현대부호이론
Project # 2

20212245 김 희서

Ⅰ. introduction

[1] 1962년 Gallager에 의해서 처음 제안된 LDPC(Low Density Parity Check)부호는 Parity Check Matrix의 원소들의 대부분이 0인 linear block code로서, turbo code처럼 샤넌의 channel capacity limit에 가까운 성능을 보입니다. 제안 당시에는 LDPC 부호 생성의 복잡성 때문에 많은 주목을 받지 못했지만, 기술의 발전에 따라 딥러닝 및 고성능 컴퓨터의 개발, 효율적인 LDPC 알고리즘이 개발되어 현재 활발히 연구가 되고 있습니다.

우수한 LDPC code 생성 조건 중 girth를 최대화 하는 방법이 있는데, Xiao-Yu Hu와 Dieter M.Arnold는 PEG(Progressive Edge-Growth)알고리즘을 제안하였습니다.

Ⅱ. PEG와 IPEG

2.1 PEG(Progressive edge-growth algorithm)

성능이 좋은 LDPC code를 생성하기 위해서는 큰 girth를 갖는 테너 그래프를 구성해야하는 데, 이를 PEG 알고리즘으로 수행합니다.

- [2]의 논문에서 저자는 'Progressive edge-growth (PEG) algorithm'이라고 불리는 'edge-by-edge' 방식으로 symbol과 check nodes 사이의 연결과 edge를 설정함으로써 큰 둘레(girth)를 가진 테너 그래프를 구성하는 방법을 제안하였습니다.

- 알고리즘의 핵심 아이디어는 현재 심벌 노드 s_j에서 가장 먼 거리에 있는 Check Node(CN)를 찾은 후, 해당 CN를 연결하여 새로운 edge를 배치하는 것입니다.

그리고 CN 집합 중에서 차수(degree)가 가장 작은 CN를 선택하고, 만약 최소 degree를 가진 CN가 2개 이상 존재한다면 임의로 한 CN를 선택합니다.

- 이를 통해, Parity Check Matrix의 Small cycle 생성을 억제하고 local girth 값을 최대화할 수 있습니다.

- symbol nodes, check nodes, symbol - node - degree sequence of the graph를 고려할 때, 그래프에 새 edge를 배치하는 것은 가능한 한 girth에 영향을 미치지 않도록 edge를 선택하는 절차부터 시작합니다.

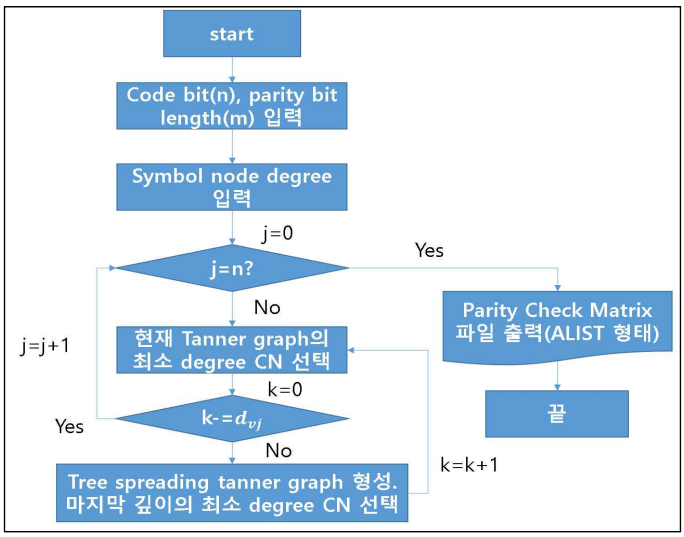
- 즉, 최적의 edge를 결정 후, 새 edge를 가진 graph는 업데이트되고, 다음 edge의 배치 절차가 진행됩니다.

게다가 girth에 대한 하, 상한과 최소 거리에 대한 하한은 기본적인 PEG Tanner graph(underlying PEG Tanner graph)의 파라미터 관점에서 유도됩니다.

- 해당 논문의 시뮬레이션 결과를 보면, PEG 알고리즘은 짧고 moderate 한 block length의 regular, irregular LDPC코드를 생성하는 강력한 알고리즘임을 알 수 있고, 다른 기존의 구조와 비교하면, PEG 알고리즘은 단순함과 어떤 주어진 block에 대해 우수한 코드를 성공적으로 생성하는 flexibility의 성질을 갖고 있습니다.

