# Introduction to Progressive Edge-Growth (PEG) Algorithm for LDPC Codes


Citra Yasin Akbar Fadhlika

The Center for Advanced Wireless Technologies (AdWiTech), Telkom University,
Jl. Telekomunikasi No. 1, Terusan Buah Batu, Bandung, 40257 Indonesia.
E-mail: {*citrayaf@student.*}*telkomuniversity.ac.id*

ZEMI
Bandung, Indonesia

# Outline

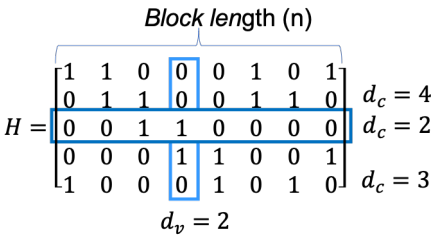
# Motivation and Problem

- Perancangan matriks *parity check* $H$ LDPC *codes* tidak bisa dilakukan secara *random*, oleh karena itu teknik perancangan matriks $H$ LDPC *codes* harus menghasilkan *girth* besar.
- Nilai *girth* kecil akan memperburuk kinerja dari *iterative decoding* LDPC codes.[1]

[1] M. Sipser and D. A. Spielman, "Expander codes," in IEEE Transactions on Information Theory, vol. 42, no. 6, pp. 1710-1722, Nov. 1996.

# Low Density Parity Check (LDPC) Codes

- LDPC *codes* terbentuk oleh sebuah matriks yang sebagian besar elemennya bernilai "$0$" dan hanya beberapa yang bernilai "$1$".[1]
- LDPC *codes* memiliki kinerja yang baik pada panjang blok $n$ mendekati tak hingga dan cenderung memiliki kinerja yang lebih buruk pada $n$ yang terbatas.[2]
- Matriks *parity check* $H$
  - ✓ Panjang blok $n$
  - ✓ *Variable node degree* $d_v$
  - ✓ *Check node degree* $d_c$

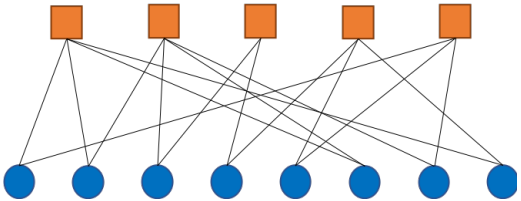



*Parity Check Matrix of Random Irregular LDPC Codes.*

[1] R. Gallager, "Low-density parity-check codes," in IRE Transactions on Information Theory, vol. 8, no. 1, pp. 21-28, January 1962.

[2] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," in IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 599-618, Feb 2001.

# Tanner Graph



- Matriks *parity check* LDPC *codes* $H$ dapat direpresentasikan menggunakan Tanner *graph* dengan dua set *node*.[1]
- Garis (*edge*) antara *variable nodes* dengan *check nodes* terhubung apabila

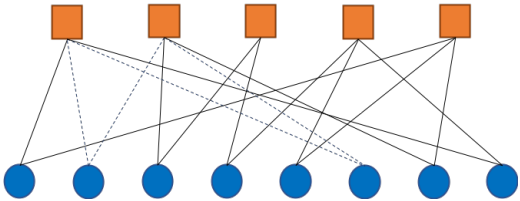
$$H_{i,j} = 1, \tag{1}$$

dengan $i$ adalah *variable node* ke-$i$ dan $j$ adalah *check node* ke-$j$.

[1] R. Tanner, "A recursive approach to low complexity codes," in IEEE Transactions on Information Theory, vol. 27, no. 5, pp. 533-547, September 1981.

# *Girth*

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



- *Girth* adalah panjang siklus terpendek dalam sebuah Tanner *graph* yang menjadi hal penting dalam menentukan kinerja LDPC *codes*.[1]
- *Girth* lokal pun terbentuk dari siklus per *node* pada sebuah Tanner *graph* LDPC *codes*.
- Terjadinya *girth* kecil akan memberikan efek buruk pada LDPC codes, karena mengurangi kinerja dari *extrinsic information* dalam proses *iterative decoding*.[2]

[1] J. Fan and Y. Xiao, "A Method of Counting the Number of Cycles in LDPC Codes," 2006 8th international Conference on Signal Processing, Beijing, 2006, pp.

