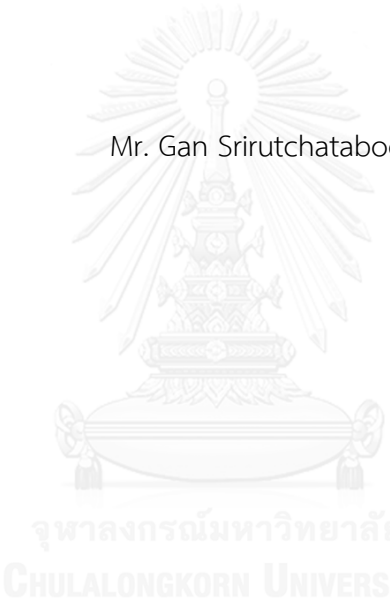
การออกแบบเมทริกซ์พาริตีเช็กของรหัสแอลดีพีซี

นายกานต์ ศรีรัชตบูรณ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2557

DESIGN OF A PARITY-CHECK MATRIX FOR LDPC CODES

Mr. Gan Srirutchataboon



A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering Program in Electrical Engineering

Department of Electrical Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2014

หัวข้อวิทยานิพนธ์	การออกแบบเมทริกซ์พาริตีเช็กของรหัสแอลดีพีซี
โดย	นายกานต์ ศรีรัชตบูรณ์
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.ลัญฉกร วุฒิสิทธิกุลกิจ
อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม	รองศาสตราจารย์ ดร.ปิยะ โควินท์ทวีวัฒน์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.บัณฑิต เอื้ออาภรณ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.พสุ แก้วปลั่ง)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.ลัญฉกร วุฒิสิทธิกุลกิจ)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม
(รองศาสตราจารย์ ดร.ปิยะ โควินท์ทวีวัฒน์)

.....กรรมการ
(ผู้ช่วยศาสตราจารย์ สุวิทย์ นาคพีระยุทธ)

.....กรรมการภายนอกมหาวิทยาลัย
(ดร.พิสิฐ วนิชชานันท์)

กานต์ ศรีรัชตบูรณ์ : การออกแบบเมทริกซ์พาริตีเช็กของรหัสแอลดีพีซี (DESIGN OF A PARITY-CHECK MATRIX FOR LDPC CODES) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: รศ. ดร.ลัญฉกร วุฒิสิทธิกุลกิจ, อ.ที่ปรึกษาวิทยานิพนธ์ร่วม: รศ. ดร.ปิยะ โควินท์ทวีวัฒน์, 80 หน้า.

ปัจจุบันความต้องการความน่าเชื่อถือของข้อมูลและระบบความปลอดภัยเพิ่มขึ้นอย่างมากจากในอดีตสำหรับระบบการติดต่อสื่อสารแบบดิจิตอล ซึ่งความผิดพลาดที่เกิดจากข้อมูลนั้นเกิดขึ้นจากความไม่เป็นอุดมคติของช่องสัญญาณ ดังนั้นการพัฒนาปรับปรุงทฤษฎีช่องสัญญาณให้มีประสิทธิภาพมากยิ่งขึ้นจึงเป็นการแก้ไขปัญหาของความต้องการดังกล่าว วิทยานิพนธ์ฉบับนี้ได้เสนอวิธีการสร้างเมทริกซ์พาริตีของรหัสแอลดีพีซีอันเป็นรหัสช่องสัญญาณที่ได้รับความนิยมมากในปัจจุบันเนื่องจากมีความสามารถในการแก้ไขข้อผิดพลาดของข้อมูลได้สูงรหัสหนึ่ง สำหรับการออกแบบเมทริกซ์พาริตีนั้นมุ่งเน้นไปยังการทำให้เมทริกซ์พาริตีมีเกิร์ตสูงหรือสามารถที่จะกำหนดขนาดของเกิร์ตในเมทริกซ์ได้เหตุเพราะพารามิเตอร์เกิร์ตนี้เป็นที่ยอมรับกันในหมู่นักวิจัยว่าสามารถทำให้เมทริกซ์พาริตีมีประสิทธิภาพที่ดีได้หากขนาดของเกิร์ตมีขนาดสูง

วิธีการแรกที่นำเสนอคือวิธีการสร้างเมทริกซ์พาริตีแบบที่เรียกว่า QC-cyclic หรือ “QC-cyclic LDPC codes” ซึ่งเมทริกซ์พาริตีชนิดนี้มีความน่าสนใจคือสามารถที่จะทำงานร่วมกับระบบฮาร์ดแวร์ได้ดีรวมทั้งยังถูกใช้ในหลายมาตรฐานของการติดต่อสื่อสารในปัจจุบัน โดยวิธีการที่นำเสนอนั้นคือการหาค่าเกิร์ตที่สูงที่สุดของคู่ของตัวเลขใด ๆ ที่ใช้สำหรับการหมุนของเมทริกซ์ย่อยของคู่ของบล็อกแถวเพื่อที่จะนำไปใช้สำหรับหมุนเมทริกซ์ในเมทริกซ์พาริตีเพื่อให้มีขนาดของเกิร์ตที่สูงที่สุดโดยอาศัยอัลกอริทึมการค้นหา เมื่อเปรียบเทียบขนาดของเกิร์ตของเมทริกซ์ที่มาจากอัลกอริทึมที่นำเสนอกับอัลกอริทึมอย่าง modified array code (MAC), Sidara-Fuja-Tanner (SFT) และ Magic square ผลลัพธ์ที่ได้นั้นปรากฏว่าเมทริกซ์ที่มาจากอัลกอริทึมที่นำเสนอนั้นมีขนาดของเกิร์ตที่สูงกว่ารวมไปถึงเมื่อเปรียบเทียบประสิทธิภาพในเทอมของ BER เมทริกซ์ที่มาจากอัลกอริทึมที่นำเสนอนั้นก็ยังคงให้ประสิทธิภาพที่เหนือกว่าเมทริกซ์ชนิดอื่น

วิธีการที่สองที่นำเสนอคือวิธีการสร้างเมทริกซ์พาริตีที่มีน้ำหนักของทุก ๆ หลักเท่ากับ 2 ซึ่งเมทริกซ์พาริตีชนิดนี้มีความซับซ้อนที่น้อยกว่าเมทริกซ์ที่มีน้ำหนักของหลักสูงกว่ารวมทั้งยังมีความเป็นไปได้ที่สูงที่จะให้ประสิทธิภาพที่ดีเมื่อถูกนำไปใช้กับระบบช่องสัญญาณแบบวนกลับยิ่งไปกว่านั้นยังถูกนำไปใช้กับรหัสแอลดีพีซีแบบนอนไบนารีที่ขนาดเล็กถึงขนาดกลางอีกด้วย โดยวิธีการที่นำเสนอนั้นเป็นกระบวนการการใส่โนดบิตและโนดพาริตีลงในกราฟแทนเนอร์ขนาดเล็กเพื่อสร้างให้เกิดกราฟแทนเนอร์ขนาดที่ต้องการโดยที่สามารถทำให้เกิร์ตทั้งหมดที่มีอยู่ในกราฟนั้นไม่ต่ำกว่าที่กำหนดไว้ อย่างไรก็ตามเมื่อนำเมทริกซ์ที่มาจากอัลกอริทึมที่นำเสนอมาเปรียบเทียบขนาดเกิร์ตกับอัลกอริทึมที่มีชื่อเสียงอย่าง progressive edge growth (PEG) และ Bit filling ที่มิติเดียวกันปรากฏว่าเมทริกซ์ที่มาจากอัลกอริทึมที่นำเสนอสามารถให้ขนาดของเกิร์ตได้สูงกว่าและเมื่อเปรียบเทียบประสิทธิภาพของเมทริกซ์ที่มาจากอัลกอริทึมที่นำเสนอกับเมทริกซ์ที่สร้างจากอัลกอริทึมอื่นที่กล่าวข้างต้นในเทอมของ BER เมทริกซ์ที่มาจากอัลกอริทึมที่นำเสนอสามารถให้ประสิทธิภาพที่เหนือกว่าทั้ง PEG และ Bit filling

จากการทดสอบประสิทธิภาพภานนั้นวิทยานิพนธ์ฉบับนี้แสดงให้เห็นถึงผลกระทบของเกิร์ตที่มีต่อประสิทธิภาพของรหัสแอลดีพีซี กล่าวคือเมื่อเมทริกซ์พาริตีมีเกิร์ตขนาดที่สูงแล้วจะให้ประสิทธิภาพที่ดีกว่าเมทริกซ์พาริตีที่มีเกิร์ตต่ำกว่า นอกจากนี้นอกจากนี้ยังมีเทคนิคที่ใช้ลดความซับซ้อนของอัลกอริทึม PEG เสนออยู่ในวิทยานิพนธ์ฉบับนี้ด้วย

ภาควิชา วิศวกรรมไฟฟ้า	ลายมือชื่อนิสิต .....
สาขาวิชา วิศวกรรมไฟฟ้า	ลายมือชื่อ อ.ที่ปรึกษาหลัก .....
ปีการศึกษา 2557	ลายมือชื่อ อ.ที่ปรึกษาร่วม .....

5470121521 : MAJOR ELECTRICAL ENGINEERING
KEYWORDS: LDPC, GIRTH, QC-CYCLIC ,COLUMN 80

GAN SRIRUTCHATABOON: DESIGN OF A PARITY-CHECK MATRIX FOR LDPC CODES. ADVISOR: ASSOC. PROF. LUNCHAKORN WUTTISITTIKULKIJ, Ph.D., CO-ADVISOR: ASSOC. PROF. PIYA KOVINTAVEWAT, Ph.D., pp.

The demand for secure and reliable communication is increasing which give rises to the evolution of modern and sophisticated digital communication systems. Most of the errors in the communication systems are introduced by the communication channel, and error correcting codes can be used to overcome this issue, thus bringing reliability in the communication link. In this thesis, the author has proposed two algorithms for constructing a parity check matrix (H) of two classes of LDPC code and the performance of these LDPC codes perform better than the existing ones. While designing such codes, special focus has been given to maximize and can predetermine the girth of H matrix.

The first proposed algorithm is about constructing H matrix for " Quasi-Cyclic low-density parity-check (QC-LDPC)". Such type of codes is used in many number of applications as it is suitable for hardware implementations. In the proposed algorithm, the H matrix of QC-LDPC is constructed by sequentially maximizing the local girth for each block column of the matrix. The performance of the proposed algorithm is compared with other existing algorithms, such as modified array code (MAC), Sidara-Fuja-Tanner (SFT) and Magic square, and the results show that the proposed algorithm performs better than the existing ones in terms of bit-error rate and provides a higher girth.

Another algorithm is proposed for constructing the H matrix with column weight 2, and it has the capability to determine the length of the girth. Such type of codes is less complex than other weights and has good potential performance in partial response channel. Moreover, it can also be used to design non-binary LDPC codes with small or moderate code lengths. The proposed algorithm carefully adds the bit and the check nodes in the Tanner graph while keeping the length of the girth until it fully expands to the required Tanner graph. However, the *H* matrix of the proposed algorithm can provide a higher girth when compared with progressive edge growth (PEG) and Bit-filling algorithms at same code length. The simulation results show that the proposed algorithm can also yield a better BER performance.

In addition, we also show the effective of the girth parameter in the H matrix, which implies that if the length of the girth is large, it can yield a better performance in terms of BER when compared with the H matrix with a lower girth length. Furthermore, this thesis also introduces a technique to improve complexity of PEG algorithm, which can be used to construct the H matrix faster than the original one and provides the same H matrix

Department:	Electrical Engineering	Student's Signature	.....
Field of Study:	Electrical Engineering	Advisor's Signature	.....
Academic Year:	2014	Co-Advisor's Signature	.....

กิตติกรรมประกาศ

ขอขอบคุณทุนสนับสนุนการวิจัยทุน 90ปี จุฬาลงกรณ์มหาวิทยาลัย สำหรับเงินสนับสนุนอุปกรณ์คอมพิวเตอร์สำหรับวัดประสิทธิภาพการทดลอง

สารบัญ

หน้า

หน้า

หน้า

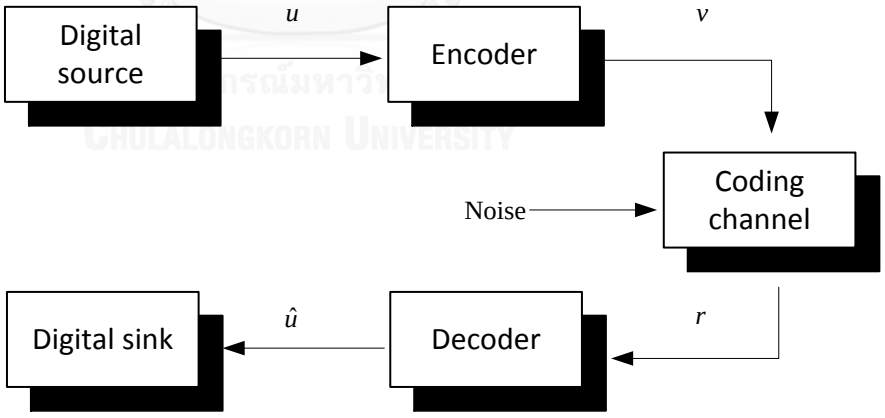
สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความสำคัญของรหัสแอลดีพีซี

ในระบบการสื่อสาร ณ ปัจจุบันนั้นมีความต้องการประสิทธิภาพและความน่าเชื่อถือของข้อมูลที่สูงขึ้นมากเมื่อเทียบกับอดีตที่ผ่านมา ไม่ว่าจะเป็นการส่งข้อมูลแบบมีสายหรือแบบไร้สาย ในปกติของระบบการสื่อสารแบบสัญญาณดิจิทัลจะมีความไม่อุดมคติเกิดขึ้นอยู่เสมออันเนื่องมากจากสัญญาณรบกวนที่เกิดขึ้นระหว่างภาครับ และภาคส่ง ด้วยสาเหตุนี้เองการส่งผ่านข้อมูลในระบบสื่อสารจึงต้องมีการผ่านกระบวนการเข้ารหัสช่องสัญญาณ (channel coding) ณ ภาคส่ง และทำการถอดรหัส ณ ที่ภาครับเพื่อแก้ไขข้อผิดพลาดที่เกิดขึ้นจากความไม่อุดมคติของช่องสัญญาณ จากรูปที่ 1.1 แสดงแบบจำลองของระบบสื่อสารที่อาศัยรหัสช่องสัญญาณ โดยมีคำอธิบายดังนี้ เมื่อสัญญาณดิจิทัลถูกปล่อยออกมายังภาคส่งก็จะมีการเข้ารหัสช่องสัญญาณก่อนถูกส่งไปยังช่องสัญญาณ จากความไม่เป็นอุดมคติของช่องสัญญาณจากสัญญาณรบกวน (Noise) ทำให้สัญญาณดิจิทัลนั้นผิดเพี้ยนไป แต่เมื่อสัญญาณไปถึงยังภาครับนั้นจะมีการถอดรหัสเพื่อแก้ไขสัญญาณดิจิทัลให้กลับมาถูกต้องให้ได้มากที่สุด



รูปที่ 1.1 รูปแบบจำลองของระบบสื่อสารที่อาศัยรหัสช่องสัญญาณ

ในวิทยานิพนธ์ฉบับนี้ได้มุ่งเน้นไปที่การพัฒนาเมทริกซ์พาริตี (เมทริกซ์ $\mathbf{H}$) ซึ่งเป็นเมทริกซ์ที่ใช้ถอดรหัสของรหัสพาริตีความหนาแน่นต่ำ (Low Density Parity Check codes : LDPC codes) หรือที่เรียกกันโดยย่อว่ารหัสแอลดีพีซี รหัสชนิดนี้ได้รับความนิยมอย่างสูงในปัจจุบัน รวมทั้งยังถูก

คาดหมายจะว่าถูกใช้อย่างแพร่หลายในมาตรฐานของระบบการสื่อสารในยุคหน้าเพราะรหัสชนิดนี้มีขีดสามารถแก้ไขข้อผิดพลาดของข้อมูลได้สูงรหัสหนึ่ง รหัสแอลดีพีซีนั้นถูกคิดขึ้นโดย Robert Gallager [1] โดยเป็นวิทยานิพนธ์ระดับปริญญาเอก ณ *สถาบันเทคโนโลยีแมสซาชูเซตส์* (MIT : Massachusetts Institute of Technology) ในค.ศ. 1962 อย่างไรก็ตามในช่วงเวลานั้นรหัสชนิดนี้กลับไม่ได้รับความนิยมมากนักอัน-เนื่องมาจากมีความซับซ้อนสูง แต่หลังจากนั้นในปีค.ศ. 1996 Mackay and Neal [2] ได้นำรหัสชนิดนี้กลับมาพัฒนาใหม่และได้รับความแพร่หลายมากในปัจจุบัน

รหัสแอลดีพีซีเป็นรหัสบล็อกแบบเชิงเส้น ซึ่งถูกกำหนดโดยเมทริกซ์ตัวต้นกำเนิด และเมทริกซ์พาริตี ซึ่งสมรรถนะของรหัสก็ขึ้นอยู่กับการออกแบบของเมทริกซ์พาริตีนี้เอง จากที่ได้กล่าวได้แล้วในย่อหน้าก่อนหน้านี้ว่าวิทยานิพนธ์ฉบับได้มุ่งเน้นในการออกแบบวิธีการสร้างเมทริกซ์พาริตี ซึ่งจะมีเกิร์ต (girth) เป็นพารามิเตอร์หลักในการออกแบบ วิทยานิพนธ์ฉบับนี้ยังออกแบบเมทริกซ์พาริตีของรหัสแอลดีพีซี 2 ชนิดด้วยกัน ซึ่งทั้ง 2 ชนิดนั้นมีคุณลักษณะที่น่าสนใจในการออกแบบ โดยเมทริกซ์พาริตีที่ได้ออกแบบนั้นเป็นแบบที่เรียกว่า Quasi-cyclic และแบบที่มีน้ำหนักของหลักเท่ากับ 2