Progressive Edge-Growth Algorithm :

for j =0 to n-1 **do**
begin
 for k=0 to d_{s_j}-1 **do**
 begin
  if k=0
   $E_{s_j}^0$ <- edge(c_i,s_j), where $E_{s_j}^0$ is the first edge incident to s_j and c_i is a check node such that it has the lowest check-node degree under the current graph setting $E_{s_0} \cup E_{s_1} \cup \cdots \cup E_{s_{j-1}}$.
  else
   expand a subgraph from symbol node s_j up to depth l under the current graph setting such that the cardinality of $N_{s_j}^l$. stops increasing but it less than m, or $\overline{N}_{s_j}^l \neq \varnothing$ but $\overline{N}_{s_j}^{l+1} = \varnothing$, then $E_{s_j}^0$ <- edge(c_i,s_j), where $E_{s_j}^k$ is the kth edge incident to s_j and c_i is a check node picked from the set $\overline{N}_{s_j}^l$ having the lowest check-node degree.
 end
end



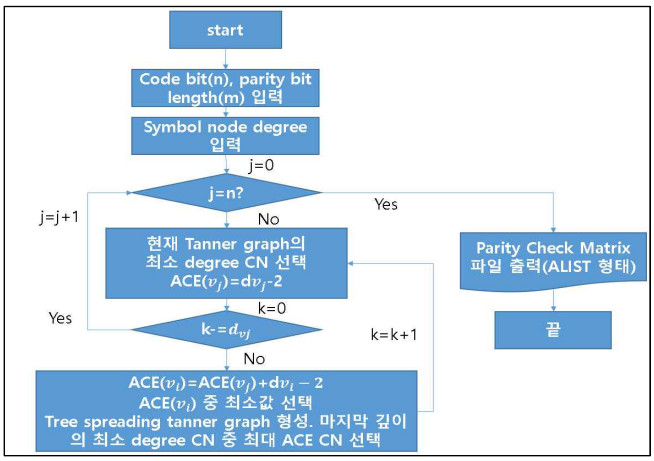
PEG 알고리즘 순서도

2.2 IPEG 알고리즘

- PEG알고리즘의 error floor현상을 개선한 알고리즘으로 ACE(Approximate Cycle Extrinsic message 차수) 개념을 도입하여 PEG 알고리즘에서 발생하는 임의 선택

조건을 개선시킨 알고리즘.

- ACE는 특정 노드를 포함하고 있는 순환 노드에서 연결 된 외부 순환 노드와의 연관성을 의미하고, ACE가 클수록 외부 정보를 받고 내보내기 쉽기에, 정보가 순환 노드 안에서 고립되지 않고, 이를 통해 SPA decoding 과정에서 오류율을 낮추는 효과가 있습니다.



IPEG 알고리즘 순서도

PEG 알고리즘과 IPEG의 차이점은 'PEG'는 멀리 있는 CN의 차수만 고려하지만, IPEG알고리즘은 최소 차수를 가진 CN의 ACE 값이 가장 큰 CN를 선택하여, IPEG 알고리즘은 PEG 알고리즘보다 cycle 간의 연결성이 높아져, 성능이 향상됩니다.

Ⅲ. 시뮬레이션

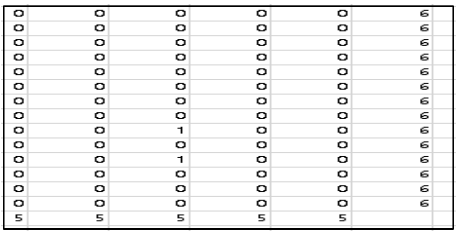
시뮬레이션의 조건은 다음과 같습니다.

1) AWGN channels are assumed.
2) The degrees of variable and check nodes are regular.
3) The number of iterations is 10, 20, and 50, respectively.
4) The lengths of LDPC codes are 600, 1200, and 2400.

다음은 LDPC code의 length에 따른 값들을 나타낸 것으로 600 일 때는 500*600이 되는 데, "degree of variable nodes : 5, degree of check nodes : 6"의 조건에 의해서 '$1200*5 = x*6$'이기에, x값이 1000이 되어, '1000*1200'이 되었습니다.

마찬가지 이유로, length가 2400일 때는 '$2400*5 = 6*x$'이기에, x값이 2000이 되어, '2000*2400'이 되었습니다.