[2] M. Sipser and D. A. Spielman, "Expander codes," in IEEE Transactions on Information Theory, vol. 42, no. 6, pp. 1710-1722, Nov. 1996.

# Progresive Edge-Growth (PEG) Algorithm (1/4)[1]

- PEG adalah salah satu metode untuk membuat LDPC *codes* berdasarkan dari Tanner *graph* dengan memperhatikan penyebaran *edge* yang terhubung pada setiap *node* untuk menghasilkan *girth* besar.
- Pembuatan LDPC codes dengan menggunakan PEG dapat dibuat berdasarkan jumlah *variable nodes* atau *block length* $n$, check node $m$, *variable node degree* (VND), dan *check node degree* (CND).
- PEG melakukan penyebaran *edge* menggunakan graf pohon dengan memperhatikan *degree* setiap *node* yang telah ditentukan dan setiap *node*-nya hanya dapat muncul sekali.
- Kedalaman dari graf pohon akan disimbolkan dengan $l$. $l$ akan mempengaruhi lokal *girth* yang terbentuk.

[1] Xiao-Yu Hu, E. Eleftheriou and D. -. Arnold, "Progressive edge-growth Tanner graphs," GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270), San Antonio, TX, 2001, pp. 995-1001 vol.2.

# Progresive Edge-Growth (PEG) Algorithm (2/4)

Secara sederhana proses pembuatan LDPC *codes* menggunakan algoritma PEG adalah sebagai berikut:

1. Tempatkan elemen 1 mulai dari kolom ke-1 sampai ke-$n$, penempatan ini memperhatikan baris atau *check node* dengan *degree* terkecil.
2. Menyebarkan *edge* ke node yang belum terhubung dengan memperhatikan *degree* dari *check node*.

*Cycle* dari LDPC *codes* yang terbentuk menggunakan PEG dapat dipastikan akan memenuhi

$$2(l+2). \tag{2}$$

# Progresive Edge-Growth (PEG) Algorithm (3/4)[1]

Batas nilai *girth* $(g)$ pada LDPC *codes* yang dirancang menggunakan algoritma PEG dapat diketahui melalui persamaan

$$t = \frac{\log\left(md_c^{max} - \frac{md_c^{max}}{d_v^{max}} - m + 1\right)}{\log\left[\left(d_v^{max} - 1\right)\left(d_c^{max} - 1\right)\right]} - 1, \tag{3}$$

$$g \geq 2\left(\lfloor t \rfloor + 2\right), \tag{4}$$

dengan nilai $d_v$, $d_c$, dan $m$ yang telah ditentukan. Dari persamaan (3) dan (4) maka

$$\frac{d_v^{max}\left[\left(d_v^{max} - 1\right)^{t+1}\left(d_c^{max} - 1\right)^{t+1} - 1\right]}{\left(d_v^{max} - 1\right)\left(d_c^{max} - 1\right) - 1} < m, \tag{5}$$

sehingga dapat diketahui batas minimal jumlah *check node* agar *girth* yang ingin dihindari dapat terpenuhi.

[1] Xiao-Yu Hu, E. Eleftheriou and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," in IEEE Transactions on Information Theory, vol. 51, no. 1, pp. 386-398, Jan. 2005.

# Progresive Edge-Growth (PEG) Algorithm (4/4)

PEG memiliki dua metode dalam proses pembuatan LDPC *codes* dengan cara:

1. Secara acak memilih *check nodes* terkecil yang ditemukan.
2. Selalu memilih *check nodes* terkecil yang ditemukan sesuai dengan urutannya $c_1, c_2, c_3, \cdots, c_m$, dengan $m$ adalah jumlah baris.

Hal ini mengakibatkan LDPC *codes* akan memiliki matriks *parity check* berbeda-beda, apabila menggunakan algoritma PEG. Pada awalnya PEG menggunakan metode pertama[1], sedangkan pada penelitian ini mengusulkan dengan menggunakan metode kedua.