1.2 แนวทางของงานวิจัย

วิทยานิพนธ์ฉบับนี้ยังได้ศึกษารูปแบบของเมทริกซ์ $\mathbf{H}$ ที่ใช้ในรหัสแอลดีพีซีทั้ง Quasi-cyclic และแบบที่มีน้ำหนักของหลักเท่ากับ 2 เพื่อใช้ในการออกแบบเมทริกซ์ $\mathbf{H}$ ทั้ง 2 ชนิด อันเนื่องจากเมทริกซ์ทั้ง 2 แบบมีคุณสมบัติที่น่าสนใจกล่าวคือ แบบ Quasi-cyclic นั้นสามารถนำมาใช้ได้ดีกับฮาร์ดแวร์และแบบที่มีน้ำหนักของหลักเท่ากับ 2 จะให้ประสิทธิภาพสูงในระบบช่องสัญญาณแบบวนกลับ จากนั้นได้เสนอวิธีการสร้างเมทริกซ์พาริตีของรหัสแอลดีพีซีหรือเมทริกซ์ $\mathbf{H}$ โดยมีเกิร์ตเป็นพารามิเตอร์หลักในการออกแบบโดยมุ่งเน้นไปยังการทำให้เมทริกซ์ $\mathbf{H}$ ที่ออกแบบมานั้นมีเกิร์ตที่มีความยาวสูงหรือสามารถกำหนดความยาวของเกิร์ตได้ ในส่วนของการวัดสมรรถนะของรหัสแอลดีพีซีนั้นทำโดยการจำลองระบบช่องสัญญาณสื่อสารด้วยโปรแกรม MATLAB โดยสัญญาณรบกวนเป็นแบบช่องสัญญาณรบกวนแบบเกาส์สีขาวแบบบวก (AWGN : Additive White Gaussian Noise) และทำการเปรียบเทียบกับวิธีการสร้างเมทริกซ์แบบอื่น

1.3 วัตถุประสงค์ของงานวิจัย

1.3.1 คิดค้นวิธีการสร้างเมทริกซ์ $\mathbf{H}$ แบบใหม่ที่มีน้ำหนักของหลักเท่ากับ 2 และสามารถที่จะกำหนดเกิร์ตที่ต่ำที่สุดในเมทริกซ์ $\mathbf{H}$ ได้

1.3.2 คิดค้นวิธีการสร้างเมทริกซ์ $\mathbf{H}$ รหัสสำหรับรหัสแอลดีพีซีทั้งแบบ Quasi-cyclic เพื่อให้มีขนาดความยาวของเกิร์ตสูง

1.4 ขอบเขตวิทยานิพนธ์

1.4.1 ปรับปรุงการสร้างเมทริกซ์ $\mathbf{H}$ แบบ Quasi-cyclic โดยปรับปรุงให้มีขนาดของเกิร์ตสูงและสามารถให้ประสิทธิภาพที่ดีเมื่อนำมาทดสอบภายใต้การจำลองช่องสัญญาณแบบสัญญาณรบกวนแบบเกาส์สีขาวแบบบวกเมื่อเทียบกับอัลกอริทึมอื่น อาทิ SFT, Magic square

1.4.2 คิดค้นการสร้างเมทริกซ์ $\mathbf{H}$ ใหม่ชนิดที่มีน้ำหนักของหลักเท่ากับ 2 โดยที่สามารถกำหนดขนาดของเกิร์ตที่ต่ำที่สุดในเมทริกซ์ $\mathbf{H}$ ได้และสามารถให้สมรรถนะที่ดีเมื่อทดสอบประสิทธิภาพภายใต้การจำลองช่องสัญญาณแบบสัญญาณรบกวนแบบเกาส์สีขาวแบบบวกเมื่อเทียบกับอัลริทึมที่มีชื่อเสียง อาทิ PEG และ Bit filling

1.4.3 เขียนโปรแกรมจำลองช่องสัญญาณการสื่อสารโดยมีสัญญาณการรบกวนเป็นสัญญาณรบกวนแบบเกาส์สีขาวแบบบวก โดยใช้โปรแกรม MATLAB

1.5 ขั้นตอนการทำวิทยานิพนธ์

1.5.1 ศึกษาทฤษฎีรหัสช่องสัญญาณ

1.5.2 ศึกษาวิธีการเข้ารหัสและถอดรหัสแอลดีพีซีแบบต่าง ๆ

1.5.3 ศึกษาวิธีการสร้างเมทริกซ์ตัวถอดรหัสทั้งแบบแบบ Quasi-cyclic และแบบที่มีน้ำหนักของหลักเท่ากับ 2

1.5.4 ปรับปรุงการสร้างเมทริกซ์ $\mathbf{H}$ แบบ Quasi-cyclic เพื่อให้มีเกิร์ตสูง

1.5.5 คิดค้นอัลกอริทึมใหม่ในการสร้างเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของกลักเท่ากับ 2 และสามารถกำหนดเกิร์ตที่ต่ำที่สุดในเมทริกซ์ $\mathbf{H}$ ได้

1.5.6 ศึกษาวิธีการใช้โปรแกรม MATLAB

1.5.7 เขียนโปรแกรมจำลองช่องสัญญาณการสื่อโดยมีสัญญาณการรบกวนเป็นสัญญาณรบกวนแบบเกาส์สีขาวแบบบวก โดยใช้โปรแกรม MATLAB

1.6 ประโยชน์ที่คาดว่าจะได้รับ

1.6.1 อัลกอริทึมใหม่ในการสร้างเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของหลักเท่ากับ 2 และสามารถกำหนดเกิร์ตที่ต่ำที่สุดในเมทริกซ์ $\mathbf{H}$ ได้

1.6.2 ปรับปรุงการสร้างเมทริกซ์ $\mathbf{H}$ แบบ Quasi-cyclic เพื่อให้มีเกิร์ตสูงเมื่อเทียบกับการสร้าง Quasi-cyclic ชนิดอื่น อาทิ SFT, Magic square

1.6.3 อัลกอริทึมใหม่ที่สามารถลดความซับซ้อนของอัลกอริทึม PEG ในการสร้างเมทริกซ์ $\mathbf{H}$

บทที่ 2
รหัสแอลดีพีซี

2.1 รหัสบล็อกเชิงเส้น

รหัสแอลดีพีซีถือว่าเป็นรหัสที่ได้รับความนิยมอย่างมากในปัจจุบันเพราะได้รับการยอมรับว่ามีประสิทธิภาพเข้าใกล้ขีดจำกัดของแชนนอนมากที่สุด (Shannon's limit) [3] ซึ่งรหัสแอลดีพีซีนั้นจัดเป็นรหัสชนิดที่เรียกว่ารหัสบล็อกเชิงเส้นซึ่งจะอธิบายถึงความหมายของรหัสบล็อกเชิงเส้นดังต่อไปนี้

รหัสบล็อกเชิงเส้น (Linear block code) สามารถเขียนแทนด้วย (n,k) ซึ่งข้อมูลจะถูกส่งเป็นบล็อก แต่ละบล็อกมีจำนวน n บิตหรือที่เรียกว่าคำรหัส (codeword) และภายในคำรหัสจะถูกแบ่งเป็น 2 ส่วน ส่วนแรกคือข้อมูลที่ต้องการจะส่งหรือบิตข้อมูล (message bit) จะมีอยู่จำนวน k บิตและสามารถเขียนอยู่ในรูป $\boldsymbol{m} = (m_0, m_1, ..., m_{k-1})$ ส่วนที่สองคือบิตพาริตี (parity bit) หรือบิตที่จะนำมาแก้ไขคำรหัสจะมีจำนวน $n-k$ บิต โดยคำรหัสทั้งหมดที่ประกอบด้วยส่วนทั้ง 2 จะได้ว่า $\boldsymbol{c} = (c_0, c_1, ..., c_{n-1})$ จะมีสมการความสัมพันธ์ดังนี้

$$c_i = \begin{cases} p_i, & i=0,1,...,n-k-1 \\ m_{i+k-n}, & i=n-k,n-k+1,...,n-1 \end{cases} \qquad (2.1)$$

$$[c_0, c_1, c_2, ..., c_n] = \boxed{[m_0, m_1, m_2, ..., m_k] \mid [p_0, p_1, p_2, ..., p_{n-k}]}$$

รูปที่ 2.1 โครงสร้างของรหัสบล็อกเชิงเส้นแบบ (n,k)

รหัสบล็อกเชิงเส้นยังมีคุณสมบัติที่ว่าเมื่อนำคำรหัส 2 คำรหัสใด ๆ มาบวกกันผลลัพธ์ที่ได้จะเป็นอีกคำรหัสหนึ่ง นอกจากนี้การถอดรหัสบล็อกเชิงเส้นจะทำการถอดทีละบล็อกซึ่งขนาดของบล็อกจะขึ้นอยู่กับแต่ละแอปพลิเคชันที่ใช้ในระบบสื่อสาร ซึ่งอัตราส่วนของบิตข้อมูลกับจำนวนบิตทั้งหมดในคำรหัสนั้นเรียกว่า "อัตรารหัส" (R : code rate) นิยามได้ดังนี้

$$R = \frac{k}{n} \qquad (2.2)$$

2.2 มุมมองของเมทริกซ์

รหัสแอลดีพีซีจะถูกกำหนดโดยเมทริกซ์ 2 ตัวด้วยกันคือเมริกซ์ตัวต้นกำเนิด ($\mathbf{G}$ matrix) และเมทริกซ์พาริตี ($\mathbf{H}$ matrix) ที่ถูกกล่าวถึงไปแล้วในบทที่ผ่านมา เมทริกซ์ $\mathbf{G}$ มีหน้าที่ให้การให้กำเนิดคำรหัสและเมทริกซ์ $\mathbf{H}$ มีหน้าที่ในการถอดรหัสตามลำดับ ต่อไปนี้จะเป็นคำอธิบายถึงเมทริกซ์ทั้ง 2 ตัวนี้

2.2.1 เมทริกซ์ตัวกำเนิด

จากที่ได้กล่าวไปแล้วว่าเมทริกซ์ $\mathbf{G}$ มีไว้สำหรับในการเข้ารหัสของรหัสแอลดีพีซี ซึ่งกระบวนการคือการนำบิตข้อมูลมาคูณกับเมทริกซ์ $\mathbf{G}$ ก็จะได้คำรหัส โดยเมทริกซ์ $\mathbf{G}$ แบ่งออกเป็น 2 ส่วน จากสมาการ (2.3) จะเห็นว่าส่วนแรกคือเมริกซ์เอกลักษณ์ (identity matrix) และต่ออยู่กับส่วนที่ 2 คือเมทริกซ์ส่วนของพาริตี

$$\mathbf{G}_{k\times n} = [I_{k\times k} \mid P_{k\times(n-k)}] = \begin{bmatrix} 1 & 0 & \cdots & 0 & p_{1,1} & p_{1,2} & \cdots & p_{1,(n-k)} \\ 0 & 1 & \cdots & 0 & p_{2,1} & p_{2,2} & \cdots & p_{2,(n-k)} \\ 0 & 0 & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p_{k,1} & p_{k,2} & \cdots & p_{k,(n-k)} \end{bmatrix} \tag{2.3}$$

ซึ่งคำรหัสที่ได้นั้นก็จะมีขนาด $1\times n$ ในส่วนของการเข้ารหัสนั้นก็จะเป็นดังสมการที่ (2.4) ดังนี้

$$\begin{aligned} c = \mathbf{mG} &= [m_1, m_2, ..., m_k, p_1, p_2, ..., p_{n-k}] \\ &= [c_1, c_2, ..., c_n] \end{aligned} \tag{2.4}$$

สมการที่ (2.4) ยังแสดงให้เห็นอีกด้วยว่าบิตข้อมูลเป็นส่วนหนึ่งของคำรหัส ที่มีบิตพาริตีเพิ่มเข้ามาซึ่งบิตพาริตีเหล่านี้เอาไว้ใช้ในการแก้ไขความผิดพลาดของคำรหัสเท่านั้น ในเวลาถอดรหัสจึงไม่จำเป็นต้องนำบิตพาริตีมาคิดรวมเป็นประสิทธิภาพของรหัส

2.22. เมทริกซ์ตัวถอดรหัส

เมทริกซ์ $\mathbf{H}$ หรือเมทริกซ์พาริตีซึ่งวิทยานิพนธ์ฉบับนี้มุ่งเน้นในการพัฒนาเมทริกซ์ $\mathbf{H}$ นี้เอง ซึ่งเมทริกซ์ $\mathbf{H}$ นั้นมีผลอย่างมากต่อประสิทธิภาพของรหัสแอลดีพีซี นอกจากนี้เมทริกซ์ $\mathbf{H}$ ยังมีความสัมพันธ์กับเมทริกซ์ $\mathbf{G}$ ตามสมการที่ (2.5) อีกด้วย

$$\mathbf{HG}^{\mathbf{T}} = \mathbf{0} \tag{2.5}$$

จากสมการที่ (2.5) จะเห็นว่าทั้งเมทริกซ์ $\mathbf{G}$ และ $\mathbf{H}$ เมื่อนำมาคูณกันแล้วจะได้เวกเตอร์ 0 นอกจากนี้หากสมการที่ (2.3) คือเมทริกซ์ $\mathbf{G}$ ที่อยู่ในรูปของสมมาตร (systematic) แล้ว เมทริกซ์ $\mathbf{H}$ ที่อยู่ในรูปสมมาตรเช่นกันจะมีส่วนของเมทริกซ์พาริตีจะอยู่ส่วนหน้าและต่อกับเมทริกซ์เอกลักษณ์ที่อยู่ส่วนหลังตามสมการที่ (2.6)

$$\mathbf{H}_{(n-k)\times n} = [\mathbf{P}^{\mathbf{T}} \mid \mathbf{I}_{(n-k)\times(n-k)}] \tag{2.6}$$

เมทริกซ์ $\mathbf{H}$ ยังมีความสัมพันธ์กับคำรหัสซึ่งเป็นไปตามสมการที่ (2.6)

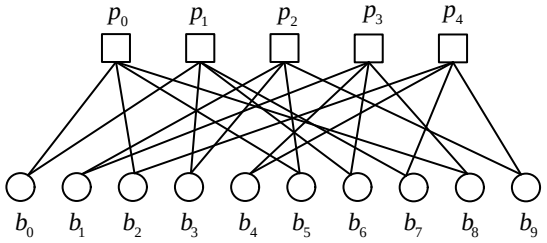
$$\mathbf{Hc}^{\mathbf{T}} = \mathbf{HG}^{\mathbf{T}}\mathbf{m}^{\mathbf{T}} = \mathbf{0} \tag{2.7}$$

2.3 กราฟแทนเนอร์

สำหรับเมทริกซ์ $\mathbf{H}$ ของรหัสแอลดีพีซียังสามารถที่จะมองได้ในรูปของกราฟแทนเนอร์ (Tanner's graph) [4] ซึ่งกราฟชนิดนี้คือกราฟ 2 ส่วน (bipartite graph) มีหมายความว่าเป็นกราฟที่เป็นการเชื่อมต่อระหว่างกลุ่มโนด 2 กลุ่ม และโนดในกลุ่มเดียวกันจะไม่เชื่อมต่อกันเอง

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

(ก)



(ข)

รูปที่ 2.2 เมทริกซ์พาริตีและกราฟแทนเนอร์ (ก) คือเมทริกซ์พาริตีและ (ข) กราฟแทนเนอร์ที่ได้จากเมทริกซ์ $\mathbf{H}$ ในรูปที่ 2.2 (ก)

จากรูปที่ 2.2 (ก) นั้นเมทริกซ์ $\mathbf{H}$ สามารถแปลงให้อยู่ในรูปของกราฟแทนเนอร์ในรูปที่ 2.2 (ข) ซึ่งแต่ละแถวของเมทริกซ์ $\mathbf{H}$ ในรูปที่ 2.2 (ก) จะถูกแทนด้วยโนดพาริตี (p_i) ในกราฟแทนเนอร์ในรูปที่ 2.2 (ข) ที่ $0 \le i \le m-1$ และแต่ละหลักของเมทริกซ์ $\mathbf{H}$ ในรูปที่ 2.2 (ก) ถูกแทนด้วยโนดบิต (b_i) ในกราฟแทนเนอร์ในรูปที่ 2.1 (ข) ที่ $0 \le i \le n-1$ โดย $m \times n$ เป็นมิติของเมทริกซ์ $\mathbf{H}$ นอกเหนือจากนี้ในกราฟแทนเนอร์จะมีของเส้นเชื่อม E ซึ่งเป็นเส้นที่ทำการเชื่อมต่อระหว่างกลุ่มโนดทั้ง 2 กลุ่ม โดยเซตของ E นั้นประกอบไปด้วยสมาชิก $(p_i, b_j) \in E$ และนิยามของน้ำหนักของทั้งโนดพาริตีและโนดบิตจะได้ว่า $\boldsymbol{D_C} = \{d_{c_0}, d_{c_1}, ..., d_{c_{M-1}}\}$ และ $\boldsymbol{D_s} = \{d_{s_0}, d_{s_1}, ..., d_{s_{N-1}}\}$ ตามลำดับ

อีกหนึ่งคุณสมบัติที่สำคัญของรหัสแอลดีพีซีคือ สมการที่เป็นภาวะคู่ของทุก ๆ สมการที่อยู่บนโนดพาริตี โดยอธิบายได้ดังนี้ จากการเข้ารหัสใน (2.4) และ $\mathbf{G}$ เมทริกซ์นั้นมีความสัมพันธ์กับเมทริกซ์ $\mathbf{H}$ ใน (2.5) และ (2.6) เมื่อนำทุกบิตที่เชื่อมอยู่กับโนดพาริตีใด ๆ มาบวกกันแล้วจะมีค่าเป็นภาวะคู่เสมอ ในที่นี้จะขอยกตัวอย่างกราฟแทนเนอร์ในรูป 2.2 ข) ซึ่งแต่ละสมการของแต่ละโนดพาริตีมีดังนี้

ที่ p_0 ได้ว่า $$b_0 \oplus b_2 \oplus b_5 \oplus b_9 = 0 \quad (2.8)$$

ที่ p_1 ได้ว่า $$b_0 \oplus b_3 \oplus b_6 \oplus b_7 = 0 \quad (2.9)$$

ที่ p_2 ได้ว่า $$b_1 \oplus b_3 \oplus b_5 \oplus b_9 = 0 \quad (2.10)$$

ที่ p_3 ได้ว่า $$b_1 \oplus b_4 \oplus b_6 \oplus b_8 = 0 \quad (2.11)$$

ที่ p_4 ได้ว่า $$b_2 \oplus b_4 \oplus b_7 \oplus b_9 = 0 \quad (2.12)$$

ทั้งหมดนี้จึงเป็นข้อสรุปได้ว่าสมการ (2.7) นั้นเป็นจริง ซึ่งในทางปฏิบัตินั้นภาครับสามารถที่จะรู้ได้ว่าเกิดความผิดพลาดของบิตข้อมูลที่ถูกส่งมาโดยใช้สมการที่ (2.7) ซึ่งผลคูณระหว่างคำรหัสและเมทริกซ์