5) Code parameters are given as follows:
degree of variable nodes : 5, degree of check nodes : 6

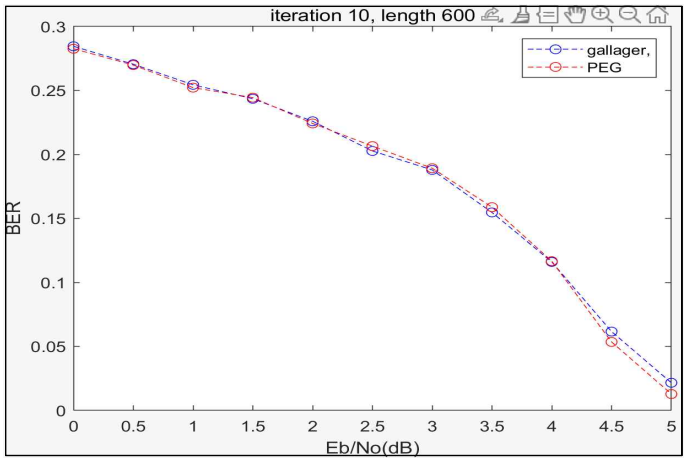


"degree of variable nodes : 5, degree of check nodes : 6"의 조건을 갖춘 code를 만들었습니다.

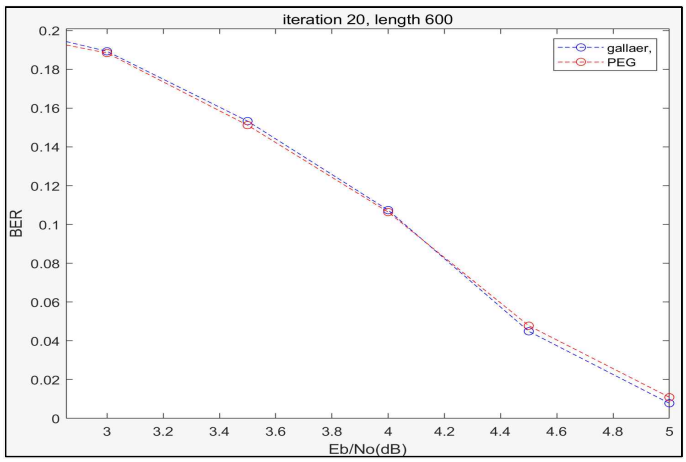
다음의 사진은 excel의 "countif"를 통해, 행 열의 '1'의 개수를 나타내는 함수로, 해당 조건에 맞게 variable node의 degree는 5, check node의 degree는 6이 나왔습니다.

다음과 같이 각 행, 열마다 degree가 같기 때문에, regular하다는 것도 확인할 수 있습니다.