[1] Xiao-Yu Hu, E. Eleftheriou and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," in IEEE Transactions on Information Theory, vol. 51, no. 1, pp. 386-398, Jan. 2005.

# The Proposed PEG Algorithm for LDPC Codes

- Algoritma PEG yang diusulkan menggunakan metode kedua dan ditambahkan sebuah algoritma untuk menghindari pembentukkan LDPC *codes* dengan *girth*-4.
- Penggunaan PEG dalam pembuatan matriks *parity check* LDPC *codes* memungkinkan untuk membuat matriks *parity check* LDPC *codes* sesuai dengan keinginan.
- Nilai panjang blok $n$, baris $m$, dan set dari VND $D_v = \{d_{v_1}, d_{v_2}, d_{v_3}, \cdots, d_{v_n}\}$ dapat ditentukan, dengan $d_v$ adalah VND.
- Proses penempatan elemen 1 dimulai dari kolom 1 sampai $n$ dan dari baris 1 ke $m$ yang prosesnya berjalan dari kiri ke kanan dan dari atas ke bawah.

F. A. Newagy and S. H. Elramly, "Novel Technique for Scaling Down LDPC Code Lengths in DVB-T2 Standard," in 2012 International Conference on Telecommunications and Multimedia (TEMU), July 2012, pp. 180–184.

# The Proposed Anti Girth-4 Algorithm (1/5)

- Algoritma anti *girth*-4 ini ditambahkan pada iterasi kedua dalam penempatan elemen satu di setiap kolom. Langkah-langkah perancangan matriks dan pengecekannya sebagai berikut:

$$H = \overbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}^{n} \Big\} m$$

1. Buat matriks nol dengan dimensi $m \times n$, dengan $n$ adalah jumlah variable *nodes* dan $m$ adalah jumlah *check nodes* yang telah ditentukan.

# The Proposed Anti Girth-4 Algorithm (2/5)

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

2. Elemen 1 pertama ditempatkan sesuai dengan algoritma pertama dari proses PEG. Elemen 1 kedua dan seterusnya sebanyak $d_v(n)$ ditempatkan menggunakan kombinasi algoritma PEG dengan *Anti Girth*-4.

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

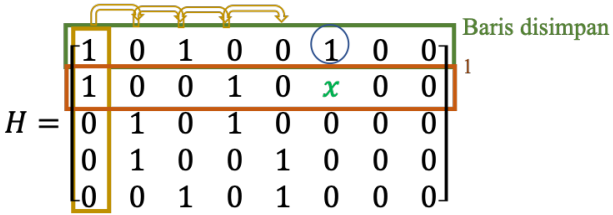
3. Elemen 1 ditempatkan pada *check node* urutan pertama dengan nilai CND terendah.

# The Proposed Anti Girth-4 Algorithm (3/5)

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

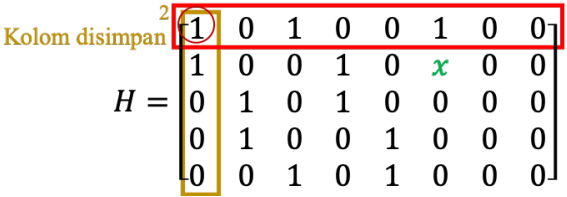
4. Simbol $x$ adalah calon letak elemen 1, $x$ dipilih berdasarkan baris yang memiliki nilai CND terendah. Kemudian langkah berikutnya menggunakan algoritma *Anti Girth*-4, melakukan pengeceken ke kiri apabila tidak ada elemen satu maka letak tersebut akan menjadi elemen 1.

# The Proposed Anti Girth-4 Algorithm (4/5)




5. Algoritma *Anti Girth*-4 dimulai dari iterasi ke-1, baris dari elemen 1 sebelumnya disimpan yang nantinya akan digunakan pada pengecekan akhir. Kemudian melakukan pengecekan dari kiri ke kanan apabila ditemukan elemen 1, maka akan berlanjut ke iterasi berikutnya. Apabila tidak ditemukan elemen 1 pada kolom tersebut, maka $x$ akan menjadi elemen 1. Pengecekan akan dilakukan sampai kolom sebelum kolom dari $x$.

