Random Construction을 이용한 LDPC 부호의 성능 분석

2013 한국 통신학회 동계학술대회
2013/01/31

전석민, 김인선, 박진수, 박기현, 송홍엽
Contents

1. Introduction
2. Random Permutation
3. Generation of Parity-Check Matrix
4. Simulation Result
5. Conclusions
Introduction

- LDPC codes are represented by check and variable node connection

![Tanner graph of LDPC codes]

- Degree Distribution determines the number of edges of Tanner graph
  - Choosing good degree distribution is important due to good performance
    \[ \lambda(x): \text{check node degree distribution} \quad \rho(x): \text{variable node distribution} \]

- There are two kinds degree distribution of LDPC code
  - Regular LDPC code, Irregular LPDC code
Introduction

- LDPC codes construction
  - PEG, ACE, Random ...

Good BER performance can be obtained from some random parity check matrix with sufficient length (Richardson 2001)

Suggest that simple permutation method & parity check matrix construction

Fig. 2 Tanner graph representation of parity-check matrix
Introduction

Research objectives

- **Find sufficient codeword length** which can achieve good BER performance
- **Verify the performance** of randomly constructed regular and irregular codes of lengths around 100, 500, 1000, 2000, and 4000.
- BER performance of the codes is compared with that of some PEG-optimized codes in 802.11n standard and 802.16 standard.
- **Result Analysis** why they are a bit worse.
### Random Permutation

<table>
<thead>
<tr>
<th>PRNG1</th>
<th>( PRNG(n) = \text{rand}()^{\lfloor \text{abs}(\sin(n) \times 16777216) \rfloor \bmod 32768} ) ^:Bit-wise XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRNG2</td>
<td>( \text{prem}[i] = \lfloor M \times \sin[n \times x] \rfloor )</td>
</tr>
<tr>
<td>Quadratic hash</td>
<td>( x(i + 1) = [x(i) + 2 \times i + 1] \pmod{m} ) ( x(i + 1) = [x(i) + t] \pmod{m} )</td>
</tr>
<tr>
<td>Primitive root</td>
<td>The primitive root is defined by ( p_1 ) ( p_1^i = n \pmod{m} )</td>
</tr>
</tbody>
</table>
Generation of Random Permutation

1. Permutation generated using the non-periodicity of the sine function (PRNG1)

- Generation length m permutation define perm[j]=j (j=1 to m), i=1, n=1

\[
\text{PRNG1}(n) = \text{rand()} \otimes (\text{floor}(\text{abs}(\sin(n) \times 16777216))) \mod 32768
\]

- \(i++, n++\)

- Swap (perm[i], perm[i + \{prng1[n] mod m-i\}])

- \(i < m ?\)

Example

1. Gen permutation 1 to 9

We assume Prng1(n)=4

\[5 2 3 4 1 6 7 8 9\]

Swap (perm[1],perm[1+4])

We assume Prng1(n)=5

\[5 7 3 4 1 6 2 8 9\]

Swap (perm[2],perm[2+5])

\[5 2 3 4 1 6 7 8 9\]

...
2. Permutation generated using the non-periodicity of the \( \sin \) function (PRNG2)

Generation length \( m \) permutation

Define \( \text{perm}[j] = 0 \) (\( j = 1 \) to \( m \)), \( i = 1, x = 1 \)

Parameter \( M, N \) randomly generated

\[
\text{PRNG2}(n) = \lfloor M \sin (N + x) \rfloor \mod m
\]

\( x++ \)

\( i < m \) ?

\( \text{perm}[\text{PRNG2}(n)] = i, i++ \)

\( \text{perm}[\text{PRNG2}(n)] = 0 \) ?

\( \text{END} \)

**Example**

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Generate \( \text{perm}[j] = 0 \) (\( j = 1 \) to \( m \))

\( \text{perm}[1] = 5 \)

| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

\( \text{perm}[2] = 3 \)

| 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

\( \text{perm}[3] = 5 \)

Regeneration \( \text{PRNG2}(n) \)

...
### 3. Quadratic hash permutation

**Generation length m permutation**
- Define perm[j]=0 (j=1 to m), i=0,
- Select x(1) & perm[x(1)]=1

\[ x(i+1) = [ \text{x}(i) + (2 \cdot i + 1) ] \mod m + 1 \]

**Example**
- \( X(1) = 3 \)
- \( X(2) = (3 + 2 \cdot 1 + 1) \mod 9 \)
- \( X(3) = (6 + 2 \cdot 2 + 1) \mod 9 \)

...
4. Primitive root permutation

Select prime number \( p \) which is greater than \( m \)