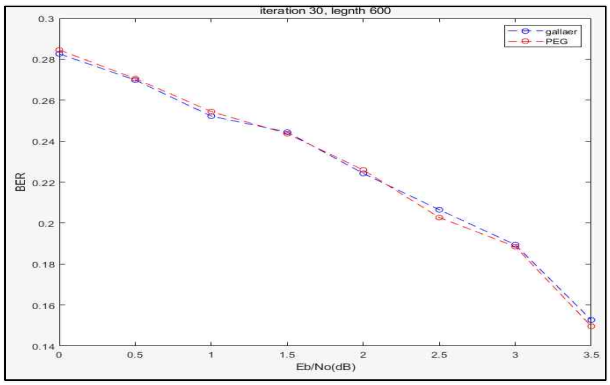
3.1 length 600



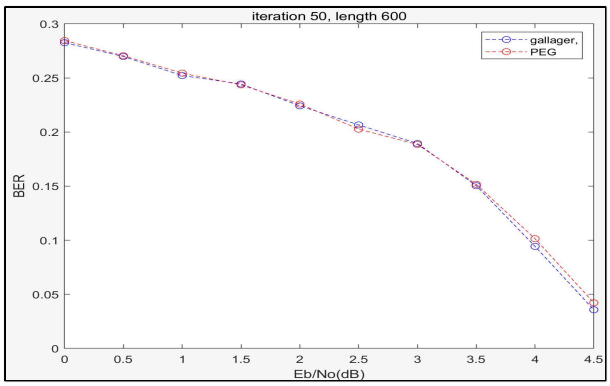
iteration 10, length 600



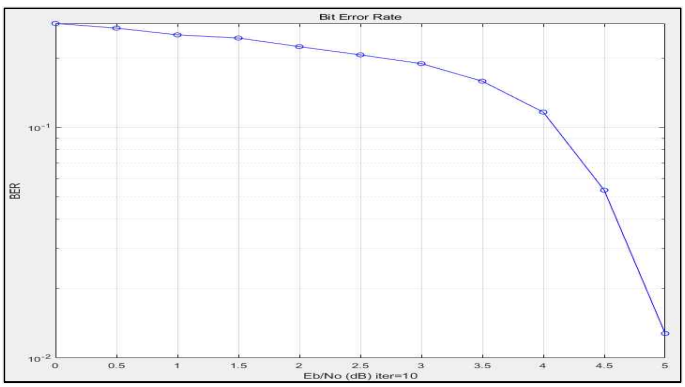
iteration 20, length 600



iteration : 30



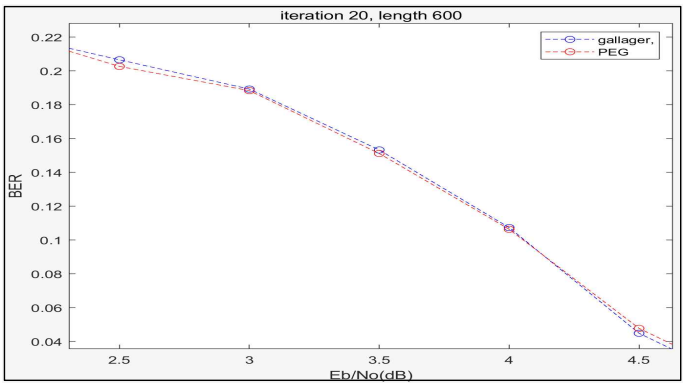
iteration 50, length 600



dB=[0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0];

iteration 10, length 600 : gallager

iteration '10, 20, 30, 50, 100'일 때의 시뮬레이션의 결과 값이고, gallager LDPC로 만들어진 값과 PEG의 결과 값을 비교하였습니다.

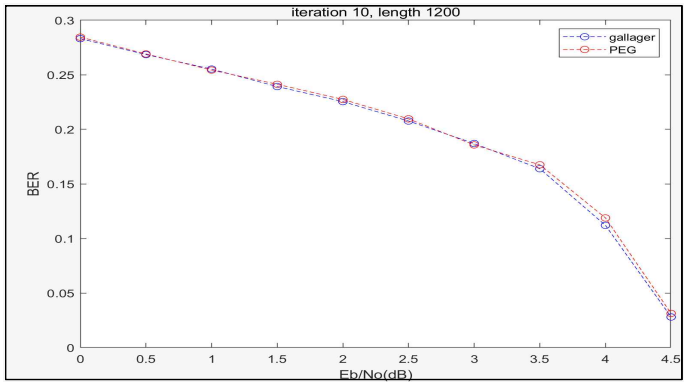


- iteration 20일 때 결과 값을 확대한 것으로, 빨간색의 PEG값이 파란색인 gallager 값보다 성능이 좋다는 것을 확인할 수 있습니다. BER의 값은 low dB에서 둘의 성능 차이가 적지만, 4dB와 같이 high dB로 갈수록 이 둘의 성능 차이가 나는 것을 확인하였습니다.

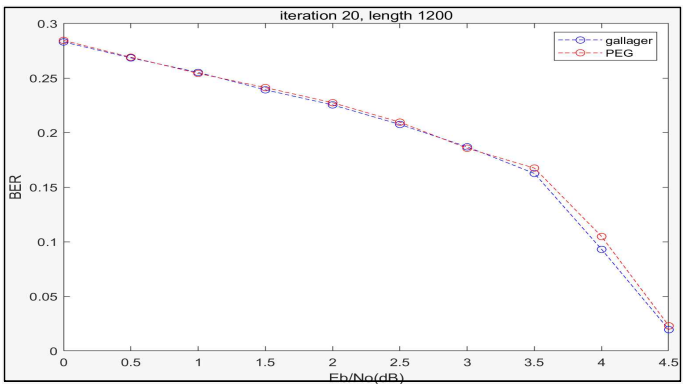
이때 결과 값이 눈에 확 보이도록 설정하기 위해서는 dB 값을 iteration 10일 때처럼, 최소 5dB이상일 때의 값을 출력하면 더 확 눈에 띄는 결과 값이 보였겠지만, 시뮬레이션의 시간이 생각보다 너무 오래 걸려, 3.5dB의 값만 출력하였습니다.

결과적으로 length 600일 때, PEG의 성능이 gallager의 성능보다 좋았습니다.