# The Proposed Anti Girth-4 Algorithm (5/5)



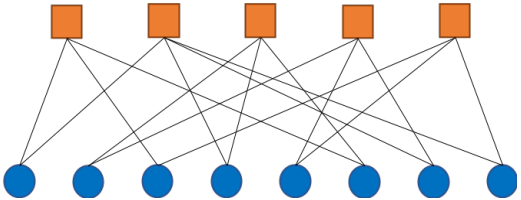

5. Iterasi ke-2 Algoritma *Anti Girth*-4, menyimpan kolom dari elemen 1 yang ditemukan pada proses pengecekan di iterasi ke-1.
6 Kemudian melakukan pengecekan akhir, yaitu apabila

$$H_{Baris\_simpan,Kolom\_simpan} = 1, \tag{6}$$

pada $x$ akan terbentuk *girth* 4 jika $x$ berelemen 1, sehingga $x$ akan berpindah ke *check node* dengan CND minimal berikutnya dan melakukan pengecekan lagi.

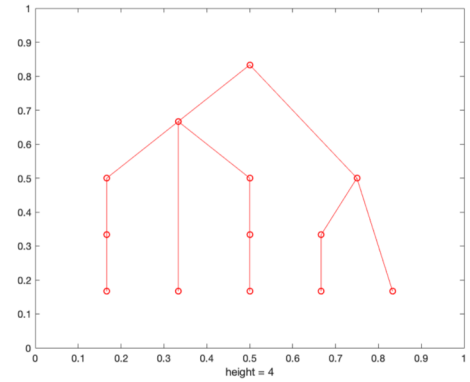
# The LDPC Codes Constructed Using PEG Algortihm

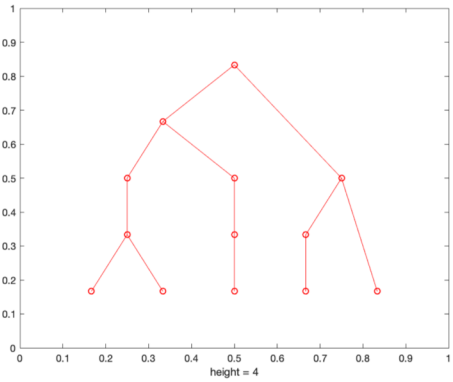
$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$



- Matriks *parity check* LDPC *codes* $H$, dengan VND setiap kolomnya 2 yang dibentuk menggunakan PEG.
- Matriks *parity check* yang terbentuk tidak memiliki *girth*-4.

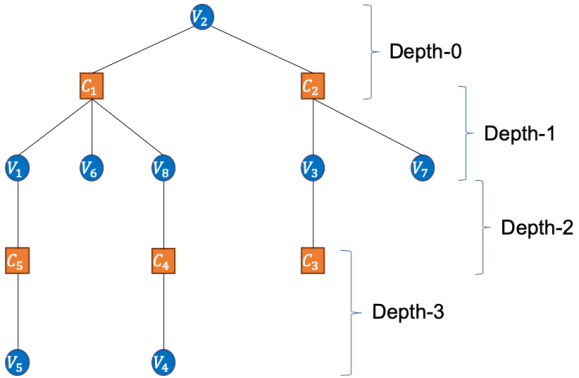
# Tree Graph of LDPC Codes




- Graf pohon matriks *parity check random* LDPC *codes* dan matriks *parity check* LDPC *codes* yang dirancang menggunakan PEG pada *variable node* ke-dua menggunakan MATLAB.

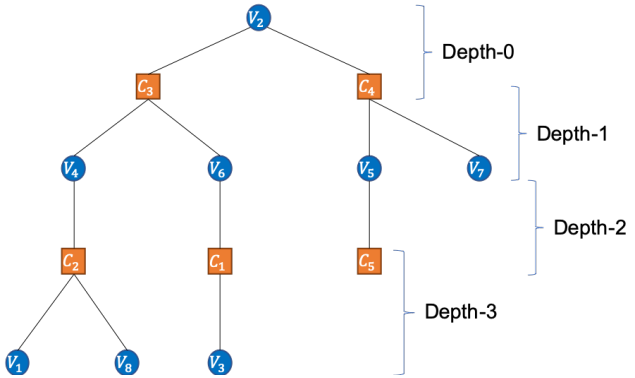
# Tree Graph of Random LDPC Codes




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

- Graf pohon dari matriks *parity check random* LDPC *codes* pada *variable node* ke-dua memiliki nilai depth $l = 3$.