$\mathbf{H}$ ที่ออกมานั้นเรียกว่าซินโดรม (syndrome) ซึ่งเป็นเวกเตอร์ หากเวกเตอร์ซินโดรมเป็น 0 ทั้งหมด จะหมายความว่าไม่มีบิตผิดเลย แต่หากไม่เป็นเช่นนั้นจะแสดงว่ามีบิตผิดอย่างน้อย 1 ตำแหน่ง

2.4 การเข้ารหัสและการถอดรหัสแอลดีพีซี

2.4.1การเข้ารหัสแอลดีพีซี

สำหรับการเข้ารหัสแอลดีพีซีทำได้โดยใช้สมการที่ (2.4) คือการนำคำรหัสมาคูณกับเมทริกซ์ $\mathbf{G}$ อย่างไรก็ตามในความเป็นจริงนั้นถ้ามีแต่เมทริกซ์ $\mathbf{H}$ ที่ไม่ได้เป็นแบบสมมาตรก็สามารถที่จะแปลงเมทริกซ์ $\mathbf{H}$ ให้อยู่ในรูปสมมาตรได้โดยใช้กระบวนการ row echelon form ซึ่งก็จะสามารถหาเมทริกซ์ $\mathbf{G}$ เมทริกซ์ที่อยู่ในรูปสมมาตรตามสมการที่ (2.3) ได้

2.2.4 การถอดรหัสแอลดีพีซี

การถอดรหัสแอลดีพีซีนั้นมีหลายวิธีซึ่งแบ่งเป็น 2 วิธีหลักคือ แบบฮาร์ด (hard decision) และแบบซอฟต์ (soft decision) วิธีทั้ง 2 จะมีคำอธิบายดังต่อไปนี้

2.4.2.1 การถอดรหัสแอลดีพีซีแบบฮาร์ด

การถอดรหัสแอลดีพีซีแบบฮาร์ดนั้นใช้วิธีการแบบเสียงข้างมากในการตัดสินคำรหัส วิธีนี้แม้ว่าจะมีความรวดเร็ว แต่การแก้ไขความผิดพลาดมักไม่มีประสิทธิภาพเท่ากับแบบซอฟต์ จากที่ได้กล่าวไปแล้วว่าสมการของบิตที่อยู่ติดกับโนดพาริตีทุกโนดนั้นมีภาวะเป็นคู่ทั้งหมด ตัวอย่างเช่น ถ้าใช้เมทริกซ์ $\mathbf{H}$ ในรูปที่ 2.2 (ก) ในการถอดรหัส จะได้ว่าบิตของคำรหัส ที่อยู่ในตำแหน่งของสมการที่ (2.8) คือบิตตำแหน่งที่ 0, 2, 5 และ 9 เมื่อนำทั้งหมดมาบวกกันจะได้เป็นภาวะคู่หรือมีค่าเป็น 0 จากทุกแถวของเมทริกซ์ $\mathbf{H}$ ในรูปที่ 2.2 (ก) จะมีสมการพาริตีทั้งหมด 5 สมการคือสมการที่ (2.8), (2.9), (2.10), (2.11) และ (2.12) โดยที่ตำแหน่งที่ j ของ b_j ใดๆนั้น ก็คือตำแหน่งที่ j^{th} ของโนดบิตในคำรหัส เมื่อทราบอย่างนี้แล้วการจะคำนวนหาค่าที่ควรจะเป็นของบิตข้อมูลใด ๆ ในคำรหัสก็จะสามารถหาได้โดยอาศัยภาวะคู่ของสมการพาริตี ยกตัวอย่างเช่น ค่าของบิตที่ตำแหน่ง b_0 สามารถคำนวนหาค่าได้จากสมการทั้ง 2 นี้คือ

$$b_2 \oplus b_5 \oplus b_9 = b_0 \qquad (2.13)$$

และ

$$b_3 \oplus b_6 \oplus b_7 = b_0 \qquad (2.14)$$

วิธีการนี้เรียกว่าการ***ปรับปรุงโนดพาริตี*** ซึ่งกระบวนการนี้ให้ทำทุก ๆ บิตของทุก ๆ สมการพาริตี ส่วนต่อมาเรียกว่า***การปรับปรุงโนดบิต*** มีตัวอย่างดังนี้ จากสมการที่ (2.8), (2.9), (2.10), (2.11) และ (2.12) นั้นมีการคำนวณหาบิตที่ตำแหน่ง b_0 จากสมการที่ (2.13) และ (2.14) ซึ่งผลลัพธ์ที่ได้จากสมการทั้ง 2 เมื่อนำมารวมกับค่าของบิตของคำรหัสที่ตำแหน่ง b_0 ที่ภาครับรับและดูว่าเสียงส่วนมากจะตัดสินใจว่าบิตที่ตำแหน่ง b_0 นั้นควรจะมีค่าใด ให้ทำเช่นนี้ไปจนตัดสินใจได้ครบทุกบิต

รหัสแอลดีพีซีเป็นรหัสที่มีการถอดรหัสแบบวนซ้ำ (iterative decoding) หมายความว่ากระบวนการถอดรหัสสามารถที่จะทำซ้ำอีกหลาย ๆ รอบได้ เพื่อให้ความถูกต้องของบิตมีเพิ่มมากยิ่งขึ้น

.2.2.42 การถอดรหัสแอลดีพีซีแบบซอฟต์

กระบวนการถอดรหัสแบบซอฟต์นั้นคล้ายกับแบบฮาร์ดแต่จากที่ตัดสินจากเสียงข้างมาก จะใช้ความน่าจะเป็นในการตัดสินแทน โดยใช้ความน่าจะเป็นแบบมีเงื่อนไขที่ ณ ภาครับคือ $p_i = \text{pr}[c_j = 1 | y]$ โดยที่ y คือค่าที่รับได้ ณ ภาครับ หมายความว่า $\text{pr}[c_i = 0 | y]$ จะมีค่าเท่ากับ $1 - p_i$ จากนั้นกำหนดให้ q_{ji}^l เป็นค่าความน่าจะเป็นที่วิ่งจากโนดบิต c_j ไปยังโนดพาริตีที่ p_i ในรอบที่ l ทุก ๆ ความน่าจะเป็นที่ q_{ij}^l จะเป็นความน่าจะเป็นที่เท่ากับ 0 คือ $q_{ij}^l(0)$ และความน่าจะเป็นที่เท่ากับ 1 คือ $q_{ji}^l(1)$ และ

$$q_{ij}^l(0) + q_{ij}^l(1) = 1 \tag{2.15}$$

ในทางกลับกันค่าความน่าจะเป็นที่ถูกส่งจากโนดพาริตี p_i ไปยังโนดบิต c_j คือ r_{ij}^l และ l คือรอบในการวนซ้ำและถูกส่งไปแบบคู่เหมือนเดิมคือ $r_{ij}^l(0)$ และ $r_{ij}^l(1)$ โดยความน่าจะเป็นที่จะเป็น 0 และ 1 ตามลำดับจะได้ว่า

$$r_{ji}^l(0) + r_{ji}^l(1) = 1 \tag{2.16}$$

และสมการสำหรับการปรับปรุงโนดบิตเป็นดังนี้

$$\text{pr}[c_1 \oplus c_2 \ldots \oplus c_n] = \frac{1}{2} + \frac{1}{2}\prod_{j=1}^{n}(1 - 2p_j) \tag{2.17}$$

พิสูจน์ถ้ามี 1 บิตจะได้ $p_j = \mathrm{pr}[c_j = 1 \mid y]$ และ

$$1 - p_j = \mathrm{pr}[c_j = 0 \mid y]$$

$1 - p_j$ เป็นภาวะคู่และ $1 - p_j = \frac{1}{2} + \frac{1}{2} - p_j$

$$1 - p_j = \frac{1}{2} + \frac{1}{2}(1 - 2p_j) \quad (2.18)$$

พิสูจน์ถ้ามี 2 บิตจะได้ภาวะคู่จะได้

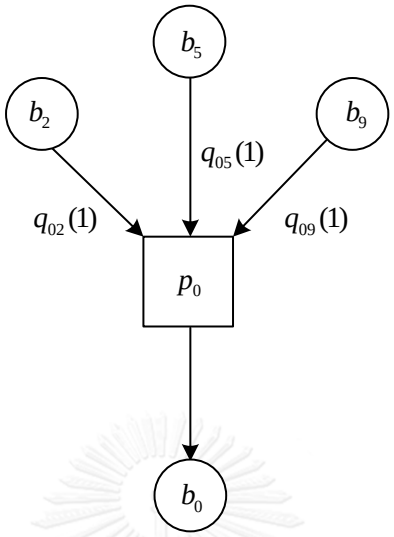
$$e(1 - \mathrm{p}_j) + (1 - e)\mathrm{p}_j$$
$$\mathrm{e} - e\,\mathrm{p}_j + \mathrm{p}_j - e\,\mathrm{p}_j$$
$$\mathrm{p}_j + \mathrm{e} - 2e\,\mathrm{p}_j$$
$$\mathrm{p}_j + \mathrm{e}(1 - 2\mathrm{p}_j)$$
$$\mathrm{p}_j + (\frac{1}{2} + \frac{1}{2}(1 - 2\mathrm{p}_j))(1 - 2\mathrm{p}_j)$$

จะได้

$$\frac{1}{2} + \frac{1}{2}(1 - 2p_1)(1 - 2p_2)$$

ซึ่งสอดคล้องกับสมการที่ (2.17) ซึ่งเป็นสมการที่ใช้ในการปรับปรุงโนดบิต เพื่อให้ง่ายต่อความเข้าใจต่อไปนี้จะเป็นการยกตัวอย่างการปรับปรุงของโนดพาริตีที่ p_0 ตามสมการที่ (2.8) ดังนี้

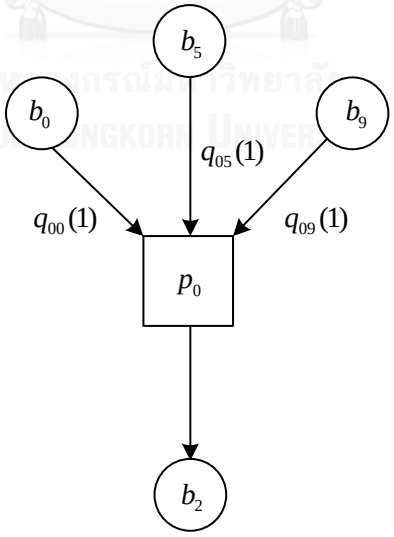
ค่าความน่าจะเป็นที่บิตที่ b_0 คำนวนได้จากรูปที่ 2.3



$$\text{pr}[c_0 = 0 \mid y] = \frac{1}{2} + \frac{1}{2}(1-2q(1)_{02})(1-2q(1)_{05})(1-2q(1)_{09})$$

รูปที่ 2.3 การคำนวนค่าความน่าจะเป็นของบิตที่ b_0 โดยอาศัยโนดพาริตี p_0

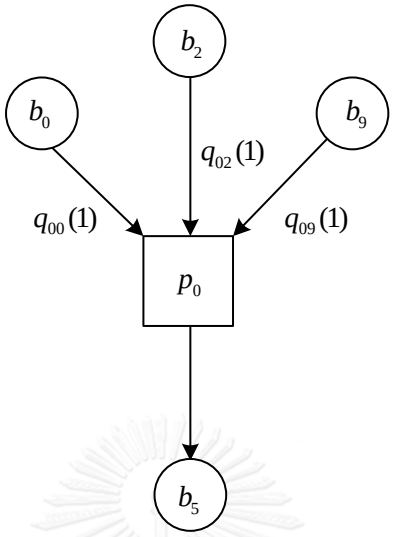
ค่าความน่าจะเป็นที่บิตที่ b_2 คำนวนได้จากรูปที่ 2.3



$$\text{pr}[c_2 = 0 \mid y] = \frac{1}{2} + \frac{1}{2}(1-2q(1)_{00})(1-2q(1)_{05})(1-2q(1)_{09})$$

รูปที่ 2.4 การคำนวนค่าความน่าจะเป็นของบิตที่ b_2 โดยอาศัยโนดพาริตี p_0

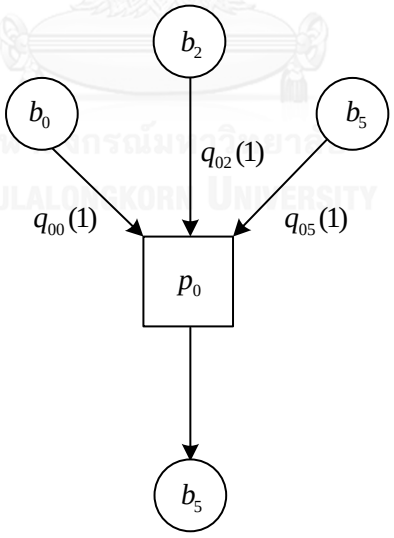
ค่าความน่าจะเป็นที่บิตที่ b_5 คำนวนได้จากรูปที่ 2.4



$$\text{pr}[c_5 = 0 \mid y] = \frac{1}{2} + \frac{1}{2}(1-2q(1)_{00})(1-2q(1)_{02})(1-2q(1)_{09})$$

รูปที่ 2.5 การคำนวนค่าความน่าจะเป็นของบิตที่ b_5 โดยอาศัยโนดพาริตี p_0

ค่าความน่าจะเป็นที่บิตที่ b_9 คำนวนได้จากรูปที่ 2.5



$$\text{pr}[c_9 = 0 \mid y] = \frac{1}{2} + \frac{1}{2}(1-2q(1)_{00})(1-2q(1)_{02})(1-2q(1)_{05})$$

รูปที่ 2.6 การคำนวนค่าความน่าจะเป็นของบิตที่ b_5 โดยอาศัยโนดพาริตี p_0

แน่นอนว่าค่าความน่าจะเป็นที่ได้เท่ากับจำนวนคี่หรือ 1 ก็คือการเอา 1 ไปลบกับค่าที่คำนวนมาได้นั้นเองและจะได้ว่า

$$r_{ji}^{l}(0) = \frac{1}{2} + \frac{1}{2}\prod_{j \in h_{ij} \neq i}(1 - 2q_j^{l-1}) \tag{2.19}$$

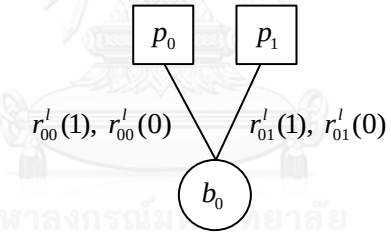
$$r_{ji}^{i}(1) = 1 - r_{ji}^{i}(0) \tag{2.20}$$

สำหรับการตัดสินใจแต่ละบิตนั้นจะใช้สมการดังต่อไปนี้

$$Q_i^l(0) = k_i(1 - p_i)\prod_{\forall j|\mathrm{H}_{ij}=1} r_{ji}^{l}(0) \tag{2.21}$$

$$Q_i^l(1) = k_i p_i \prod_{\forall j|\mathrm{H}_{ij}=1} r_{ji}^{l}(1) \tag{2.22}$$

โดย k คือค่าสัมประสิทธิ์ใด ๆ ที่ใช้ในการนอมอลไลซ์ (normalization) อย่างไรก็ตามสำหรับการถอดรหัสในรอบต่อ ๆ ไปนั้นไปนั้นแน่นอนว่าจำเป็นที่จะต้องมีการคำนวนในการปรับปรุงโนดพาริตี ซึ่งการคำนวนจะยังคงใช้สมการที่ (2.21) และ (2.22) แต่จะไม่มีค่าของโนดพาริตีที่เราต้องการนำมาปรับปรุงมารวมในการคำนวนด้วย



รูปที่ 2.7 ข้อมูลของโนดพาริตี p_0 และโนดพาริตี p_1 ที่ส่งมายังบิต b_0 ตามเมทริกซ์รูปที่ 2.2 ก)

รูปที่ 2.7 นั้นสรุปได้ว่าหากต้องการตัดสินโนดบิตให้ใช้ข้อมูลทั้งหมดของโนดพาริตีที่อยู่ติดกับโนดบิตที่ต้องการตัดสินใจ ถ้าต้องการปรับปรุงโนดพาริตีให้ใช้ข้อมูลทั้งหมดนอกจากข้อมูลจากโนดพาริตีที่ต้องการปรับปรุง

2.5 พารามิเตอร์สำหรับการออกแบบรหัสแอลดีพีซี

จากที่ได้กล่าวไปแล้วว่ารหัสแอลดีพีซีนั้นเป็นรหัสบล็อกแบบเชิงเส้นที่มีประสิทธิภาพสูง และมีขีดความสามารถที่เข้าใกล้ขีดจำกัดของแชนนอน เมื่อแต่ละบล็อกที่ส่งนั้นมีขนาดใหญ่และถอดรหัสด้วยวิธีแบบซอฟต์หรือ sum-product decoding [5] แต่อย่างไรก็ตามยังมีพารามิเตอร์อีกมากมายที่

สามารถส่งผลกระทบต่อประสิทธิภาพ ต่อไปนี้จะเป็นการกล่าวถึงพารามิเตอร์อื่น ๆ ที่มีผลต่อประสิทธิภาพของรหัสแอลดีพีซี [6] ดังต่อไปนี้

2.5.1 เกิร์ต

เกิร์ต (Girth) คือเส้นทางที่น้อยที่สุดที่เดินจากโนดบิต ๆ หนึ่งและกลับมายังโนดบิตเดิม ซึ่งวงโคจรอยู่ในกราฟของแทนเนอร์ การออกแบบรหัสแอลดีพีซีที่ดีนั้นควรจะมีความยาวของเกิร์ตที่สูงหรืออย่างน้อยที่สุดไม่ให้เกิดความยาวเท่ากับ 4

2.5.2 ระยะที่น้อยที่สุดของคำรหัส

ระยะแฮมมิงที่น้อยที่สุดของคำรหัส (Minimum distance) คือจำนวนบิตที่น้อยที่สุดที่แตกต่างกันของคำรหัสทั้งหมดที่เป็นไปได้ กล่าวคือมีหากระยะแฮมมิงที่น้อยที่สุดของคำรหัสมีค่าสูงแล้วการตรวจจับความผิดพลาดจะมีประสิทธิภาพมากขึ้น เนื่องจากคำรหัสแต่ละคำถ้ามีความแตกต่างกันมากการถอดรหัสก็จะมีความถูกต้องสูง