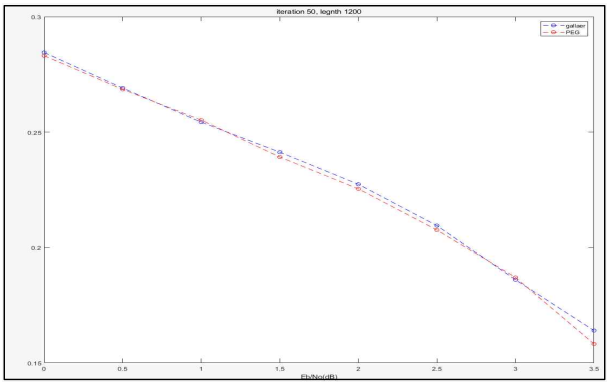
3.2 length 1200



iteration : 10



iteration : 20



iteration : 50

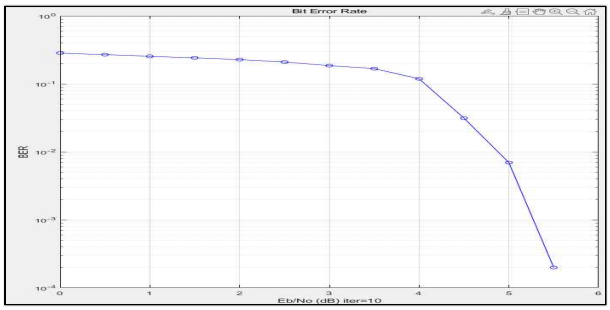
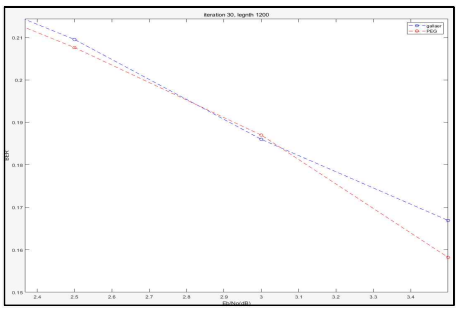
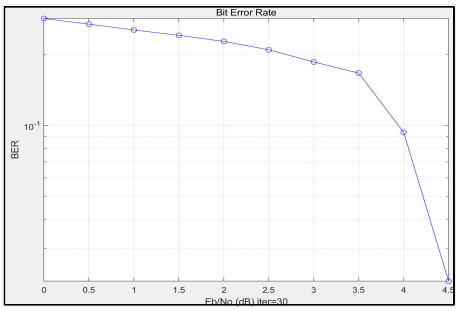


Fig 1. dB=[0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5];
iteration 10, length 1200 : gallager



iteration 30일 때 결과 값을 확대한 것으로, 빨간색의 PEG값이 파란색인 gallager 값보다 성능이 좋다는 것을 확인할 수 있습니다.

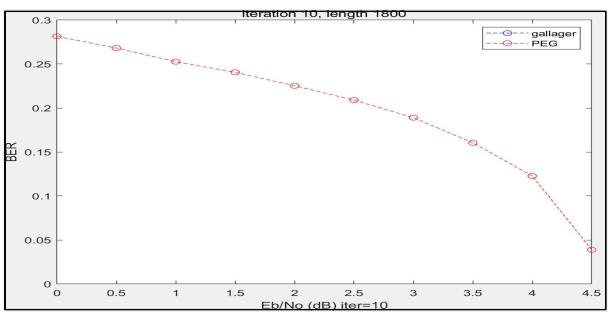


PEG, iteration 30, length : 1200

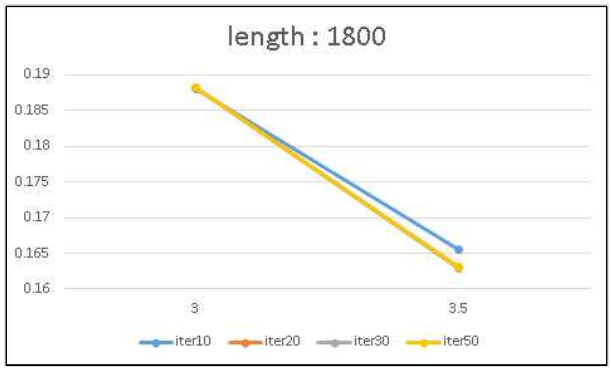
그리고 BER의 값은 low dB에서는 둘의 성능 차이가 적지만, 3, 3.5dB와 같이 high dB로 갈수록 이 둘의 성능 차이가 나는 것을 확인하였습니다.

