Shuffled Belief Propagation Decoding

Juntan Zhang and Marc Fossorier
Department of Electrical Engineering
University of Hawaii at Manoa
Honolulu, HI 96816
Low-Density Parity Check (LDPC) Codes

- First proposed by R. G. Gallager in 1960’s, and resurrected recently [Gallager-IRE62, MacKay-IT99].
- Can achieve near Shannon limit performance with belief propagation (BP) or sum-product algorithm [Richardson-Urbanke-IT01].
- Advantages over turbo codes:
  - better distance properties;
  - parallel decoding structure for high speed decoders.
- Disadvantages:
  - encoding complexity is high;
  - decoder complexity is high for full parallel implementation.
### Representations of LDPC Codes

**Parity Check Matrix**

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

- **N** is the set of bit (variable) nodes.
- **M** is the set of check nodes.
- **H** is the parity check matrix.

**Bipartite Graph**

- Nodes represent bit (variable) nodes and check nodes.
- Edges connect bit nodes to check nodes.

- **Check nodes** and **Bit (variable) nodes** are connected by edges.
An LDPC code is regular if $H$ has constant row weight and column weight, or equivalently, the check nodes have constant degree $d_c$ and variable nodes have constant degree $d_v$.

An LDPC code is irregular if row and column weights are not constants.

Irregular LDPC codes are defined by degree distributions.

Long irregular LDPC codes have better performance than regular LDPC codes, and can beat turbo codes [Richardson-Urbanke-IT01].
Geometric LDPC Codes

- Originally studied for majority logic decoding decades ago, and constructed based on finite geometries (Euclidean and projective geometries) [Weldon-Bell66, Rudolph-IT67];
- BP algorithm can be applied to the decoding of this family of codes [Lucas-Fossorier-Kou-Lin-COM00, Kou-Lin-Fossorier-IT01];
- Encoding can be easily implemented with shift registers since they are cyclic codes;
- They have very good minimum distance properties;
- Decoding complexity is high.
An example: (7, 3) DSC code:

\[ H = \begin{bmatrix}
1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 \\
\end{bmatrix} \]

- Parity matrix is Squared!
- Not full rank.
- Equal number of bit nodes and check nodes.
- Node degrees are larger.
Some one-step majority decodable codes:

### PG-LDPC codes (DSC)

<table>
<thead>
<tr>
<th>$(N, K)$</th>
<th>rate</th>
<th>$d_{\text{min}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(7, 3)$</td>
<td>0.429</td>
<td>4</td>
</tr>
<tr>
<td>$(21, 11)$</td>
<td>0.524</td>
<td>6</td>
</tr>
<tr>
<td>$(73, 45)$</td>
<td>0.616</td>
<td>10</td>
</tr>
<tr>
<td>$(273, 191)$</td>
<td>0.700</td>
<td>18</td>
</tr>
<tr>
<td>$(1057, 813)$</td>
<td>0.769</td>
<td>34</td>
</tr>
<tr>
<td>$(4161, 3431)$</td>
<td>0.825</td>
<td>66</td>
</tr>
</tbody>
</table>

### EG-LDPC codes

<table>
<thead>
<tr>
<th>$(N, K)$</th>
<th>rate</th>
<th>$d_{\text{min}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(15, 7)$</td>
<td>0.467</td>
<td>5</td>
</tr>
<tr>
<td>$(63, 37)$</td>
<td>0.587</td>
<td>9</td>
</tr>
<tr>
<td>$(255, 175)$</td>
<td>0.686</td>
<td>17</td>
</tr>
<tr>
<td>$(1023, 781)$</td>
<td>0.763</td>
<td>33</td>
</tr>
<tr>
<td>$(4095, 3367)$</td>
<td>0.822</td>
<td>65</td>
</tr>
</tbody>
</table>
Processing in check nodes: Principles:

incoming messages + constraints ⇒ outgoing messages

\[ L_{mn} = 2 \tanh^{-1} \left( \prod_{n' \in N(m) \setminus n} \tanh \left( \frac{z_{mn'}}{2} \right) \right) \]
Processing in bit nodes:

\[ z_{mn} = F_n + \sum_{m' \in M(n) \setminus m} L_{m'n} \]

\[ z_n = F_n + \sum_{m \in M(n)} L_{mn}, \text{ for hard decision} \]
Standard IDBP

- Initialization
- Step 1: Update the Belief Matrix
  - Horizontal Step: Update the whole Check-to-Bit Matrix
  - Vertical Step: Update the whole Bit-to-Check Matrix
- Step 2: Hard decision and stopping test
- Step 3: Output the decoded codeword
Standard BP Algorithm; Step 1:

(i) Horizontal Step:

\[
\epsilon_{mn}^{(i)} = \log \frac{1 + \prod_{n' \in N(m) \setminus n} \tanh(z_{mn'}^{(i-1)} / 2)}{1 - \prod_{n' \in N(m) \setminus n} \tanh(z_{mn'}^{(i-1)} / 2)}
\]

(ii) Vertical Step:

\[
z_{mn}^{(i)} = F_n + \sum_{m' \in M(n) \setminus m} \epsilon_{m'n}^{(i)}
\]

\[
z_n^{(i)} = F_n + \sum_{m \in M(n)} \epsilon_{mn}^{(i)}
\]
Update the belief matrix in i-th iteration with standard belief propagation decoding.
Shuffled Belief Propagation

- Initialization
- Step1: Update the Belief Matrix, for n=0,..N-1
  - Horizontal Step: Update the n-th column of Check-to-Bit Matrix
  - Vertical Step: Update the n-th column of Bit-to-Check Matrix
- Step2: Hard decision and stopping test
- Step3: Output the decoded codeword
Shuffled Belief Propagation; Step 1:

( i ) Horizontal Step:

\[ \varepsilon_{mn}^{(i)} = \log \frac{1 + \prod_{n' \in N(m) \setminus n} \tanh(z_{mn'}^{(i)} / 2) \prod_{n' > n} \tanh(z_{mn'}^{(i-1)} / 2)}{1 - \prod_{n' \in N(m) \setminus n} \tanh(z_{mn'}^{(i)} / 2) \prod_{n' < n} \tanh(z_{mn'}^{(i-1)} / 2)} \]

( ii ) Vertical Step:

\[ z_{mn}^{(i)} = F_n + \sum_{m' \in M(n) \setminus m} \varepsilon_{m'n}^{(i)} \]

\[ z_{n}^{(i)} = F_n + \sum_{m \in M(n)} \varepsilon_{mn}^{(i)} \]
Update the belief matrix in $i$-th iteration with shuffled belief propagation decoding.
Implementation of shuffled BP

- Backward-forward implementation
- Computation Complexity
Optimality and Convergence Property of Shuffled BP

Given the Tanner Graph of the code is connected and acyclic.

- Shuffled BP is optimal in the sense of MAP
- Shuffled BP converges faster (or at least no slower) than Standard BP
Parallel Shuffled BP

- Divide the N bits into G groups, each group contains $N_G$ bits. (regular partition).

- In each group, the updatings are processed in parallel. The processings of groups are in sequential.
Parallel Shuffled Belief Propagation

( i ) Horizontal Step:

\[ \varepsilon_{mn}^{(i)} = \log \frac{1 + \prod_{n' \in N(m) \setminus n, n' \leq g \cdot N_G - 1} \tanh \left( \frac{z_{mn}^{(i)}}{2} \right) \prod_{n' \in N(m) \setminus n, n' \geq g \cdot N_G} \tanh \left( \frac{z_{mn'}^{(i-1)}}{2} \right)}{1 - \prod_{n' \in N(m) \setminus n, n' \leq g \cdot N_G - 1} \tanh \left( \frac{z_{mn}^{(i)}}{2} \right) \prod_{n' \in N(m) \setminus n, n' \geq g \cdot N_G} \tanh \left( \frac{z_{mn'}^{(i-1)}}{2} \right)} \]

( ii ) Vertical Step:

\[ z_{mn}^{(i)} = F_n + \sum_{m' \in M(n) \setminus m} \varepsilon_{m'n}^{(i)} \]

\[ z_{n}^{(i)} = F_n + \sum_{m \in M(n)} \varepsilon_{mn}^{(i)} \]
Update the belief matrix in i-th iteration with Group shuffled BP decoding

\[
\begin{bmatrix}
\mathcal{E}_{00}^{(i)} & \mathcal{E}_{01}^{(i)} & \mathcal{E}_{03}^{(i)} \\
\mathcal{E}_{11}^{(i)} & \mathcal{E}_{12}^{(i)} & \mathcal{E}_{14}^{(i)} \\
\mathcal{E}_{22}^{(i)} & \mathcal{E}_{23}^{(i)} & \mathcal{E}_{25}^{(i)} \\
\mathcal{E}_{33}^{(i)} & \mathcal{E}_{34}^{(i)} & \mathcal{E}_{36}^{(i)} \\
\mathcal{E}_{44}^{(i)} & \mathcal{E}_{45}^{(i)} & \mathcal{E}_{46}^{(i)} \\
\mathcal{E}_{55}^{(i)} & \mathcal{E}_{56}^{(i)} & \mathcal{E}_{66}^{(i)}
\end{bmatrix}
\]
A Small Example: LDPC (6,2) code

Parity Check Matrix

\[ H = \begin{bmatrix}
0 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
\end{bmatrix} \]

Tanner Graph
Decoding with Standard BP

Iteration 4

\[ \text{bit} \leftarrow \text{check} \]

\[ SUY \quad \rightarrow \quad SUY \]

\[ SUY \quad \rightarrow \quad SUY \]

\[ SUY \quad \rightarrow \quad SUY \]

\[ SUY \quad \rightarrow \quad SUY \]

\[ SUY \quad \rightarrow \quad SUY \]

\[ \text{Converge} \]
Decoding with Shuffled BP

Iteration 2

\[ \begin{align*}
    & \text{bit} \rightarrow \text{check} \\
    & \begin{array}{c}
        SUY \\
        SUY \\
        SUY \\
        SUY \\
        SUY \\
    \end{array} \rightarrow \begin{array}{c}
        SUY \\
        SUY \\
        SUY \\
        SUY \\
    \end{array} \\
    & \text{Converge}
\end{align*} \]
Decoding with Group Shuffled BP ($G = 2$)

Iteration 3

Bit $\rightarrow$ Check

$SUY \rightarrow SUY$

$SUY \rightarrow SUY$

$SUY \rightarrow SUY$

$SUY \rightarrow SUY$

$SUY \rightarrow SUY$

$SUY \rightarrow SUY$

Converge
Comparison of Speed of Convergence

- Standard BP
  \[ I^{G=1} = \begin{bmatrix} 2 & 3 & 3 & 4 & 3 & 4 \end{bmatrix} \]

- Shuffled BP
  \[ I^{G=6} = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix} \]

- Group Shuffled BP
  \[ I^{G=2} = \begin{bmatrix} 2 & 3 & 3 & 3 & 2 & 3 \end{bmatrix} \]
Pe of LDPC (8000,4000)(3,6) code with shuffled and standard BP
Average Number of Iterations
Pe of LDPC(8000,4000)(3,6) with Group Shuffled BP decoding
Average Number of Iterations
Conclusion

- Shuffled BP achieves a good trade-off between performance and complexity.
- Group shuffled BP can decrease decoding delay and is suitable for hardware implementation.