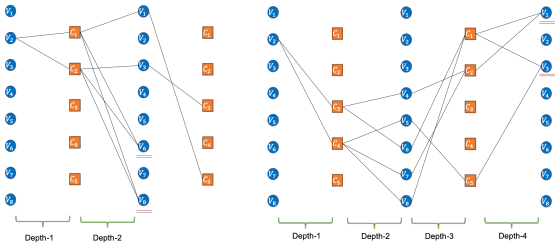
# Tree Graph of LDPC Codes using PEG




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

- Graf pohon dari matriks *parity check* LDPC *codes* yang dirancang menggunakan PEG pada *variable node* ke-dua memiliki nilai depth $l = 3$.

# Proposed Girth Calculation Method



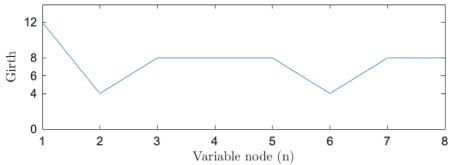

*Local girth* dapat dihitung dengan menggunakan teknik pembuatan graf pohon PEG yang dimodifikasi. Berikut langkahnya:

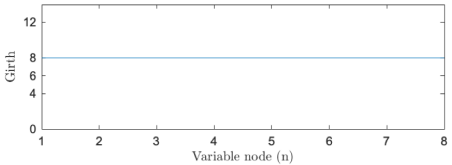
1. Tentukan satu *variable node* dan sebarkan *edges*-nya sesuai dari matriks.
2. Sebarkan *edges* terus menerus sampai terdapat *node* yang tidak dapat menyebarkan *edge*.
3. Hitung *branch* $b$ dimulai dari satu, lalu dengan menggunakan persamaan

$$g = b \times 2, \tag{7}$$

maka nilai *local girth* dapat diketahui.

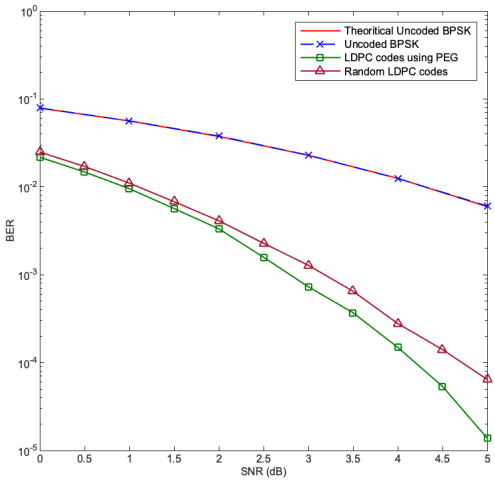
# Girth Distributions




- Hasil penghitungan *girth* dari matriks *parity check random* LDPC *codes* dan LDPC *codes* yang dirancang menggunakan algoritma PEG dengan nilai VND yang sama.

# Performance Evaluation on AWGN Channel




- Simulasi menggunakan iterasi pada LDPC *codes* sebanyak $t = 80$ dan menggunakan modulasi BPSK.
- LDPC *codes* yang dirancang menggunakan PEG memiliki kinerja yang lebih baik dibandingkan *random* LDPC *codes* dengan ukuran matriks dan VND yang sama.

# Conclusion

- Telah melakukan perancangan matriks *parity check* LDPC *codes* dengan menggunakan algoritma PEG yang digabungkan dengan algoritma Anti *Girth*-4 .
- Mengusulkan sebuah algoritma untuk menghindari *girth*-4 pada matriks *parity check* LDPC *codes* dan teknik untuk menghitung *girth* dari *matriks parity check* LDPC *codes*.
- Algoritma PEG memudahkan dalam memodifikasi dan merancang LDPC *codes* yang memiliki nilai *girth* besar.
- Metode kedua dari PEG pada LDPC codes akan cenderung menyamakan CND dari matriks parity *check codes* yang terbentuk.