Fig 1.의 값을 보면, dB의 값의 범위를 넓히면 확실히 성능의 변화가 눈에 띄는 것을 확인할 수 있는데, 앞서 언급한 것처럼 시뮬레이션 시간에 의해, 다음과 같이 좁은 면적을 수행하였습니다. 그러나 iteration 10일 때 범위를 넓혀서 확인한 결과 PEG의 성능이 좋았습니다.

3.3 length 1800

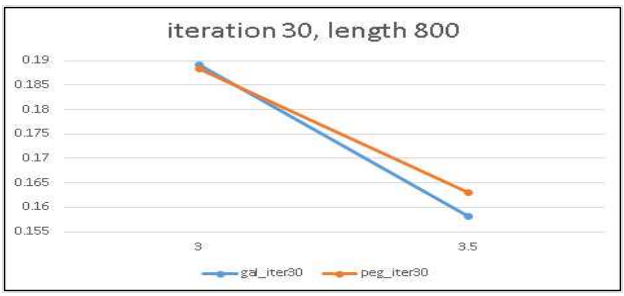


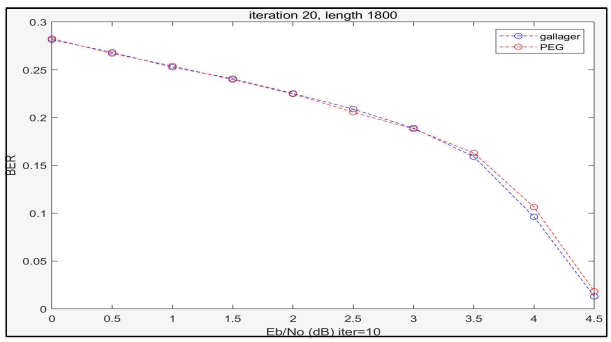
iteration 10, length 1800



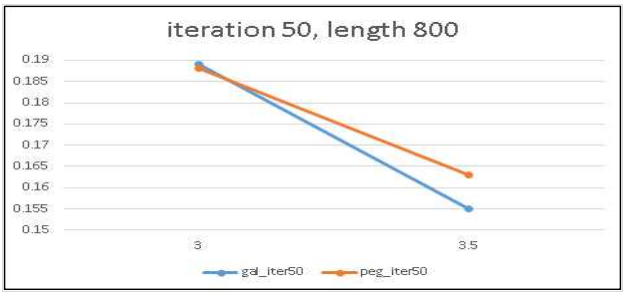
다음은 length에 따라 결과 값이 달라지는 것을 보기 위해서, '1800'을 추가로 시뮬레이션 한 것입니다. 즉, 1200과 2400사이의 변화 추이를 보기 위해서 진행하였습니다.

다음의 결과 값을 통해, code length가 1800일 때, iteration에 따른 결과 값을 확인했습니다. iteration이 많을수록 성능이 좋지만, 시간이 오래 걸렸습니다.



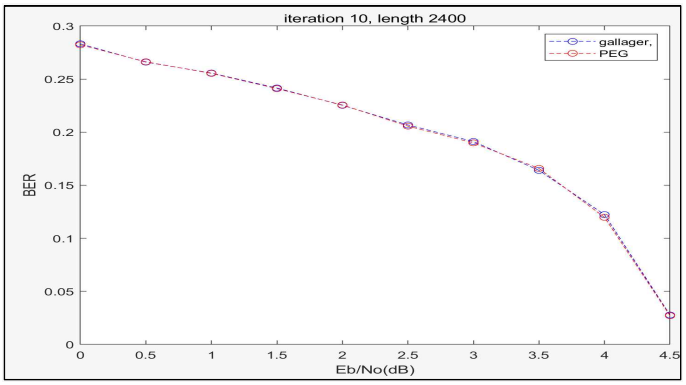


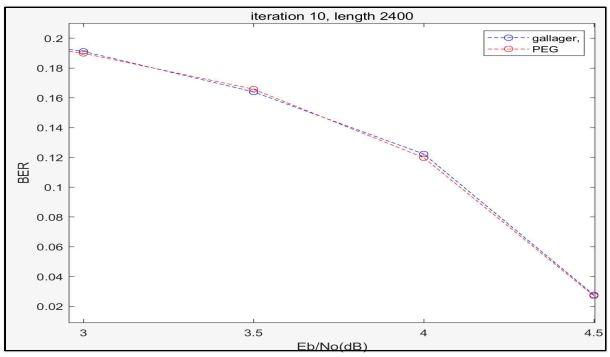
iteration 20, length 1800



- 다음은 length 800이고, iteration의 값이 30, 50일 때의 결과 값을 비교한 것으로, 3dB를 기준으로 gal_iteration50의 값이 더 좋은 성능을 보여주었습니다. 해당 경우도, dB값의 범위를 크게 할수록 PEG의 결과 값이 더 좋아집니다.