Primitive root is obtained from \( p \) & select one primitive root defined by \( p_1 \)

Generation permutation: 
\[ \text{Perm}[i] = p_1^i \mod m \]

If \( \text{perm}[i] \) is greater than \( m \), \( \text{perm}[i] \) is removed

**Example**

Select number 3 which is primitive of 5

- \( 3 \)
- \( 4 = (3 \times 3) \mod 5 \)
- \( 2 = (4 \times 3) \mod 5 \)
- \( 1 = (2 \times 3) \mod 5 \)

- \( 2 \)
- \( 4 = (2 \times 2) \mod 5 \)
- \( 3 = (4 \times 2) \mod 5 \)
- \( 1 = (3 \times 2) \mod 5 \)
Method of generation of parity check matrix by some permutation

Example

(2,4) regular LDPC code

permutation={1,5,7,9,2,3,6,11,4,8,10,12}
Generation of Parity-Check Matrix

- **2-Cycle problem**
  - Two lines from one check node can connect to one variable node
  - If the problem appears, we regenerated the permutation or exchanged the line for the line of next node

```
Permutation={1, 3, 4, 2, 5, 7, ...}
```

```
Permutation={1, 3, 5, 2, 4, 7, ...}
```

2-cycle problem
Simulation Settings

- Regular LDPC codes of the proposed method are compared with codeword length 100, 500, 1000, 2000 (3,6), (4,8) regular PEG-optimized LDPC code.

- Irregular LDPC codes of the proposed method are compared with PEG-optimized and 802.11n standard LDPC code which length is 648, 1296, 2284, 4568 in several codes on the standard (Rate:0.5)

- Degree distribution of irregular LDPC code is

\[ \lambda(x) = 0.2558x + 0.3140x^2 + 0.0465x^3 + 0.3837x^{10} \]
\[ \rho(x) = 0.8140x^6 + 0.1860x^7 \]

<table>
<thead>
<tr>
<th>variable node</th>
<th>check node</th>
</tr>
</thead>
<tbody>
<tr>
<td># degree 2</td>
<td># degree 3</td>
</tr>
<tr>
<td>594</td>
<td>486</td>
</tr>
</tbody>
</table>

Length 1296 802.16n LDPC standard code degree
Simulation Settings

- Irregular LDPC codes of the proposed method are compared with PEG-optimized and 802.16 standard LDPC code which length is 576, 1056, 2016 in several codes on the standard. (Rate: 0.5)

- Degree distribution of irregular LDPC code is

\[
\lambda(x) = 0.4583x + 0.3333x^2 + 0.2083x^3 \\
\rho(x) = 0.6667x^6 + 0.3333x^7
\]

<table>
<thead>
<tr>
<th># degree 2</th>
<th># degree 3</th>
<th># degree 6</th>
<th># degree 7</th>
<th># degree 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>484</td>
<td>352</td>
<td>220</td>
<td>352</td>
<td>176</td>
</tr>
</tbody>
</table>

Length 1056, 802.16n LDPC standard code degree
Simulation Result

(3,6) Regular LDPC Codes – Length 100, 500, 1000, 2000

- Length = 100
- Length = 500
- Length = 1000
- Length = 2000
Simulation Result

(4,8) Regular LDPC Codes – Length 100, 500, 1000, 2000
Simulation Result

- 802.11n Standard LDPC Codes (Irregular) – Length 648, 1296, 2284, 4568

Graphs showing BER vs. Eb/N0 (dB) for different lengths of LDPC codes.
Simulation Result

802.16 Standard LDPC Codes – Length 576, 1056, 2016

- Length = 576
- Length = 1056
- Length = 2016
Simulation Result

- (3,6) & (4,8) Regular LDPC Codes Simulation Result

- BER performance of (3,6) Regular LDPC Codes

- BER performance of (4,8) Regular LDPC Codes
Simulation Result

- 802.11n & 802.16 Irregular LDPC Codes Simulation Result
  
  BER performance of 802.11n Irregular LDPC Codes
  
  BER performance of 802.16 Irregular LDPC Codes
Conclusions

- We can achieve good BER performance by using random permutation in waterfall region without some optimization process.
- In (3,6) regular code, the proposed codes has same BER performance as PEG optimized code at codeword length 2000.
- In (4,8) regular code, the proposed codes has better performance than PEG optimized code at codeword length 100.
- The BER simulation results show that the SNR loss between proposed method and standard LDPC code in irregular LDPC code.