol{p}_1$ holds the first g parity bits and $\boldsymbol{p}_2$ the remaining $n-k-g$ parity bits

- $\boldsymbol{p}_1$ is then obtained from

$$\boldsymbol{p}_1^T = -\widetilde{\boldsymbol{D}}^{-1}\widetilde{\boldsymbol{C}}\boldsymbol{u}^T$$

- Using back-substitution, $\boldsymbol{p}_2$ can be calculated as

$$\boldsymbol{p}_2^T = -\boldsymbol{T}^{-1}\left(\boldsymbol{A}\boldsymbol{u}^T + \boldsymbol{B}\boldsymbol{p}_1^T\right) = -\boldsymbol{T}^{-1}\left(\boldsymbol{A} - \boldsymbol{B}\widetilde{\boldsymbol{D}}^{-1}\widetilde{\boldsymbol{C}}\right)\boldsymbol{u}^T$$

- Note that for (matrix) operations over $\mathbb{F}_2$, “$+$” and “$-$” are equivalent!

Encoding

Encoding

Encoding - (Almost) Time Linear – Complexity

- Although at first glance, the procedure seems complex, it has some complexity advantages
- Many of the intermediate terms needed in the computation are *sparse* and can be precomputed and stored
- It can be shown that the total complexity is $O(n + g^2)$, which means that when g is very small, it can be neglected and the complexity approaches $O(n)$
- $\widetilde{\boldsymbol{D}}^{-1}$ is dense but only of size $g \times g$ (g small!) and can be precomputed with cost $O(g^3)$ and stored inside the device

- We illustrate the procedure by a small toy example

Encoding – (Almost) Time Linear – Example

- We are given the matrix $\boldsymbol{H}$
- $\boldsymbol{H}$ will be transformed to $\boldsymbol{H}'$ by swapping 2^{nd} and 3^{rd} row as well as 6^{th} and 10^{th} column

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

$$\boldsymbol{H}' = \left(\begin{array}{ccccc|cc|ccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right) \qquad \widetilde{\boldsymbol{H}} = \left(\begin{array}{ccccc|cc|ccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}\right)$$

Encoding – (Almost) Time Linear – Example (2)

- We want to encode $\boldsymbol{u} = (1\,1\,0\,0\,1)$ to $\boldsymbol{x} = (\boldsymbol{u}\,\boldsymbol{p}_1\,\boldsymbol{p}_2)$
- $\boldsymbol{p}_1$ can be calculated by

$$\boldsymbol{p}_1^T = \widetilde{\boldsymbol{D}}^{-1}\widetilde{\boldsymbol{C}}\boldsymbol{u}^T = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

and $\boldsymbol{p}_2$ by

$$\boldsymbol{p}_2^T = \boldsymbol{T}^{-1}\left(\boldsymbol{A}\boldsymbol{u}^T + \boldsymbol{B}\boldsymbol{p}_1^T\right)$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\left[\begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix}\right] = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Encoding

- **Example:**[1] Example of encoding
- **Example:**[2] Example of constructing a parity-check matrix and encoding

[1] **File:** `Encode_LDPC.m`
[2] **File:** `Example_Encoding_and_Constructing.m`