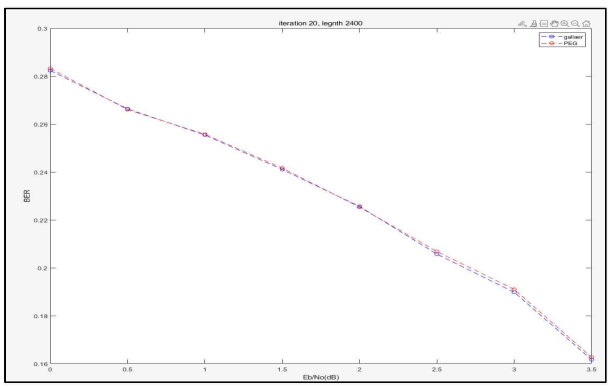
3.4 length 2400



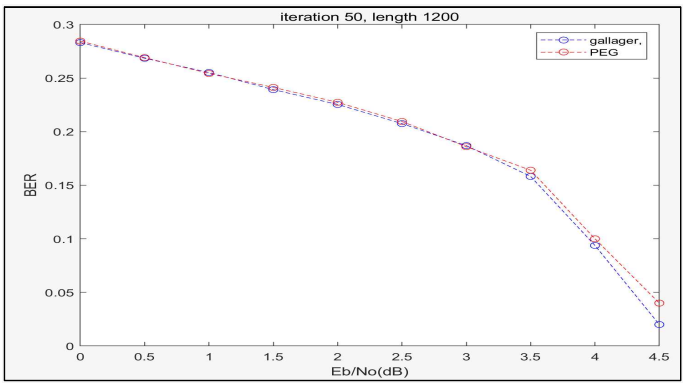


dB : 3 3.5 4 4.5(dB), iteration : 10

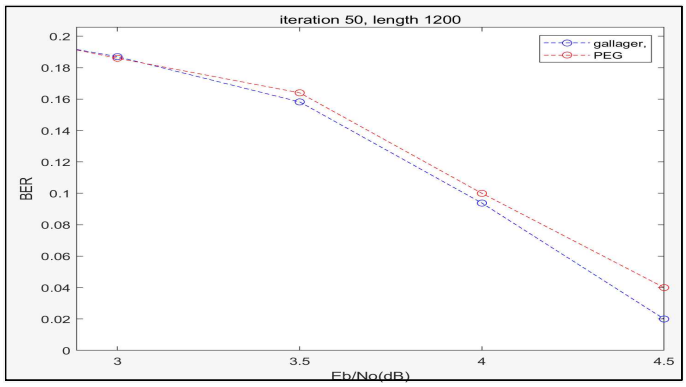
iteration의 값이 적을 때는 BER의 성능이 gallager와 PEG가 유사한 것을 확인할 수 있었습니다.



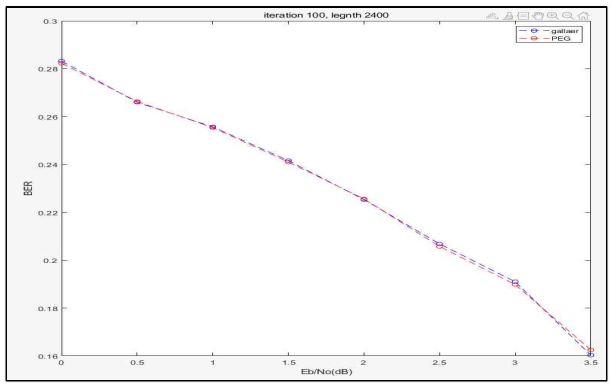
iteration : 20



3dB를 기준으로 PEG의 성능이 gallager의 코드보다 성능이 안 좋은 것을 그래프를 통해 확인했습니다.