2.5.3 เซตหยุด

เซตหยุด (stopping set distribution) ที่มีขนาดเล็กนั้นสามารถที่จะลดประสิทธิภาพของการถอดรหัสแอลดีพีซีแบบ message-passing ได้ จึงทำให้ขนาดที่เล็กที่สุดของ stopping set สามารถที่จะบอกถึงประสิทธิภาพของการถอดรหัสได้หากใช้การถอดรหัสแอลดีพีซีแบบ message-passing

2.5.4 ความหนาแน่น

คำว่า low density parity check codes นั้นมีความหมายถึงเมทริกซ์ $\mathbf{H}$ ที่มีจำนวนเลข 1 น้อยในเมทริกซ์ (sparse matrix) แต่หากในเมทริกซ์ $\mathbf{H}$ มีเลข 1 มากก็จะช่วยเพิ่มประสิทธิภาพในการถอดรหัสแต่ก็จะเพิ่มความซับซ้อนให้มากขึ้นด้วยเนื่องจากการเพิ่มเลข 1 นั้นหมายถึงการเพิ่มน้ำหนักของหลักในเมทริกซ์ $\mathbf{H}$ ซึ่งจะเป็นการเพิ่มตัวช่วยในการตัดสินใจบิตข้อมูล อย่างไรก็ตามหากเพิ่มเลข 1 มากเกินไปขนาดของเกิร์ตที่ได้จะมีจำนวนลดลงซึ่งทำให้ประสิทธิภาพของรหัสแอลดีพีซีลดลงตามไปด้วย

บทที่ 3
อัลกอริทึม PEG, PEG-Like และ Bit filling

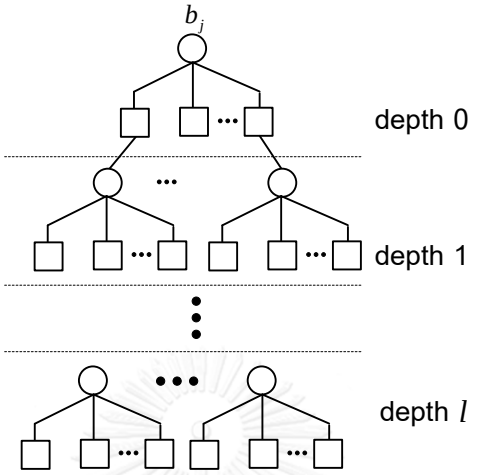
บทนี้จะเป็นการนำเสนออัลกอริทึมที่มีชื่อเสียงอย่างมากอย่างอัลกอริทึม Progressive-Edge-Growth (PEG) ที่ถูกเสนอโดย Xiao-Yu Hu และ Evangelos Eleftheriou [7] อัลกอริทึมชนิดนี้เป็นที่สนใจและถูกนักวิจัยนำมาปรับปรุงอย่างมาก อาทิ [8], [9] และ [10] ความพิเศษของอัลกอริทึมชนิดนี้คือสามารถสร้างเมทริกซ์ $\mathbf{H}$ ที่มีเกิร์ตสูงได้ ด้วยเหตุนี้เองอัลกอริทึม PEG จึงเหมาะสมสำหรับการนำมาเปรียบเทียบประสิทธิภาพกับอัลกอริทึมที่นำเสนอในบทที่ 5 รวมทั้งในบทนี้ยังเสนองานวิจัยที่สามารถลดความซับซ้อน (complexity) จากการสร้างเมทริกซ์ $\mathbf{H}$ ของอัลกอริทึม PEG ที่มีชื่อว่า PEG-like algorithm [11] ที่นอกจากมีข้อดีในการลดความซับซ้อนของการสร้างเมทริกซ์ $\mathbf{H}$ ของอัลกอริทึม PEG ได้แล้ว ยังสามารถใช้ลดความซับซ้อนของการสร้างเมทริกซ์ $\mathbf{H}$ ของอัลกอริทึมที่ปรับปรุงมาจากอัลกอริทึม PEG ได้อีกด้วย ในการแนะนำอัลกอริทึมทั้ง 2 จะเป็นประโยชน์ในการปูพื้นฐานที่จะใช้การอ้างอิงถึงทั้งในบทที่ 4 และบทที่ 5 ส่วนอีกอัลกอริทึมหนึ่งที่จะถูกนำเสนอในบทที่ 3 เช่นเดียวกันคืออัลกอริทึมที่ชื่อว่า bit filling ซึ่งถูกเสนอโดย [12] อัลกอริทึมชนิดนี้จะถูกนำไปใช้ในการเปรียบเทียบผลการทดลองเพื่อวัดประสิทธิภาพกับอัลกอริทึมที่นำเสนอเช่นเดียวกับอัลกอริทึม PEG ในบทที่ 5 เนื่องจากคุณสมบัติเด่นของอัลกอริทึมชนิดนี้ที่สามารถกำหนดเกิร์ตต่ำสุดในเมทริกซ์ $\mathbf{H}$ ได้จึงมีความคล้ายคลึงกันกับอัลกอริทึมที่นำเสนอในบทที่ 5

3.1 อัลกอริทึม Progressive Edge-Growth

พารามิเตอร์ที่ใช้ในอัลกอริทึม PEG ส่วนใหญ่ถูกอ้างอิงไปแล้วในหัวข้อที่ 2.3 เรื่องของกราฟแทนเนอร์ ในที่นี้จึงขอเพิ่มเติมรายละเอียดอีกเล็กน้อยดังต่อไปนี้ แต่ละน้ำหนักของโนดบิตจะถูกนิยามโดย $E_{b_j}^k$ โดยที่ $0 \le k \le d_{b_j} - 1$ คือการแทนอันดับของน้ำหนักแต่ละอันดับของโนดบิต b_j ใด ๆ

กระบวนการที่สำคัญมากของอัลกอริทึม PEG คือการขยายกราฟ (spreading subgraph) ของโนดบิด b_j ใด ๆ ในรูปกราฟแบบต้นไม้ (graph tree) ซึ่งจะได้ตามรูปที่ 3.1 โดยมีความลึก (depth) เริ่มจากความลึก 0 ถึง l และแต่ละความลึกนั้นนับจากที่มีการขยายกราฟไปถึงโนดพาริตีทุก ๆ โนดพาริตีที่อยู่ภายใต้ความลึกที่ l นี้จะอยู่ในเซต $N_{b_j}^l$ และทุก ๆ โนดพาริตีที่ไม่อยู่ภายใต้ความลึกที่ l จะอยู่ในเซต $\bar{N}_{b_j}^l$ ซึ่งจะได้ว่า $N_{b_j}^l \cup \bar{N}_{b_j}^l = V_{p_i}$ ที่ V_{p_i} คือเซตของทุก ๆ โนดพาริตีที่อยู่บนของกราฟแทนเนอร์

กระบวนการสร้างเมทริกซ์ $\mathbf{H}$ โดยใช้อัลกอริทึม PEG เริ่มต้นจากการกำหนดมิติของเมทริกซ์ขนาด $M \times N$ และน้ำหนักของหลักที่ต้องการ การสร้างนั้นเริ่มจากหลักที่ 1 ไปจนถึงหลักสุดท้ายและแต่ละหลักนั้นเริ่มจากน้ำหนักของหลักที่อันดับที่ 0 ถึงน้ำหนักอันดับที่ $d_{b_j} - 1$



รูปที่ 3.1 การขยายกราฟจากโนดบิด b_j ใดๆ

โดยถ้าเป็นน้ำหนักอันดับที่ 0 (หรือน้ำหนักแรกของแต่ละโนดบิต) ให้ทำการเลือกแบบสุ่ม (random) กับโนดพาริตีที่มีน้ำหนักน้อยที่สุดมาต่อกับโนดบิตที่กำลังสนใจอยู่ หากเป็นน้ำหนักมากกว่า 0 ให้ทำการขยายกราฟแบบต้นไม้จากโนดบิตที่สนใจอยู่ไปจนถึงความลึกที่ l ซึ่งจะเกิด 2 เงื่อนไขดังนี้

1) $\bar{N}_{b_j}^{l} \neq \phi$ ให้ทำการเลือกโนดพาริตีของเซตของ $\bar{N}_{b_j}^{l}$ ที่มีน้ำหนักน้อยที่สุดมาต่อกับโนดบิตที่กำลังสนใจอยู่
2) $\bar{N}_{b_j}^{l} = \phi$ ให้เลือกโนดพาริตีที่มีน้ำหนักน้อยที่สุดที่อยู่ในความลึกที่ l เท่านั้นมาต่อกับโนดบิตที่กำลังสนใจอยู่ จะเห็นได้ว่าหากเป็นกรณีที่ 1 เกิร์ตที่เกิดขึ้นมาใหม่ที่มาจากการต่อโนดพาริตีที่มีน้ำหนักน้อยที่สุดในเซต $\bar{N}_{b_j}^{l}$ กับโนดบิตที่กำลังสนใจนั้นเกิร์ตจะมีค่าเป็น ∞ หรือไม่เกิดเกิร์ตขึ้นเลย แต่หากเกิดกรณีที่ 2 นั้นเกิร์ตที่เกิดจากการต่อโนดพาริตีที่มีน้ำหนักน้อยที่สุดในความลึกที่ l กับโนดบิตที่กำลังสนใจนั้นเกิร์ตจะมีค่าเป็น $2l+1$

Progressive Edge-Growth (PEG) algorithm

for $j = 0$ to $N - 1$ do

begin

for $k = 0$ to $d_{p_i} - 1$ do

begin

if $k = 0$

$E_{b_j}^0 \leftarrow$ edge (p_i, b_j) , where $E_{b_j}^0$ is the first edge incident to b_j and p_i is a check node such that it has the lowest check-node degree under the current graph setting $E_{b_0} \cup E_{b_1} \cup ... \cup E_{b_{j-1}}$

else

expand a subgraph from bit node b_j up to depth l under the current graph setting such that the cardinality of $N_{b_j}^l$ stop increasing but is less than M or $\bar{N}_{b_j}^l \neq \phi$ but $\bar{N}_{b_j}^{l+1} = \phi$ then b_j and p_i is a check node picked from the set $\bar{N}_{b_j}^l$ having the lowest check-node degree

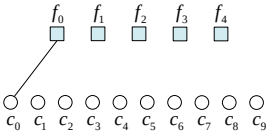
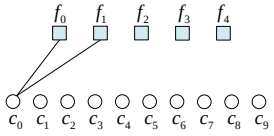
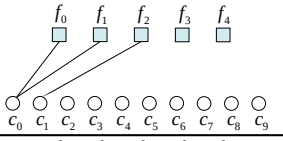
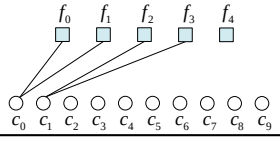
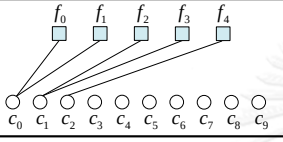
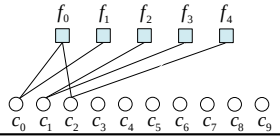
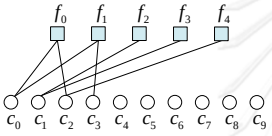
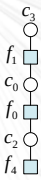
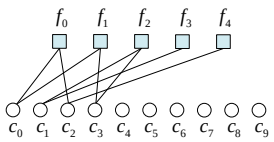
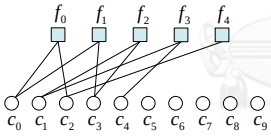
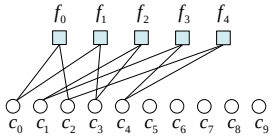
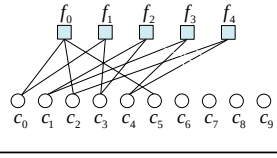
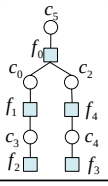
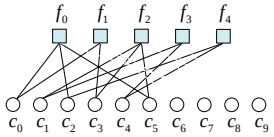
end

end

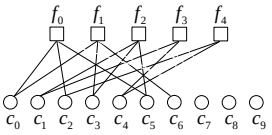
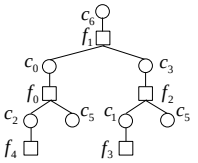
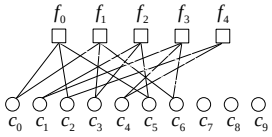
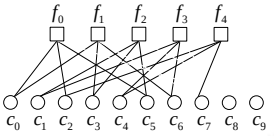
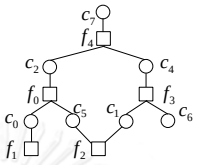
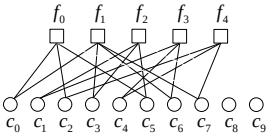
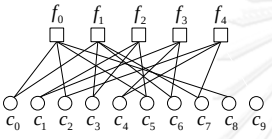
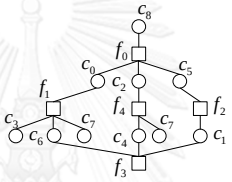
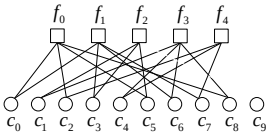
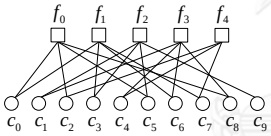
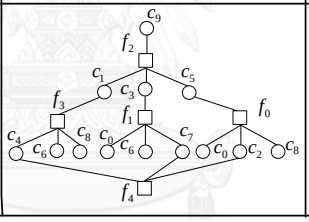
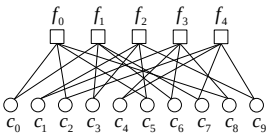
จากที่ตารางที่ 3-1 สรุปวิธีสร้างเมทริกซ์ $\mathbf{H}$ โดยใช้อัลกอริทึม PEG ในการสร้างเมทริกซ์ขนาด 5×10 และมีน้ำหนักของหลักที่เท่ากับ 2 ต่อไปจะขออธิบายเพียงบางส่วนเพื่อให้เกิดความเข้าใจมากยิ่งขึ้นเพราะส่วนที่ไม่ได้อธิบายไว้นั้นจะใช้วิธีเดียวกันกับที่อธิบายไว้แล้ว

พิจารณาการสร้างที่โนดบิต b_0 การเลือกเส้นเชื่อมที่ $E_{b_0}^0$ นั้นเป็นการเลือกแบบสุ่ม จะเห็นได้ว่าเริ่มแรกนั้นทุก ๆ โนดพาริตีต่างมีน้ำหนักเป็น 0 ทั้งหมด ฉะนั้นจึงสามารถเลือกให้เส้นเชื่อมที่ $E_{b_0}^0$ ไปต่อกับโนดพาริตีใดก็ได้ (ในที่นี้เลือกเชื่มกับโนดพาริตีที่ p_0) จากที่กล่าวไปแล้วว่าต้องการสร้างเมทริกซ์ที่มีน้ำหนักของหลักเท่ากับ 2 โนดบิต b_0 จึงต้องมีเส้นเชื่มที่ $E_{b_0}^1$ เพิ่มขึ้นมา กระบวนการนี้ต้องขยายกราฟแบบต้นไม้จากโนดบิตที่ b_0 ซึ่งได้ความลึกเท่ากับ 0 เท่านั้น แต่จะเห็นได้ว่ามีโนดพาริตีที่เป็นสมาชิกใน $\bar{E}_{b_0}^0$ อยู่ด้วย จึงต้องเลือกเส้นเชื่อม $E_{b_0}^1$ ไปเชื่อมกับโนดพาริตีที่มีน้ำหนักน้อยที่สุดใน $\bar{N}_{b_0}^l$ (ในที่นี้เลือกเชื่มกับโนดพาริตีที่ p_1) ต่อมาดูการสร้างเส้นเชื่อมที่โนด b_9 เนื่องจากเส้นเชื่อม $E_{b_9}^0$ นั้นเป็นการเลือกแบบสุ่มดังนั้นจึงขอข้ามไป มาดูที่การเชื่อมเส้นเชื่อมที่ $E_{b_9}^1$ หลังจากการขยายกราฟแบบต้นไม้จากโนดบิตที่ b_9 ซึ่งจะได้ความลึกเท่ากับ 2 และไม่มีโนดพาริตีใดอยู่ในเซตของ $\bar{N}_{b_9}^l$ เลย ทำให้ต้องเลือกเชื่อมกับโนดพาริตีที่อยู่ในความลึกที่ 2 ซึ่งมีสมาชิกอยู่ตัวเดียวนั่นคือ p_4 นั้นเอง

ตารางที่ 3-1 รายละเอียดการสร้างเมทริกซ์พาริตีเช็กขนาด 5x10 โดยใช้อัลกอริทึม PEG

โนดบิตที่พิจารณา	การจัดสรรกิ่งเชื่อมต่อจากโนดบิตไปยังโนดตรวจสอบครั้งแรก	ผลการขยายกราฟย่อยที่เริ่มต้นจากโนดบิต	การจัดสรรกิ่งเชื่อมต่อจากโนดบิตไปยังโนดตรวจสอบครั้งที่สอง
c_0			
c_1			
c_2			
c_3			
c_4			
c_5			

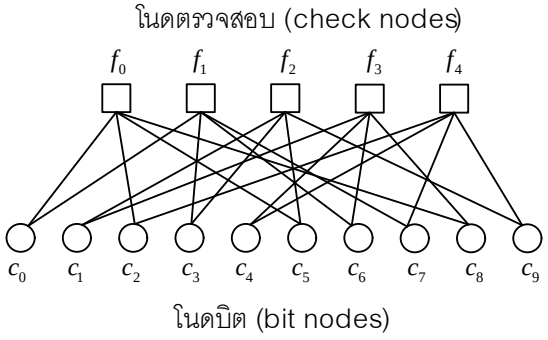
ตารางที่ 3-1 รายละเอียดการสร้างเมทริกซ์พาริตีเช็กขนาด 5x10 โดยใช้อัลกอริทึม PEG (ต่อ)

โนดบิตที่พิจารณา	การจัดสรรกิ่งเชื่อมต่อจากโนดบิตไปยังโนดตรวจสอบครั้งแรก	ผลการขยายกราฟย่อยที่เริ่มต้นจากโนดบิต	การจัดสรรกิ่งเชื่อมต่อจากโนดบิตไปยังโนดตรวจสอบครั้งที่สอง
c_6			
c_7			
c_8			
c_9			

หลังจากที่เสร็จทุกขั้นตอนตามตารางที่ 3.1 แล้ว จะได้เมทริกซ์ $\mathbf{H}$ ที่รูปที่ 3.2 (ก) และกราฟของแทนเนอร์ตามรูปที่ 3.2 (ข)

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

(ก)



(ข)

รูปที่ 3.2 (ก) เมทริกซ์พาริตีที่ถูกสร้างด้วยอัลกอริทึมแบบ PEG (ข) กราฟของแทนเนอร์ที่ถูกสร้างด้วยอัลกอริทึมแบบ PEG

3.2 อัลกอริทึม PEG-like

อัลกอริทึมที่ชื่อว่า PEG-like ถูกคิดขึ้นเพื่อลดความซับซ้อนในการขยายกราฟแบบต้นไม้ซึ่งเป็นหนึ่งในกระบวนการสำคัญของอัลกอริทึม PEG โดยแนวคิดนั้นมาจากการที่ตัดการค้นหา (search) โนดบิตในการขยายกราฟแบบต้นไม้ ที่ทำเช่นนี้ได้อันเนื่องมาจากในกราฟแทนเนอร์นั้น ระหว่างโนดพาริตีคู่ใด ๆ จะมีโนดบิตเชื่อมอยู่ระหว่างกลางเสมอคือคุณสมบัติของความเป็นกราฟสองส่วน (bipartite graph) จากคุณสมบัติดังกล่าวนี้เองทำให้สามารถที่จะแทนเมทริกซ์ $\mathbf{H}$ ขนาด $M\times N$ ด้วยโทโพโลยี (topology) ที่มีขนาด $M\times M$ ได้ซึ่งวิธีการนั้นจะอธิบายต่อไป

ในอัลกอริทึม PEG นั้นการขยายกราฟแบบต้นไม้จากโนดบิตที่ b_j ใด ๆ ต้องอาศัยการค้นหาทั้งหลักและแถวเพื่อไปถึงความลึกที่ l แต่ละหลักนั้นก็ต้องค้นหาถึง $\sum_{i=1}^{\text{column weight - 1}}2(l_i+1)$ ด้วยกัน แต่หาก $\mathbf{H}$ เมทริกซ์ถูกแทนที่ด้วยโทโพโลยีแล้วจำนวนครั้งที่ต้องการในการค้นหาในแต่ละหลักนั้นจะต้องการเพียง $\sum_{i=1}^{\text{column weight - 1}}(l_i+1)$ เท่านั้น ฉะนั้นจำนวนครั้งในการค้นหาและเวลาในการขยายกราฟแบบต้นไม้ก็คือความซับซ้อนนั้นเอง

3.2.1 วิธีการสร้างเมทริกซ์ตัวถอดรหัสด้วยอัลกอริทึม PEG-like

เริ่มต้นด้วยโทโพโลยีขนาด $M\times M$ โดยกำหนดเป็น $T_{M\times M}$ แต่ละแถวของ $T_{M\times M}$ สามารถแทนที่ได้ด้วยแต่ละแถวของเมทริกซ์ $\mathbf{H}$ ในรูปที่ 3.3 แสดงโทโพโลยีเริ่มต้นขนาด 5×5 และทุกเอลิเมนต์

	t_{i0}	t_{i1}	t_{i2}	t_{i3}	t_{i4}
t_{0j}	0	0	0	0	0
t_{1j}	0	0	0	0	0
t_{2j}	0	0	0	0	0
t_{3j}	0	0	0	0	0
t_{4j}	0	0	0	0	0

รูปที่ 3.3 โทโพโลยีเริ่มต้นขนาด 5×5

ในโทโพโลยีนี้มีค่าเป็น 0 ทั้งหมด และทุก ๆ เอลิเมนต์ในโทโพโลยีนั้นแทนด้วยสัญลักษณ์ t_{pq} ที่ $0\le p,q\le M-1$ ละ p คือตำแหน่งของแถว และ q คือตำแหน่งของหลักในโทโพโลยีตามลำดับ ถ้า $t_{pq}\neq 0$ หมายความว่าที่แถว p^{th} และ q^{th} ถูกโนดบิตใดบิตหนึ่งเชื่อมต่ออยู่ในเมทริกซ์ $\mathbf{H}$ ต่อไปนี้จะเป็นการอธิบายถึงอัลกอริทึม PEG-like

1. เลือก p^{th} ใด ๆ ในโทโพโลยีที่มีเลข 0 อยู่มากที่สุด ให้กำหนดเป็นแถวตั้งต้นหรือ S_{pq}
2. ถ้า $S_{pq}=\{0,0,..,0_M\}$ ให้เลือกหลักใดก็ได้มา 1 หลักและใส่เลขสัญลักษณ์ $t_{pq}=t_{qp}$ ซึ่งเลขนี้เองจะเป็นตัวบอกเลขลำดับของหลักในเมทริกซ์ $\mathbf{H}$ หรือดูเลขของหลักที่ไม่ได้เป็น 0 ใน S_{pq} นำเลขของทุกหลักที่ได้มานั้นมาเป็นเลขของทุกแถวที่ต้องการค้นหาในรอบต่อไป และให้หาเลขของหลักของแถวในรอบนั้น ๆ ทำแบบนี้ต่อไปจนไม่สามารถหาเลขของหลักที่จะนำมาเป็นแถวในการค้นหาต่อไปได้ จากนี้ไปก็จะเกิด 2 เงื่อนไข
   2.1 หากการค้นหานั้นค้นหาไปครบทุกแถวในโทโพโลยีแล้ว ให้นำเลขของแถวที่ถูกค้นหาเป็นแถวสุดท้ายที่มีเลข 0 มากที่สุดมาเป็นเลขของหลักที่จะใส่เลขสัญลักษณ์ของแถวใน S_{pq} จากนั้นใส่สัญลักษณ์
   2.2 หากการค้นหานั้นค้นหาไปไม่ครบทุกแถวในโทโพโลยี ให้นำเลขของแถวที่ไม่ถูกค้นหาที่มีเลข 0 มากที่สุดมาเป็นเลขของหลักที่จะใส่เลขสัญลักษณ์ของแถวใน S_{pq} จากนั้นใส่สัญลักษณ์
3. ทำแบบนี้ไปจนสามารถเติมสัญลักษณ์ครบทั้ง $M-1$ สัญลักษณ์ จำไว้เสมอว่าเลขสัญลักษณ์ที่ใส่ไปในโทโพโลยีนั้นเป็นเลขลำดับของเมทริกซ์ $\mathbf{H}$

อัลกอริทึม PEG-like

For $j=0$ to $n-1$

For $k=0$ to $d_{sj}-1$ do

If $k = 0$

Choose the *i*-th row randomly in $T_{M\times M}$ that has the largest number of 0's and assign it to S_{pq}.

Else

Select all the *i*-th row in $T_{M\times M}$ adjacent to the *j*-th column

End

do the Step 2

End

End

เพื่อให้ง่ายต่อความเข้าใจจึงขอยกตัวอย่างการสร้างเมทริกซ์ $\mathbf{H}$ ด้วยอัลกอริทึม PEG-like ดังต่อไปนี้

จากรูปที่ 3.3 ที่เป็นโทโพโลยีแบบเริ่มต้นนั้นจะเห็นว่าทุกแถวนั้นมีจำนวน 0 เท่ากัน จึงสามารถเลือกแถวใดก็ได้เป็น S_{pq} ในตัวอย่างจะเลือกแถวที่แรกหรือ $p=0$ จะเห็นได้ว่าแถวนี้ไม่มีหลักใดมีค่าที่มากกว่า 0 เลยในที่นี้สามารถที่จะเลือกหลักที่ 2 ได้หรือ $q=1$ (เป็น $q=0$ ไม่ได้เพราะโนดบิตแต่ล่ะโนดไม่สามารถเลือกตัวเองได้) เมื่อเลือกแล้วก็ใส่สัญลักญณ์เลข 1 ที่ $t_{0,1}$ และ $t_{1,0}$ (ที่ใส่สัญลักษณ์ 1 เพราะต้องการสร้างหลักที่ 1 ใน $\mathbf{H}$ เมทริกซ์) ซึ่งจะเป็นดังรูปที่ 3.4

	t_{i0}	t_{i1}	t_{i2}	t_{i3}	t_{i4}
t_{0j}	0	1	0	0	0
t_{1j}	1	0	0	0	0
t_{2j}	0	0	0	0	0
t_{3j}	0	0	0	0	0
t_{4j}	0	0	0	0	0

รูปที่ 3.4 โทโพโลยีขนาด 5×5 ที่เริ่มสร้างหลักที่ 1 ในเมทริกซ์พาริตีไปแล้ว

จากเมทริกซ์ $\mathbf{H}$ ตามรูปที่ 3.5 ซึ่งจะเห็นว่าหลักแรกของเมทริกซ์มีแถวแรกและแถวที่สองมีค่าเท่ากับ 1

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

รูปที่ 3.5 เมทริกซ์พาริตีที่ถูกสร้างจากรูปที่ 3.4

จากนั้นจะขออธิบายการสร้างหลักที่ 10 กันบ้าง ซึ่งก่อนที่จะทำการอธิบายจะขอแสดงโทโพโลยีที่ได้ทำไปถึงหลักที่ 9 ก่อนซึ่งแสดงอยู่ในรูป 3.6

	t_{i0}	t_{i1}	t_{i2}	t_{i3}	t_{i4}
t_{0j}	0	1	6	9	3
t_{1j}	1	0	4	7	8
t_{2j}	6	4	0	2	0
t_{3j}	9	7	2	0	5
t_{4j}	3	8	0	5	0

รูปที่ 3.6 โทโพโลยีขนาด 5×5 ที่สร้างไปถึงหลักที่ 9 ในเมทริกซ์ $\mathbf{H}$ ไปแล้ว

จากรูปที่ 3.6 จะเห็นว่าแถวที่ $p=2$ และ $p=4$ ในโทโพโลยีนั้นมีเลข 0 มากที่สุดคือ 2 ตัว แต่ในที่นี้ขอเลือกแถวที่ $p=2$ เป็น S_{pq} ปรากฏว่าในแถวนี้มีหลักที่ไม่ใช่ 0 คือ $q=0,\ q=1$ และ $q=3$ ให้นำหลักที่ q ทั้งหมดที่หาได้นี้มาเป็นแถวที่ใช้ค้นหาต่อไปในโทโพโลยี จะได้ว่า

แถว p_0 ได้หลักที่ไม่ใช่ 0 คือ $q=1,\ q=2,\ q=3$ และ $q=4$

แถว p_1 ได้หลักที่ไม่ใช่ 0 คือ $q=0,\ q=2,\ q=3$ และ $q=4$

แถว p_3 ได้หลักที่ไม่ใช่ 0 คือ $q=0,\ q=1,\ q=2$ และ $q=4$

ซึ่งแถวที่ถูกค้นหาไปแล้วจะไม่ได้นำมารวมด้วย จึงจะเหลือเพียงหลักเดียวเท่านั้นที่ยังไม่ถูกค้นหาคือ $q=4$ จากตรงนี้จึงพบว่าไม่มีหลักใดที่ไปไม่ถึงอีก หลักสุดท้ายที่ไปถึงคือ $q=4$ จึงเลือก $q=4$ ในการใส่สัญลักษณ์ เมื่อเลือกแล้วก็ใส่สัญลักษณ์เลข 10 ที่ $t_{2,4}$ และ $t_{4,2}$ และจะได้โทโพโลยีตามรูปที่ 3.7

	t_{i0}	t_{i1}	t_{i2}	t_{i3}	t_{i4}
t_{0j}	0	1	6	9	3
t_{1j}	1	0	4	7	8
t_{2j}	6	4	0	2	0
t_{3j}	9	7	2	0	5
t_{4j}	3	8	0	5	0

รูปที่ 3.7 โทโพโลยีขนาด 5×5 ที่สร้างไปถึงหลักที่ 10 ของเมทริกซ์ $\mathbf{H}$

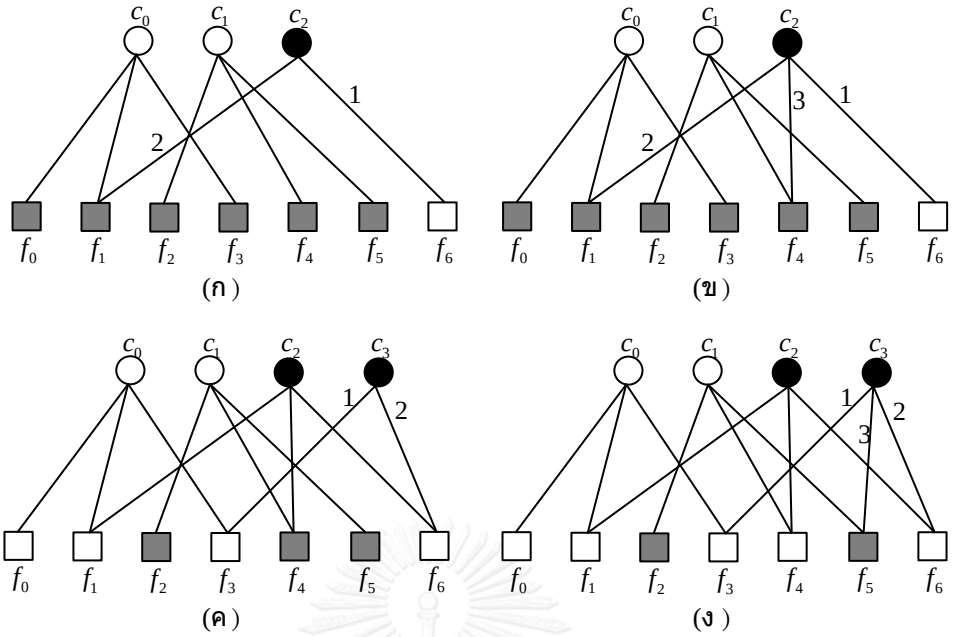
ซึ่งเมื่อแปลงโทโพโลยีที่รูปที่ 3.7 แล้วจะได้เมทริกซ์ $\mathbf{H}$ ตามรูปที่ 3.8

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

รูปที่ 3.8 เมทริกซ์พาริตีที่ถูกสร้างจากรูปที่ 3.7

3.3 อัลกอริทึม Bit filling

จากที่ได้กล่าวไว้แล้วในตอนต้นว่านอกจากอัลกอริทึม PEG และอัลกอริทึม PEG-like ยังมีอีกอัลกอริทึมหนึ่งที่ชื่อว่า bit filling ที่จะถูกนำเสนอในบทนี้ด้วย อัลกอริทึมชนิดนี้จะถูกนำไปเปรียบเทียบประสิทธิภาพกับงานวิจัยที่นำเสนอในบทที่ 5 ร่วมกับอัลกอริทึม PEG และจะขออธิบายเกี่ยวกับอัลกอริทึม bit filling ดังต่อไปนี้

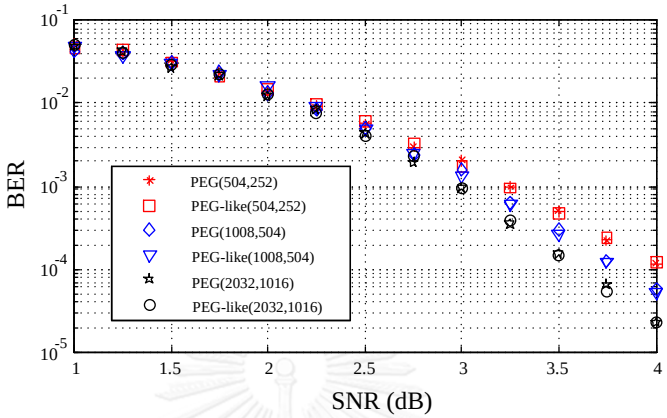


รูปที่ 3.9 ตัวอย่างการสร้างเมทริกซ์พาริตีในแบบ bit filling อัลกอริทึม

อัลกอริทึม bit filling เป็นอัลกอริทึมที่ใช้ในการสร้างเมทริกซ์ $\mathbf{H}$ โดยมีวิธีการคือพยายามเพิ่มโนดบิตไปเรื่อย ๆ เพื่อให้ได้จำนวนที่ต้องการ โดยที่โนดบิตที่ได้เพิ่มมานั้นจะเชื่อมต่อกับโนดพาริตีตามน้ำหนักของหลักที่ได้กำหนดไว้และจะมีเกิร์ตไม่ต่ำกว่าที่ถูกกำหนดไว้เช่นกัน รูปที่ 3.9 เป็นตัวอย่างการสร้างเมทริกซ์ $\mathbf{H}$ แบบอัลกอริทึม bit filling ซึ่งมีขนาด 4×7 และมีน้ำหนักของหลักเท่ากับ 3 และไม่ใช้เกิดเกิร์ตที่มีความยาวเท่ากับ 4 เลย โดยมีคำอธิบายดังต่อไปนี้ รูปที่ 3.9 (ก) บิตโนด c_0 และ c_1 มีน้ำหนักของหลักที่เท่ากับ 3 และทุกน้ำหนักต่อกับโนดพาริตีโดยที่ไม่ทำให้เกิดเกิร์ต เมื่อโนดบิต c_2 ถูกเพิ่มเข้ามา น้ำหนักที่ 1 โนดบิต c_2 จะถูกเชื่อมของกับพาริตีโนด f_6 ที่ว่างอยู่ ส่วนน้ำหนักที่ 2 ถูกเชื่อมกับพาริตีโนด f_1 ซึ่งยังไม่ทำให้เกิดเกิร์ต รูปที่ 3.9 (ข) เป็นการใส่น้ำหนักที่ 3 ของบิตโนด c_2 ซึ่งหากไม่ต้องการให้เกิดเกิร์ตที่มีขนาดเท่ากับ 4 ต้องเลือกเชื่อมต่อกับพาริตีโนด f_2, f_4 และ f_5 เท่านั้น กรณีนี้สมมุติว่าเลือกเชื่อมต่อกับพาริตีโนด f_4 ในรูปที่ 3.9 (ค) เมื่อโนดบิต c_3 ถูกเพิ่มเข้ามาและกำหนดให้มีน้ำหนักเท่ากับ 3 น้ำหนักแรกของโนดบิต c_3 เลือกไปเชื่อมกับโนดพาริตีใดก็ได้ที่มีน้ำหนักของโนดพาริตี (row weight) น้อยที่สุดและในที่นี้สมมุติว่าเลือกเชื่อมกับโนดพาริตี f_3 ในส่วนของน้ำหนักที่ 2 และน้ำหนักที่ 3 ถูกเชื่อมกับโนดพาริตี f_5 และ f_6 ซึ่งไม่ทำให้เกิดเกิร์ตเท่ากับ 4 เลยซึ่งเป็นไปตามรูปที่ 3.9 (ง) จะเห็นว่ากราฟแทนเนอร์ที่ได้มามีโนดบิตจำนวน 4 บิตและโนดพาริตีจำนวน 7 บิตและไม่มีเกิร์ตที่มีขนาดความยาวเท่ากับ 4 เลยตามที่ต้องการ