dB=[3.0 3.5 4.0 4.5], iteration 50, length 1200

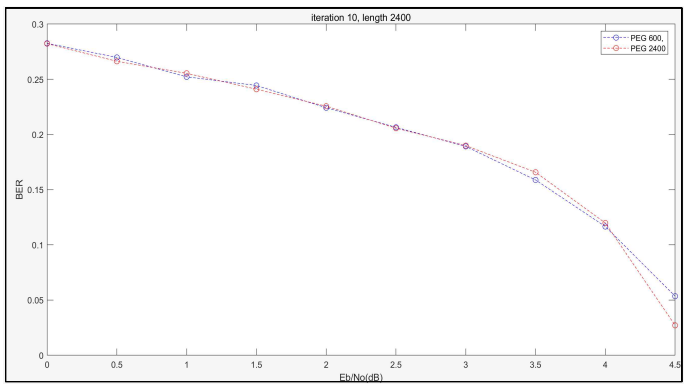


iteration : 100

다음은 iteration 50 일 때의 결과 값을 확대한 것으로, 3dB를 기준으로 성능이 gallager가 PEG보다 좋아진다는 것을 확인할 수 있습니다. 그러나 이것은 일시적일 뿐, dB의 범위를 넓히면, PEG의 값이 더 좋은 성능을 갖는다는 것을 알 수 있습니다.

3.5 PEG length

마지막은 동일한 iteration일 때, PEG의 길이에 따른 결과 값을 비교하였습니다.



PEG length :600 vs PEG length :2400, iteration 10

- PEG 알고리즘은 짧고 moderate한 block length의 regular, irregular LDPC 코드를 생성하는 강력한 알고리즘이기에, 코드가 길어지면서 PEG가 성능의 우수성이 줄어든다는 것을 확인하기 위해서, 600, 2400일 때의 결과 값을 비교하였습니다.

그런데 만일 length가 2400일 때, 성능이 더 좋지 않았다면, short block length에 효과적인 알고리즘이라는 것을 확인할 수 있었겠지만, 4dB를 기준으로 'PEG length 2400'이 성능이 더 좋은 값을 나타냈기에, 코드가 길고 짧다는 것은 2400을 기준으로 확인할 수 있지 않다는 것을 알 수 있었습니다. 해당 조건을 확인하기 위해서는 더 긴 length로 비교해야 할 것 같습니다.

3.5 ProgressiveEdgeGrowthACE 실행 법

function H = ProgressiveEdgeGrowthACE(N, M, Dv, DoYouWantACE)

- ProgressiveEdgeGrowthACE(1200, 1000,'123.csv',0)를 Matlab을 통해서 실행하는 것이고, 1200과 1000은 H의 '1000*1200'의 값을 나타냅니다.

D	E	F	G	H	I	J
5	5	5	5	5	5	5

D_v의 값은 variable degree sequence를 나타내는 데, 이때 해당 조건이 '5'이기 때문에, '123.csv' 파일은 다음과 같이 구성하였습니다.

Ⅳ. 결론

시뮬레이션은 gallager LDPC와 PEG의 성능을 비교함으로써, PEG의 성능의 우수성을 확인할 수 있었습니다.

PEG의 경우, 짧고 moderate한 block length의 regular, irregular LDPC코드를 생성하는 강력한 알고리즘이라는 것에 알맞게, 길이가 짧은 600일 때와 달리, 길이가 길어질수록, low dB와 달리 high dB일 때, gallager가 PEG보다 성능이 좋아지는 것을 확인하려고 시뮬레이션을 했습니다. (길이가 길어지면, gallager > PEG)

그 결과, 길이가 길고 짧다는 기준이 600, 2400이상의 범위라는 것을 확인할 수 있었습니다. 따라서 주어진 조건인 '600, 1200, 2400'으로 길이가 길다 짧다는 것을 판단할 수 없어, 해당 조건을 확인하기 위해서는 더 긴 길이가 필요하다는 것을 확인할 수 있었습니다.

그리고 길이와 iteration을 바꿔가면서 결과 값을 비교하였는데, iteration이 많아질수록 성능이 좋아지지만, 시뮬레이션 시간이 기하급수적으로 늘어나는 것 또한 확인할 수 있었고, 만일 더 좋은 결과 값을 그래프 적으로 표현하기 위해서는 dB의 범위를 최소 5dB 정도는 늘렸어야 했는데, 생각 이상의 시뮬레이션 시간이 필요했기 때문에, 이를 하지 못한 것이 아쉬웠던 시뮬레이션 프로젝트였습니다.

참고 문헌

[1] 이문호, 박중용, "[특집/기술해설]4세대 이동통신을 위한 LDPC 원리 및 응용"

[2] X.-Y. Hu, E. Eleftheriou, D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," IEEE Trans. Information Theory, vol. 51, no. 1, pp. 386-398, January 2005.

[3] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, "Progressive edge-growth Tanner graphs," in Proc. IEEE Global Telecommunications Conf., San Antonio, TX, Nov. 2001. CD Proceedings, paper, no. 0-7803-7208-5/01.