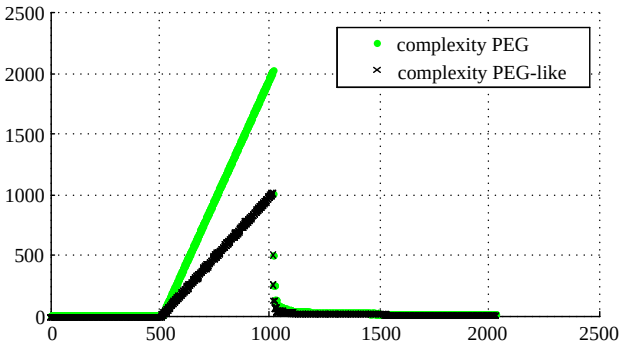
3.4 ผลการทดลองและการสรุปผล

ผลการทดลองในบทนี้ได้นำมาจากงานวิจัยที่เสนอการสร้าง $\mathbf{H}$ เมทริกซ์แบบ PEG-like อัลกอริทึม เพื่อลดความซับซ้อนในการสร้างเมทริกซ์ $\mathbf{H}$ แบบ อัลกอริทึม PEG ซึ่งมีดังต่อไปนี้



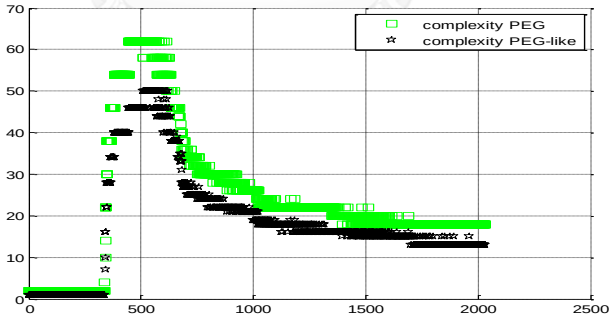
รูปที่ 3.10 BER ของอัลกอริทึม PEG กับ PEG-like

จากรูปที่ 3.10 แสดงผลในรูปของ BER (Bit-Error Rate) ระหว่างอัลกอริทึม PEG กับ PEG-like โดยใช้ช่องสัญญาณเป็นสัญญาณรบกวนแบบเกาต์สีขาวแบบบวก (AWGN : additive white Gaussian noise) ที่จำนวน k messages bit แบบไบนารี $a_k = \{0,1\}$ เข้ารหัส LDPC ได้คำรหัสจำนวน N บิต โดยได้เป็นสัญญาณแบบ $b_k = \{-1,1\}$ และ ณ ภาครับได้รับสัญญาณเป็น $y_k = b_k + n_k$ โดย n_k คือสัญญาณรบกวนแบบ AWGN ที่มีค่าเฉลี่ย (mean) เป็น 0 และมีความแปรปรวน (veriance) เท่ากับ σ^2 และที่ภาครับนี้เองถูกถอดรหัสด้วยรหัสแอลดีพีซีด้วยวิธีแบบซอฟต์ (message passing) และขนาดของ n_k นั้นแตกต่างกันไปได้แก่ 504 (สีแดง), 1008 (สีน้ำเงิน) และ 2032 (สีดำ) มีอัตรารหัสมีค่าเท่ากับ 0.5 และน้ำหนักของหลักมีค่าเท่ากับ 2 จากรูปนั้นจะเห็นได้ว่า BER แต่ละขนาดนั้นของทั้งอัลกอริทึม PEG และ PEG-like ล้วนให้ BER ที่ไม่แตกต่างกันมากนัก



รูปที่ 3.11 ความซับซ้อนของอัลกอริทึม PEG กับ PEG-like โดยมีน้ำหนักของหลักเท่ากับ 2

จากรูปที่ 3.11 นั้นแสดงความซับซ้อนของอัลกอริทึม PEG กับ PEG-like และเมทริกซ์ $\mathbf{H}$ ขนาด 1016×2032 โดยมีน้ำหนักของหลักเท่ากับ 2 โดยความซับซ้อนที่ว่านี้สามารถวัดได้จากจำนวนครั้งที่ทำการค้นหาเส้นทางระหว่างแถวและหลักในเมทริกซ์ $\mathbf{H}$ ซึ่งจะเห็นว่าในช่วงแรกนั้นความซับซ้อนของอัลกอริทึม PEG กับ PEG-like นั้นไม่ได้แตกต่างกันมากนัก อันเนื่องมากจากโนดพาริตียังไม่ได้ถูกต่อขึ้นมากไปกว่าความลึกที่ 0 การค้นหาจึงไม่ได้เกิดขึ้น แต่เมื่อกราฟแทนเนอร์ถูกสร้างให้มีความยาวมากขึ้น (มีระดับความลึกของการขยายกราฟย่อยแบบต้นไม้มากขึ้น) จะเห็นได้ว่าอัลกอริทึม PEG-like นั้นมีความซับซ้อนที่น้อยกว่าว่าอัลกอริทึม PEG



รูปที่ 3.12 ความซับซ้อนของอัลกอริทึ่ม PEG กับ PEG-like โดยมีน้ำหนักของหลักเท่ากับ 3

จากรูปที่ 3.12 แสดงความซับซ้อนของอัลกอริทึม PEG กับ PEG-like และเมทริกซ์ $\mathbf{H}$ ขนาด 1016×2032 โดยมีน้ำหนักของหลักเท่ากับ 3 จะเห็นว่าในช่วงแรกนั้นความซับซ้อนยังคงมีค่าใกล้เคียงกันเช่นด้วยกับรูปที่ 3.10 แต่ระยะนั้นจะสั้นลงอันเนื่องมาจากเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของหลักที่เท่ากับ 3 นั้นการเชื่อมต่อกันของกราฟแทนเนอร์จะเชื่อมต่อกันได้เร็วกว่าเมทริกซ์ $\mathbf{H}$ ที่มี

น้ำหนักของหลักที่เท่ากับ 2 แน่นอนว่าเมื่อกราฟแทนเนอร์ที่มีการเชื่อมต่อกันมากขึ้นอัลกอริทึม PEG-like นั้นก็จะให้ความซับซ้อนที่น้อยกว่า

บทที่ 4
การออกแบบเมทริกซ์พาริตีแบบ Quasi-Cyclic เพื่อให้มีเกิร์ตสูง

บทที่ 4 นี้นำเสนอผลงานในการออกแบบอัลกอริทึมสำหรับการสร้างเมทริกซ์ $\mathbf{H}$ ของรหัสแอลดีพีซีชนิดที่เรียกว่า “Quasi-Cyclic LDPC codes” โดยอัลกอริทึมที่ผู้เขียนวิทยานิพนธ์นำเสนอถูกออกแบบมาเพื่อทำให้เมทริกซ์ $\mathbf{H}$ แบบ Quasi-Cyclic นั้นมีเกิร์ตที่มีขนาดความยาวสูงที่สุด อนึ่งเพื่อศึกษาถึงผลกระทบของเกิร์ตที่มีต่อรหัสแอลดีพีซีเมื่อเมทริกซ์ $\mathbf{H}$ มีน้ำหนักของหลักเท่ากัน มีมิติที่เท่ากันแต่เกิร์ตมีขนาดที่แตกต่างกันกัน นอกจากนี้เมทริกซ์ $\mathbf{H}$ แบบ Quasi-Cyclic เองนั้นก็มีความน่าสนใจอันเนื่องจากถูกนำมาใช้ในหลากหลายในแอปพลิเคชันของระบบสื่อสาร เพราะเมทริกซ์ชนิดนี้มีความเหมาะสมในการใช้งานกับระบบฮาร์ดแวร์ (hardware) [13] ตัวอย่างของมาตรฐานการสื่อสารที่นำเมทริกซ์ $\mathbf{H}$ แบบ Quasi-Cyclic มาประยุกต์ใช้งานมีอาทิ มาตรฐาน IEEE 802.16e และ 802.11n เป็นต้น หัวข้อที่จะกล่าวถึงต่อไปจะเสนอเกี่ยวลักษณะและโครงสร้างของเมทริกซ์ $\mathbf{H}$ แบบ Quasi-Cyclic ดังต่อไปนี้

4.1 รหัสแอลดีพีซีแบบ Quasi-Cyclic

เมทริกซ์ $\mathbf{H}$ ของ Quasi-Cyclic LDPC codes นั้นมีลักษณะที่ประกอบไปด้วยอาร์เรย์ (array) ของเมทริกซ์เล็ก ๆ หลาย ๆ อาร์เรย์ ซึ่งเมทริกซ์เล็ก ๆ ที่ว่านี้มีชื่อเรียกว่า “circulant sub-matrices” หรือเมทริกซ์ย่อย ซึ่งเมทริกซ์ย่อยเหล่านี้จะมีขนาดเท่ากันคือ $p \times p$ จากที่มาของคำว่า “circulant sub-matrices” นั้นมาจากความหมายที่ว่าเป็นเมทริกซ์ที่ถูกหมุนมาจากเมทริกซ์เอกลักษณ์ (identity- matrix) โดยหมุนได้ตั้งแต่ 0 ถึง $p-1$ ส่วนเมทริกซ์ที่หมุน 0 ครั้งคือเมทริกซ์เอกลักษณ์และเมื่อหมุนครับ p ครั้งเมทริกซ์จะหมุนกลับมาที่เมทริกซ์เอกลักษณ์อีกครั้ง พิจารณามิติขนาด $M \times N$ ของเมทริกซ์ $\mathbf{H}$ ตามรูปที่ 4.1

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}^{a(0,0)} & \mathbf{I}^{a(0,1)} & . \; . & \mathbf{I}^{a(0,k-1)} \\ \mathbf{I}^{a(1,0)} & \mathbf{I}^{a(1,1)} & . \; . & \mathbf{I}^{a(1,k-1)} \\ . & . & . \; . & . \\ . & . & . \; . & . \\ \mathbf{I}^{a(j-1,0)} & \mathbf{I}^{a(j-1,1)} & . \; . & \mathbf{I}^{a(j-1,k-1)} \end{bmatrix}_{M \times N},$$

รูปที่ 4.1 เมทริกซ์ $\mathbf{H}$ ของ Quasi-Cyclic LDPC codes

j และ k เป็นเลขสมาชิกของเมทริกซ์ย่อยในแนวแถวและแนวหลักตามลำดับ ส่วนค่า $a(j,k)$ เป็นจำนวนครั้งที่หมุนของเมทริกซ์ย่อยและ

$$j \times p = M \tag{4.1}$$

$$k \times p = N \tag{4.2}$$

และ I^0 แสดงเมทริกซ์เอกลักษณ์ ส่วนลักษณะการหมุนสามารถเป็นไปได้ทั้งหมุนซ้ายและหมุนขวา ข้อสังเกตที่สำคัญของเมทริกซ์ย่อยคือถ้ารู้ถึงตัวเลขที่เมทริกซ์หมุน จะสามารถรู้เอลิเมนต์ทุกตัวในเมทริกซ์ย่อยนั้นได้ ซึ่งแสดงให้เห็นในรูปที่ 4.2

$$\mathbf{I}^0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{I}^1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \equiv \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

รูปที่ 4.2 ตัวอย่างเมทริกซ์ย่อยของเมทริกซ์พาริตีใน Quasi-Cyclic LDPC codes

I^0 นั้นแสดงเมทริกซ์เอกลักษณ์ซึ่งหากเป็นเมทริกซ์ย่อยที่หมุนทางขวา หลักต่อไปของ I^0 จะมีเลข 1 อยู่ที่แถวที่ 2 I^1 นั้นแสดงเมทริกซ์เอกลักษณ์ที่หมุน 1 ครั้ง (รูปที่ 4.2 หมุนขวา) หลักที่สองของ I^1 จะมีเลข 1 อยู่ที่แถวที่ 3 และหลักที่ 3 ซึ่งเป็นหลักสุดท้ายใน I^1 จะมีเลข 1 ที่แถวที่ 1 ตามรูปที่ 4.2 ทั้งนี้ในรูปเป็นเมทริกซ์ย่อย ที่มีขนาด 3×3

4.2 ตัวอย่างการสร้างเมทริกซ์ตัวถอดรหัสแบบ Quasi-Cyclic ที่น่าสนใจ

4.2.1 โครงสร้างเมทริกซ์พาริตีแบบอาร์เรย์

ปี ค.ศ. 2000 Fan [14] ได้พัฒนาเมทริกซ์ $\mathbf{H}$ ที่มีโครงสร้างแบบอาร์เรย์ขึ้น (array LDPC code) ซึ่งสามารถช่วยแก้ปัญหาเรื่องความซับซ้อนของการสร้างเมทริกซ์ได้ และมีสมรรถณะใกล้เคียงกับเมทริกซ์ $\mathbf{H}$ แบบสุ่มด้วย (อ้างอิงจากหนังสือ [15]) สำหรับโครงสร้างเมทริกซ์ $\mathbf{H}$ แบบแถวอาร์เรย์นี้จะเป็นไปตามรูปที่ 4.3 จะเห็นได้ว่าโครงสร้างของเมทริกซ์ $\mathbf{H}$ รูปแบบนี้เป็นโครงสร้างรหัสแบบปกติ (regular LDPC codes) อย่างไรก็ตามจะเห็นได้ว่าเมทริกซ์ $\mathbf{H}$ แบบอาร์เรย์จะไม่ได้อยู่ในรูปสมมาตร ดังนั้นการเข้ารหัสจึงต้องอาศัยกระบวนการกำจัดแบบเกาส์เซียนจากที่กล่าวไว้แล้วในบทที่ 2

$$H=\begin{bmatrix} I & I & I & I & I \\ I & I^{1} & I^{2} & \cdots & I^{k-1} \\ I & I^{2} & I^{4} & \cdots & I^{2(k-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I & I^{j-1} & I^{2(j-1)} & \cdots & I^{(j-1)(k-1)} \end{bmatrix}_{jp\times kp}$$

รูปที่ 4.3 โครงสร้างของเมทริกซ์พาริตีแบบอาร์เรย์

4.2.2 การสร้างเมทริกซ์พาริตีอาร์เรย์แบบปรับปรุง

จากบทความวิจัยของ Richardson [16] ที่เขียนไว้ว่า การเพิ่มประสิทธิภาพให้กับการเข้ารหัสที่มีความซับซ้อนแบบเชิงเส้นนั้นสามารถทำได้โดยการจัดรูปเมทริกซ์ $\mathbf{H}$ ให้อยู่ในรูปของสามเหลี่ยมด้านบน ซึ่งมาในปีค.ศ. 2002 Eleftheriou [17] จึงได้เสนอวิธีการสร้างเมทริกซ์ $\mathbf{H}$ ที่เรียกว่า MAC (Modified Array code) ขึ้น โดยมีโครงสร้างที่เป็นสามเหลี่ยมด้านบนเพื่อให้ง่ายต่อการเข้ารหัสโดยคำว่า Modified Array code นั้นหมายถึงการปรับปรุงโครงสร้างมาจากเมทริกซ์ $\mathbf{H}$ ที่มีโครงสร้างแบบอาร์เรย์ซึ่งโครงสร้างเมทริกซ์ $\mathbf{H}$ แบบ MAC จะมีโครงสร้างดังรูปที่ 4.4

$$H=\begin{bmatrix} I & I & I & I & I & I & I & I \\ 0 & I & I^{1} & \cdots & I^{j-2} & I^{j-1} & \cdots & I^{k-2} \\ 0 & 0 & I & \cdots & I^{2(j-3)} & I^{2(j-2)} & \cdots & I^{2(\mathrm{k}-3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & I & I^{j-1} & \cdots & I^{(\mathrm{j}-1)(\mathrm{k}-3)} \end{bmatrix}_{jp\times kp}$$

รูปที่ 4.4 โครงสร้างของเมทริกซ์ $\mathbf{H}$ แบบ MAC

สำหรับการเข้ารหัสโดยใช้โครงสร้างเมทริกซ์ $\mathbf{H}$ แบบ MAC มีลักษณะดังต่อไปนี้ จากรูปที่ 4.4 ให้ m เป็นเซตของบิตข้อมูลที่ $\boldsymbol{m}=\{m_1,m_2,...,m_k\}$ และ p เป็นเซตของบิตพาริตีที่ $p=\{p_1,p_2,...,p_{n-k}\}$ การคำนวนบิตพาริตีจะใช้สมการดังต่อไปนี้

$$p_9+m_1+m_5=0 \tag{4.3}$$

$$p_8+m_3+m_4=0 \tag{4.4}$$

$$p_7+m_2+m_6=0 \tag{4.5}$$

$$p_6+p_8+m_1+m_6=0 \tag{4.6}$$

$$p_5+p_7+m_3+m_5=0 \tag{4.7}$$

$$p_4 + p_9 + m_2 + m_4 = 0 \quad (4.8)$$

$$p_3 + p_6 + p_9 + m_3 + m_6 = 0 \quad (4.9)$$

$$p_2 + p_5 + p_8 + m_2 + m_5 = 0 \quad (4.10)$$

$$p_1 + p_4 + p_7 + m_1 + m_4 = 0 \quad (4.11)$$

คำรหัสที่ได้ออกมานั้นจะอยู่ในรูปสมการที่ (4.12)

$$c = [\mathrm{p} \mid \mathrm{m}] \quad (4.12)$$

นอกจากนี้จากวิธีการสร้างเมทริกซ์ $\mathbf{H}$ แบบ MAC ยังถูกนำไปปรับปรุงสำหรับการสร้างเมทริกซ์ $\mathbf{H}$ ที่มีขนาดใหญ่ขึ้น เสนอโดย Singhaudom [18] ซึ่งเรียกว่า เมทริกซ์อาร์เรย์แบบปรับปรุง (interleaved modified array code) หรือ IMAC อีกด้วย

4.2.3 อัลกอริทึมการสร้างเมทริกซ์พาริตีแบบที่ใช้เมทริกซ์แบบ Magic Square

เมทริกซ์ $\mathbf{H}$ แบบที่ใช้โครงสร้างของเมทริกซ์แบบ Magic Square เป็นการสร้างเมทริกซ์ $\mathbf{H}$ แบบ Quasi-Cyclic โดยใช้คุณสมบัติของเมทริกซ์แบบ magic square เข้ามาช่วยในการหมุนของเมทริกซ์ย่อย การสร้างด้วยโครงลักษณะนี้ถูกพัฒนาขึ้นโดย Chutima Prasartkaew และ Somsak Choomchuay [16] ในปี ค.ศ. 2012 โดยตัวอย่างโครงสร้างของ $\mathbf{H}$ เมทริกซ์เป็นไปตามรูปที่ 4.5

$$H = \begin{bmatrix} I & I & I & I & I & I & I & I & I \\ 0 & I & X & X & X & X & X & X & X \\ 0 & 0 & I & X & X & X & X & X & X \end{bmatrix}_{jp \times kp}$$

รูปที่ 4.5 ตัวอย่างโครงสร้างของ $\mathbf{H}$ เมทริกซ์โดยใช้คุณสมบัติของ magic square

จากรูปที่ 4.5 ตัวแปร X ที่เพิ่มเข้ามานั้นคือตัวเลขที่ต้องการจะใช้ในการหมุนเมทริกซ์ย่อยซึ่งจะนำมาจากเมทริกซ์แบบ magic square วิธีการนี้ถูกเสนอสร้างขึ้นด้วยกัน 3 วิธีดังต่อไปนี้

1. ใช้ตัวเลขในแต่ละแถวของเมทริกซ์แบบ magic square ที่เลขแต่ละตัวในแถวมีค่าไม่เกิน p หากแถวใดมีตัวเลขที่เกินค่า p ให้ข้ามแถวนั้นไป

2. เริ่มใส่เลข 1 ที่อยู่ในเมทริกซ์แบบ magic square และใส่เลขเรียงไปเรื่อย ๆ นับจากตำแหน่งที่มีเลข 1 อยู่นั้น จากซ้ายไปขวาและบนลงล่าง หากมีค่าใดเกินกว่า p ให้ข้ามเลขตัวนั้นไป
3. แบบที่ 3 นี้จะมีโครงสร้างแตกต่างไปจากเดิมและเป็นไปตาม รูปที่ 4.6 ซึ่งมีคำอธิบายดังนี้

$$H=\begin{bmatrix} I & I & I & I & I & I & I & I & I \\ 0 & I & X & X & X & X & X & X & X \\ 0 & 0 & I & \beta_1 & Y_1 & Y_2 & \beta_2 & Z_1 & Z_2 \end{bmatrix}_{jp\times kp}$$

รูปที่ 4.6 ตัวอย่างโครงสร้างของเมทริกซ์พาริตีโดยใช้คุณสมบัติของ magic square แบบที่ 3

- ค่า β_1 จะมีค่าเท่ากับ $p/2$ ถ้าค่า p เป็นเลขคู่หรือเท่ากับ $(p-1)/2$ ถ้าค่า p เป็นเลขคี่ ถ้าในเมทริกซ์มีการใช้ค่านี้ไปแล้ว ให้ใช้เลขที่ใกล้เคียงที่สุด
- ค่า Y_1 และค่า Y_2 ให้ใช้ค่า β_1-1 และ β_1-2 ตามลำดับ
- ค่า β_2 จะมีค่าเท่ากับ $\beta_1/2$ ถ้าค่า β_1 เป็นเลขคู่หรือเท่ากับ $(\beta_1-1)/2$ ถ้าค่า β_1 เป็นเลขคี่

ถ้าในเมทริกซ์มีการใช้ค่าที่คำนวนมาได้ไปแล้ว ให้ใช้เลขที่ใกล้เคียงที่สุดกับค่า Z_1, Z_2 และค่า β_2-1 และ β_2-2 ตามลำดับ เพื่อใส่ลงไปในที่ว่างดังกล่าว

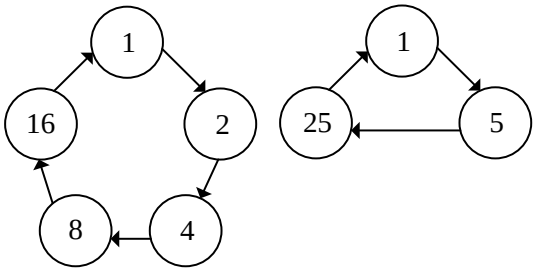
4.2.4 การสร้างเมทริกซ์พาริตีโดยอาศัยคุณสมบัติของโครงสร้างแบบ SFT

หัวข้อนี้เป็นการเสนอวิธีการเมทริกซ์ $\mathbf{H}$ แบบ Quasi-Cyclic โดยใช้ประโยชน์จากกราฟแทนเนอร์ของไซดาระ-ฟูตะ (Sridhara-FujaTanner) มาช่วยในการสร้างเมทริกซ์ [17] ซึ่งมีกรรมวิธีดังต่อไปนี้ พิจารณาในกรณีที่ $j=3$ และ $k=9$ และ $p\geq jk$

$$H=\begin{bmatrix} I^{1} & I^{a} & I^{a^2} & \cdots & I^{a^{k-1}} \\ I^{b} & I^{a\times b} & I^{a^2\times b} & \cdots & I^{a^{k-1}\times b} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ I^{b^{j-1}} & I^{a\times b^{j-1}} & I^{a^2\times b^{j-1}} & \cdots & I^{a^{k-1}\times b^{j-1}} \end{bmatrix}_{pj\times kj}$$

รูปที่ 4.7 โครงสร้างของเมทริกซ์พาริตีแบบ QC-LDPC codes โดยใช้โครงสร้างแบบ Sridhara-FujaTanner

สัญลักษณ์ a และ b จะมีค่าอยู่ใน $a,b \in \{2,3,...,p-1\}$ และทั้ง a และ b มีขอบเขตอยู่ใน $GF(p)$ ซึ่งการพิจารณานั้นจะต้องสร้างวงแหวนทั้งขนาด a และ b ขึ้น ซึ่งจะขอยกตัวอย่างตามรูปที่ 4.8



(ก) **(ข)**

รูปที่ 4.8 วงแหวนโครงสร้างแบบ Sridhara-FujaTanner (ก) วงแหวนขนาด 5 (ข) วงแหวนขนาด 3

การสร้างเมทริกซ์ $\mathbf{H}$ แบบ SFT นั้นการเลือก a และ b ก็มาจากขนาดของวงแหวนว่าจะใช้ขนาดใดให้เป็นบล็อกแถวหรือบล็อกหลักในเมทริกซ์ $\mathbf{H}$ แบบ QC-LDPC codes

4.3 อัลกอริทึมการออกแบบเมทริกซ์พาริตีที่นำเสนอ

หัวข้อนี้คือผลงานวิจัยที่เสนอเกี่ยวกับวิธีสร้างเมทริกซ์ $\mathbf{H}$ แบบ Quasi-Cyclic LDPC codes ซึ่งมีกรรมวิธีดังต่อไปนี้ เช่นเดิม พิจารณาในกรณีที่ $j=3$ และ $k=9$ และ $p \geq jk$

1. สร้าง $j \times k$ เมทริกซ์ $\bar{\mathbf{H}}$ จาก $j=3$ และ $k=9$ เมทริกซ์ $\bar{\mathbf{H}}$ จะได้ดังตารางที่ 4-1 (ก) ที่ j และ k เป็นจำนวนของบล็อกแถวและบล็อกหลักตามลำดับ R เป็นเลขของการหมุนของเมทริกซ์ย่อยที่แบบสุ่มและ Z R เป็นเลขของการหมุนของเมทริกซ์ย่อยที่กำลังจะใส่ลงไป จากเมทริกซ์ย่อยในแต่ละบล็อกที่มีขนาด $p \times p$ ดังนั้นขนาดของเมทริกซ์ $\mathbf{H}$ จะมีขนาด $jp \times kp$

ตารางที่ 4-1 ตัวอย่าง $\bar{\mathbf{H}}$ ขนาด 3×9 หลังกระบวนการเสร็จแล้ว

block-column index

block-row index $\bar{\mathbf{H}}$	1	2	3	4	5	6	7	8	9
1	*R*	*R*	*R*	*R*	*R*	*R*	*R*	*R*	*R*
2	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*
3	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*

(ก)

block-column index

block-row index $\bar{\mathbf{H}}$	1	2	3	4	5	6	7	8	9
1	19	9	44	17	28	8	31	13	32
2	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*
3	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*

(ข)

2. สุ่มตัวเลขตั้งแต่ 1 ถึง $p-1$ แทนที่ทุกตัวใน R ตัวอย่างดังตารางที่ 4-1 b)

3. ขั้นตอนต่อไปคือการใส่ตัวเลขเพื่อแทนที่ Z ทั้งหมด โดยใช้ค่าที่จะเป็นไปได้ทั้งหมดของจำนวนแถวทั้งหมดที่ต้องการใส่ในแต่ละหลัก ตัวอย่างเช่น ตารางที่ 4.1 (ข) แถวแรกในแต่ละหลักของเมทริกซ์ $\bar{\mathbf{H}}$ ถูกใส่ตัวเลขลงไปแล้ว ฉะนั้นทุกหลักในเมทริกซ์ $\bar{\mathbf{H}}$ จะเหลือ 2 แถวที่ยังไม่ได้ใส่ตัวเลข จากตัวเลขตั้งแต่ 0 ถึง $p-1$ จะมีความเป็นไปได้ทั้งหมดที่จะจับคู่กัน 2 ตัวดังนี้ $\binom{p}{1}^2$ ซึ่งจะได้สูตรทั่วไปดังนี้ ซึ่งทั้งหมดจะอยู่ในเซตของ P_{fc}

$$\text{data patterns} = \binom{p}{1}^{j-1} \tag{4.13}$$

4. ใส่ทุกค่าในเซตของ P_{fc} จากหลักที่แรกไปจนหลักสุดท้าย ในหลักใด ๆ ที่แทนค่าอยู่นั้น เมื่อแทนแต่ละค่าให้หาเกิร์ตของแต่ละค่าแทนแทนลงไปด้วย เมื่อค่าใดให้เกิร์ตสูงที่สุดให้เลือกค่านั้นแทนลงในหลักที่กำลังกระทำอยู่ในเมทริกซ์ $\bar{\mathbf{H}}$ เลย เมื่อเริ่มกระทำหลักต่อไปในเมทริกซ์ $\bar{\mathbf{H}}$ ค่าที่ถูกใส่ไปในหลักอื่น ๆ แล้วนั้นให้ตัดออกทันที เพราะถ้าค่าซ้ำกันในกระบวนการวัดเกิร์ตก็จะเกิดเกิร์ตที่มีค่าเท่ากับ 4 กระบวนการกระทำในแถวแรก ๆ ของเมทริกซ์ $\bar{\mathbf{H}}$ นั้นอาจไม่สามารถหาเกิร์ตได้เลยเพราะเมทริกซ์ยังมีตัวเลขที่เท่ากับ 1 จึงไม่เพียงพอที่จะทำให้เกิดเกิร์ต เหตุการณ์เช่นนี้ให้ใส่ค่าเกิร์ตของค่าในเซตของ P_{fc} ที่หาค่า

ไม่ได้ให้มีค่าเท่ากับ ∞ ซึ่งมีค่าสูงที่สุด ตัวอย่างหลังจากที่ใส่ตัวเลขการหมุนของเมทริกซ์ย่อยเสร็จแล้วแสดงดังตารางที่ 4-2

ตารางที่ 4-2 ตัวอย่างเมทริกซ์ $\bar{\mathbf{H}}$ ขนาด 3×9 หลังกระบวนการเสร็จแล้ว

	block-column index								
block-row index	$\bar{\mathbf{H}}$	1	2	3	4	5	6	7	8	9
	1	19	9	44	17	28	8	31	13	32
	2	0	2	3	5	10	7	29	20	18
	3	0	3	5	14	20	12	36	23	45

เพื่อความเข้าใจที่มากยิ่งขึ้นจะขออธิบายเพิ่มเติมเกี่ยวตัวเลขการหมุนของเมทริกซ์ย่อยซึ่งจะมีความสัมพันธ์กับอันดับของหลักแรกในเมทริกซ์ย่อยที่มีค่าเท่ากับ 1 จากรูปที่ 4.9 แสดงแถวแรกของเมทริกซ์ย่อย ที่ $p=3$ ซึ่งแทนด้วยเลข 1 ถึง 3 ซึ่งหากจำนวนแถวที่ยังไม่ได้ใส่เลขการหมุนของเมทริกซ์ย่อยมีค่าเท่ากับ 2 เซต P_{fc} จะมีจำนวนเท่ากับ 9 ตามที่เห็นรูปที่ 4.9

$$
\begin{array}{l}
1^{st}\text{ column of} \\
\text{each submatrix} \rightarrow \\
\text{in the } 2^{nd}\text{ row}
\end{array}
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
$$

$$
\begin{array}{l}
1^{st}\text{ column of} \\
\text{each submatrix} \rightarrow \\
\text{in the } 3^{rd}\text{ row}
\end{array}
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
$$

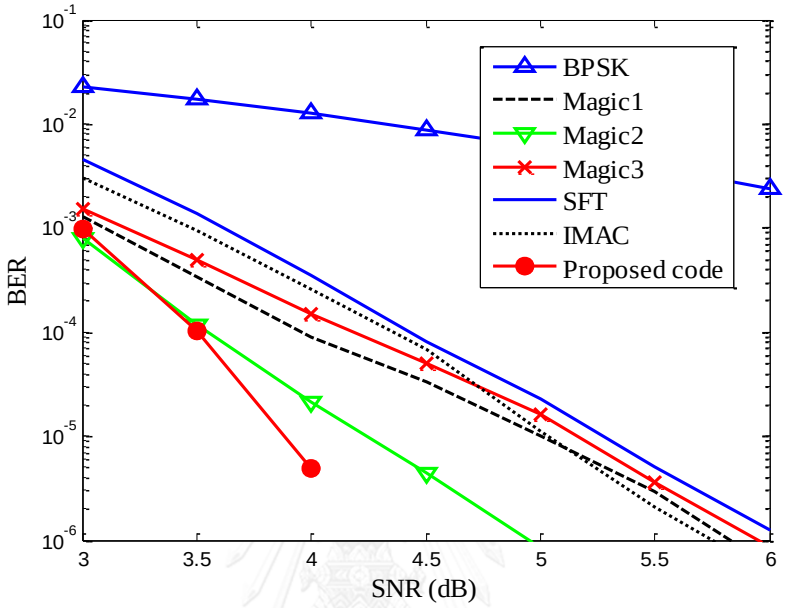
รูปที่ 4.9 คือแถวแรกของเมทริกซ์ย่อยที่มีขนาด $p=3$

4.4 ผลการทดลองและการสรุปผล

ในส่วนของผลการทดลองจะแสดงผลจะอยู่ในรูปของ BER (Bit-Error Rate) โดยใช้ช่องสัญญาณเป็นสัญญาณรบกวนแบบเกาส์สีขาวแบบบวกที่จำนวน k บิตข้อมูลและเป็นแบบไบนารี $a_k=\{0,1\}$ การเข้ารหัสแอลดีพีซีได้คำรหัสจำนวน N บิต โดยได้เป็นสัญญาณแบบ $b_k=\{-1,1\}$ และ ณ ภาครับได้รับสัญญาณเป็น $y_k=b_k+n_k$ โดย n_k คือสัญญาณรบกวนแบบ แบบเกาส์สีขาวแบบบวกที่มีค่าเฉลี่ยเท่ากับ 0 และมีความแปรปรวนเท่ากับ σ^2 และที่ภาครับนี้เองถูกถอดรหัสด้วยรหัสแอลดีพีซีด้วยวิธีแบบซอฟต์ (message passing) และการส่งข้อมูลเป็นจำนวน 50000 บล็อก และการถอดรหัสวนซ้ำจำนวน 50 รอบ วัดค่าไปจนกว่ามีจำนวนบิตผิดพลาด 500 บิต ซึ่ง

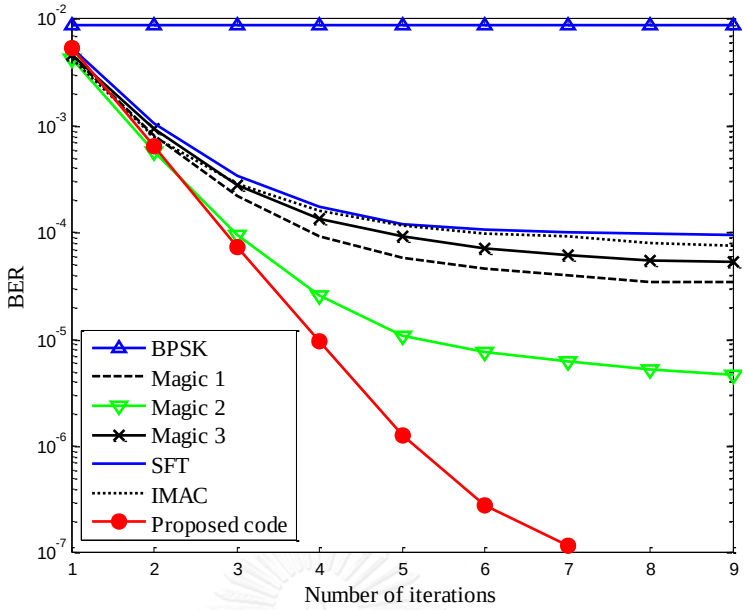
$$SNR = 10\log_{10}\left(E_b / N_0\right) \tag{4.14}$$

สมการ (4.14) SNR คืออัตราส่วนกำลังของสัญญาณ (E_b) ต่อกำลังของสัญญาณรบกวน (N_0)



รูปที่ 4.10 BER ระหว่างเมทริกซ์ที่มาจากอัลกอริทึมที่นำเสนอและอัลกอริทึมแบบ magic square, แบบ SFT และ IMAC

จากรูปที่ 4.10 แสดงค่า BER ของอัลกอริทึมที่ถูกนำเสนอและอัลกอริทึมต่าง ๆ โดยเป็นเมทริกซ์ที่มีขนาด 171×513 ซึ่งอัลกอริทึมต่าง ๆ ที่ว่ามานี้คือแบบ magic square ทั้ง 3 แบบ SFT และ IMAC จะเห็นได้ว่า อัลกอริทึมที่นำเสนอให้ประสิทธิภาพลบบิตผิดพลาดได้ดีกว่าอัลกอริทึมแบบอื่น ๆ



รูปที่ 4.11 BER ระหว่างเมทริกซ์ของอัลกอริทึมที่นำเสนอและอัลกอริทึมแบบ magic square, แบบ SFT และ IMAC ทั้ง 10 รอบที่ SNR = 4.5

จากรูปที่ 4.11 แสดงค่า BER ระหว่างเมทริกซ์จากอัลกอริทึมที่นำเสนอและอัลกอริทึมแบบ magic square, แบบ SFT และ IMAC ในการถอดรหัสทั้ง 10 รอบที่ $\mathbf{SNR = 4.5}$ ซึ่งแสดงให้เห็นว่าเมทริกซ์ $\mathbf{H}$ จากอัลกอริทึมที่นำเสนอสามารถลบความผิดพลาดได้อย่างรวดเร็วโดยใช้เพียงแค่ 7 รอบก็สามารถลบความผิดพลาดได้ทั้งหมด ซึ่งการสร้างเมทริกซ์แบบอื่น ๆ แม้จะมีการวนซ้ำถึง 10 รอบ หากแต่ยังสามารถพบข้อผิดพลาดอยู่ ที่เป็นเช่นนี้เป็นเพราะเกิร์ตของเมทริกซ์จากอัลกอริทึมที่นำเสนอนั้นให้เกิร์ตที่สูงกว่าการสร้างเมทริกซ์ $\mathbf{H}$ ด้วยอัลกอริทึมแบบ magic square, แบบ SFT และ IMAC นั้นเอง ซึ่งสามารถดูได้จากตารางที่ 4.2

ตารางที่ 4-3 ตารางการเปรียบเทียบเกิร์ตของเมทริกซ์จากอัลกอริทึมที่นำเสนอและเมทริกซ์ที่สร้างด้วยด้วยอัลกอริทึมแบบ magic square, แบบ SFT และ IMAC

Parameter	Minimum Girth			
	Magic	SFT	IMAC	Proposed
$p = 57, j = 3, k = 9$	6	6	6	8
$p = 61, j = 3, k = 10$	6	6	6	8

บทที่ 5
การสร้างเมทริกซ์พาริตีแบบที่มีน้ำหนักของทุกหลักเท่ากับสอง

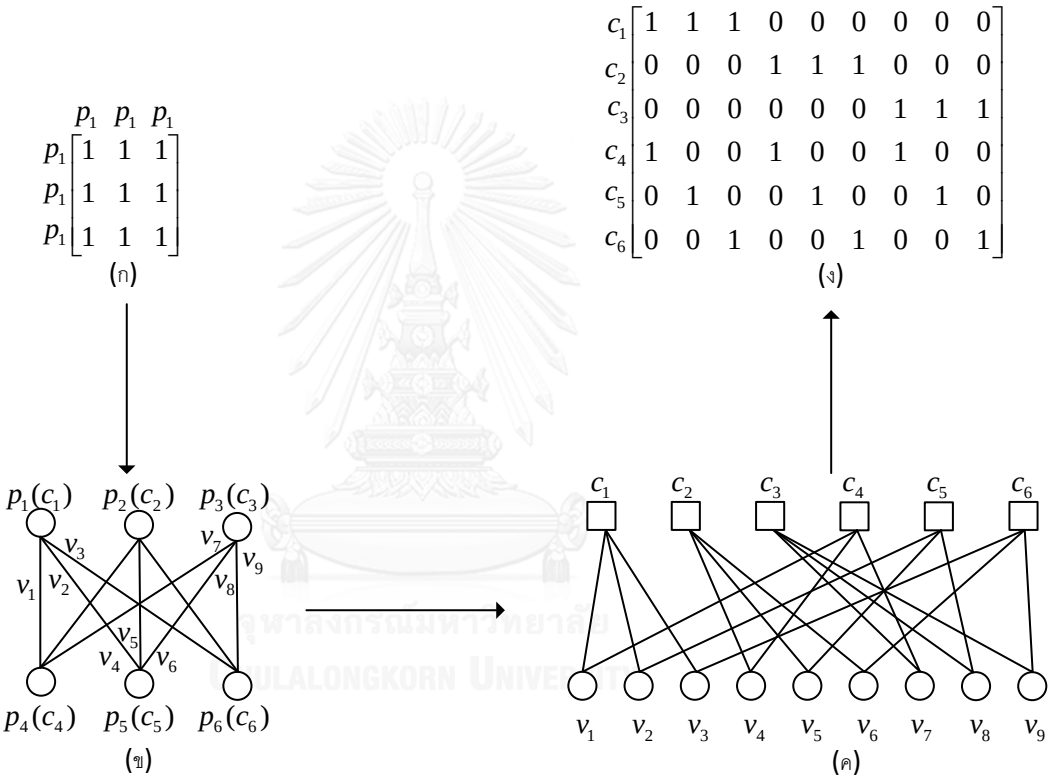
บทนี้ได้เสนอวิธีการสร้างเมทริกซ์ $\mathbf{H}$ ชนิดที่เรียกว่าเป็นเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของทุกหลัก (column weight) เท่ากับ 2 โดยอัลกอริทึมที่นำเสนอจะสามารถสร้างเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของทุกหลักเท่ากับ 2 และสามารถกำหนดขนาดของเกิร์ตที่ต่ำที่สุดในเมทริกซ์ $\mathbf{H}$ ได้ตามต้องการ อย่างไรก็ดีเมทริกซ์ $\mathbf{H}$ ชนิดที่มีน้ำหนักของหลักเท่ากับ 2 นี้มีคุณลักษณะพิเศษที่น่าสนใจดังที่จะกล่าวต่อไปนี้

Robert Gallagher ผู้ที่คิดค้นรหัสแอลดีพีซีนั้นได้พิสูน์ไว้ใน [1] ว่าเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของหลักเท่ากับ 2 นั้นการเพิ่มขึ้นของระยะแฮมมิง (minimum Hamming distance) จะเป็นไปอย่าง logarithmically เมื่อเทียบกับขนาดของคำรหัส ในขณะที่เมทริกซ์ $\mathbf{H}$ มีน้ำหนักของหลักที่มากกว่า 2 การเพิ่มขึ้นของระยะแฮมมิงจะเป็นไปอย่าง exponentially เมื่อเทียบกับขนาดของคำรหัส อย่างไรก็ตาม [19] และ [20] ได้พิสูจน์ว่าเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของหลักเท่ากับ 2 มีความซับซ้อนที่น้อยกว่าและยังมีประสิทธิภาพอย่างมากเมื่อถูกนำไปใช้ในช่องสัญญาณแบบวนกลับ (partial response channel) มากไปกว่านั้นยังให้ประสิทธิภาพที่สูงถ้าเป็นเมทริกซ์ $\mathbf{H}$ แบบที่เป็นนอนไบนารี (non-binary) [21] จากคุณสมบัติพิเศษที่น่าสนใจดังกล่าวได้ทำให้การออกแบบเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของหลักเท่ากับ 2 จึงมีความน่าสนใจ

ในอดีตที่ผ่านมามีผู้วิจัยมากมายให้ความสนใจในการออกแบบเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของหลักเท่ากับ 2 อาทิ Malema และ Liebelt [22] ซึ่งสร้างเมทริกซ์ $\mathbf{H}$ ที่มาจากกราฟเคจ (distance-graphs or cages) [23] , [24], โดยให้จุด (vertex) และเส้น (edge) แทนที่ด้วยแถวและหลักในเมทริกซ์ $\mathbf{H}$ ซึ่งเกิร์ตที่ได้ในเมทริกซ์ $\mathbf{H}$ จะมีขนาดเป็น 2 เท่าของกราฟเคจ ต่อมา Venkiah และคณะ [8] ได้เสนอวิธีการสร้างการสร้างกราฟเคจที่มีเกิร์ตสูงโดยใช้อัลกอริทึม PEG เพื่อนำกราฟเคจนี้มาสร้างเมทริกซ์ $\mathbf{H}$ ที่มีที่มีน้ำหนักของหลักเท่ากับ 2 ซึ่งวิธีนี้เรียกว่าอัลกอริทึม randomized PEG ในขณะที่ปี 2011 Tao และคณะ [25] ได้เสนอวิธีสร้าง (k,k) QC-LDPC codes โดยใช้อัลกอริทึมการค้นหา (search algorithm) เพื่อให้มีเกิร์ตสูง จากนั้นจึงแปลง (k,k) QC-LDPC codes เป็นเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของหลักเท่ากับ 2 ซึ่งจะทำให้ผลลัพธ์ของเมทริกซ์ $\boldsymbol{H}$ ที่ได้มานั้นมีเกิร์ตเป็น 2 เท่าของ (k,k) QC-LDPC codes ตัวเดิมโดยใช้วิธีแบบ finite geometry ซึ่งวิธีการก็จะได้กล่าวถึงดังต่อไปนี้

5.1 ตัวอย่างการสร้างเมทริกซ์พาริตีโดยใช้วิธีแบบ Finite Geometry

ในหัวข้อนี้จะเป็นการนำเสนอการสร้างเมทริกซ์ $\mathbf{H}$ โดยใช้วิธีแบบ finite geometry อันเนื่องมาจากอัลกอริทึมที่นำเสนอมีส่วนที่คล้ายคลึงกับการสร้าง $\mathbf{H}$ เมทริกซ์โดยใช้วิธีแบบ finite geometry รวมไปถึงงานต่าง ๆ ที่น่าสนใจที่ผู้วิจัยท่านอื่นที่นำเสนอเกี่ยวกับการสร้าง $\mathbf{H}$ เมทริกซ์ที่มีน้ำหนักของหลักเท่ากับ 2 นั้น ก็ได้มีการนำวิธีแบบ finite geometry มาใช้ในการสร้างเมทริกซ์ $\mathbf{H}$ ด้วยเช่นกัน



รูปที่ 5.1 ตัวอย่างการสร้างเมทริกซ์ $\mathbf{H}$ โดยใช้ finite geometry

เมทริกซ์ $\mathbf{H}$ โดยใช้วิธีแบบ finite geometry นั้นเริ่มต้นจากการนำเสนอของ Kou และคณะ โดยมีคำอธิบายดังนี้ กราฟ G ที่เป็นแบบ finite geometry จะประกอบไปด้วย m จุดและ n เส้น มีโครงสร้างดังต่อไปนี้

- แต่ละเส้นจะประกอบด้วย ρ จุด
- จะมีเพียงเส้นเดียวเท่านั้นที่อยู่ระหว่างจุด 2 จุด

- แต่ละจุดจะเชื่อมกับเส้นจำนวน γ เส้น
- ทุก ๆ คู่ของเส้นจะเชื่อมกันด้วยจุดพียง 1 จุด

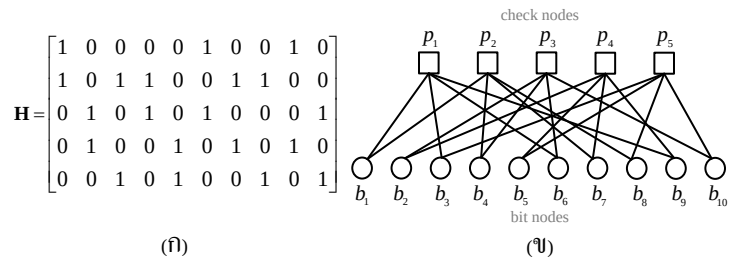
จากกราฟ G จะสามารถนำมาแปลงให้อยู่ในรูปของเมทริกซ์ $\mathbf{H}$ ที่ $\mathbf{H}=[h_{i,j}]$ โดยแต่ละแถวและหลักของเมทริกซ์ $\mathbf{H}$ นั้นสัมพันธ์กับจุดและเส้นบน G พิจารณาที่ $\mathbf{H}_1$ และ G_1 คือ เมทริกซ์ $\mathbf{H}$ และกราฟแทนเนอร์ตามลำดับ ซึ่งหากแปลง G_1 โดยใช้ finite geometry เป็น G_2 จะได้ว่าเกิร์ตของ G_2 มีค่าเป็น 2 เท่าของเกิร์ต G_1

จากรูป 5.1 (ข) คือกราฟของแทนเนอร์ของรูป 5.1 (ก) ซึ่งจะเห็นว่ามีเกิร์ตเท่ากับ 4 แต่หากมองรูปที่ 5.1 (ข) เป็นแบบ finite geometry ที่แต่ละจุดจะแทนด้วยแต่ละโนดพาริตีและแต่ละเส้นแทนด้วยแต่ละโนดบิต หรืออีกนัยหนึ่งคือแต่ละจุดแทนด้วยแต่ละหลักของเมทริกซ์ $\mathbf{H}$ และแต่ละเส้นแทนด้วยแต่ละหลักของเมทริกซ์ $\mathbf{H}$ ซึ่งกราฟแทนเนอร์ใหม่ที่ได้จะเป็นไปตามรูปที่ 5.1 (ค) ซึ่งมีเมทริกซ์ $\mathbf{H}$ ตามรูปที่ 5.1 (ง)

5.2 นิยามและความสำคัญ

หัวข้อนี้จะอธิบายถึงนิยามและความสัมพันธ์ของเมทริกซ์ $\mathbf{H}$ และกราฟแทนเนอร์อีกครั้งเพื่อจะได้มีความเข้าใจในอัลกอริทึมที่นำเสนอได้ดียิ่งขึ้น แม้ว่านั้นหัวข้อที่ผ่านมาจะได้นิยามไว้บ้างแล้ว

พิจารณาอีกครั้งที่เมทริกซ์ $\mathbf{H}$ ขนาด $M\times N$ ซึ่ง M และ N คือจำนวนโนดพาริตีและโนดบิตตามลำดับ เซตของโนดพาริตีคือ $\{p_1, p_2,..., p_M\}$ และเซตของบิตโนดนั้นถูกแทนที่ด้วย $\{b_1, b_2,..., b_M\}$ ให้ d_{p_i} คือจำนวนน้ำหนักของโนดพาริตีโนดที่ i ส่วน d_{p_i} คือจำนวนน้ำหนักของโนดบิตโนดที่ j พิจารณารูปที่ 5.2



(ก) (ข)

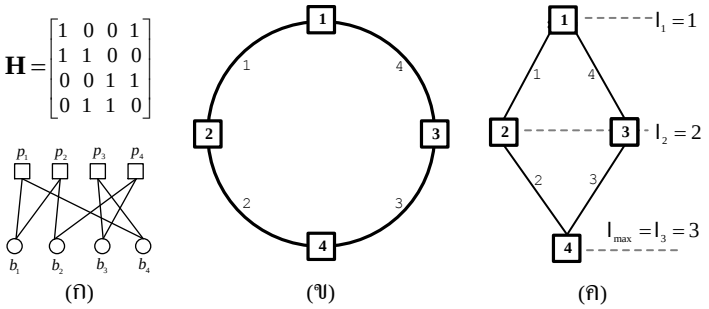
รูปที่ 5.2 (ก) ตัวอย่างของเมทริกซ์พาริตีขนาด 5×10 และ (ข) ตัวอย่างกราฟของแทนเนอร์จากรูป (ก)

ตัวอย่างของเมทริกซ์ $\mathbf{H}$ ขนาด 5×10 ในรูปที่ 5.2 (ก) และตัวอย่างกราฟแทนเนอร์ในรูปที่ 5.2 (ข) ซึ่งจะเห็นได้ว่าในรูปที่ 5.2 (ก) นั้นน้ำหนักของโนดบิตทุกโนดมีน้ำหนักเป็น 2 ซึ่งก็คือเมทริกซ์ $\mathbf{H}$ ที่มีหลักเป็น 2 นั้นเอง นอกจากนี้น้ำหนักของทุก ๆ โนดพาริตียังเท่ากันอีกด้วยคือมีน้ำหนักที่เท่ากับ 4

จากที่ได้กล่าวไปแล้วว่าอัลกอริทึมที่นำเสนอนี้มีหลักการที่คล้ายคลึงกับหลักการของ finite-geometry ซึ่งจะกล่าวดังต่อไปนี้ ทุก ๆ โนดพาริตี p_i จะถูกนิยามให้เป็นจุด (vertex) และทุก ๆ โนดบิต b_j ถูกนิยามโดยเส้น (edge) ซึ่งอัลกอริทึมที่นำเสนอนี้แรกเริ่มกราฟในโครงสร้างของ finite geometry จะถูกสร้างในลักษณะวงแหวนซึ่งมีตัวอย่างดังต่อไปนี้ พิจารณาเมทริกซ์ $\mathbf{H}$ ที่มีเกิร์ตเท่ากับ 8 ในรูปที่ 5.3 ก) ซึ่งสามารถสร้างให้อยู่ในรูปของกราฟแบบวงแหวนในรูปแบบของ finite-geometry ที่ทุก ๆ โนดพาริตีแทนที่ด้วยสี่เหลี่ยม และทุก ๆ โนดบิตแทนที่ด้วยเส้น โดยหมายเลขที่กำกับบนสี่เหลี่ยมและที่อยู่ข้างเส้นนั้นบอกถึงเลขอันดับของแถวและหลักในเมทริกซ์ $\mathbf{H}$ ตามลำดับ ดังนั้นถ้าขยายกราฟแบบรูปต้นไม้จากสี่เหลี่ยมใด ๆ ของรูป ในรูปที่ 5.3 ข) จะได้จำนวนขั้น (layer) หรือ l เท่ากับ $3l$ โดยที่โนดพาริตีที่ 1 อยู่ที่ l_1 และโดยที่โนดพาริตีที่ 4 อยู่ที่ l_3 ซึ่งเป็นขั้นสุดท้ายนี้ขอนิยามให้เป็น $l_{\max}$ ทั้งหมดนี้แสดงอยู่ใน 5.3 ค) จากรูปวงแหวนตั้งต้นนี้เกิร์ตที่ได้จะดีจำนวนเท่ากับ

$$g \le 2|P_m| \tag{5.1}$$

กำหนดให้เกิร์ตนั้นถูกนิยามโดย g และ P_M คือจำนวนของสี่เหลี่ยมนั่นเอง (รูปที่ 5.3 ข) จำนวนสี่เหลี่ยมเท่ากับ 4)



(ก) (ข) (ค)

รูปที่ 5.3 (ก) เมทริกซ์พาริตีขนาด 4×4 และกราฟของแทนเนอร์ (ข) วงแหวนเริ่มต้นที่ได้มาจากรูปที่ 5.3 (ก) และ (ค) ผลของการขยายกราฟในรูปแบบต้นไม้จากรูปที่ 5.3 (ข)

5.3 การสร้างเมทริกซ์พาริตีแบบที่น้ำหนักของทุกหลักเท่ากับสองด้วยอัลกอริทึมที่นำเสนอ

ต่อไปเป็นการนำเสนองานวิจัยที่ได้ออกการสร้างเมทริกซ์ $\mathbf{H}$ ที่มีน้ำหนักของหลักเท่ากับ 2 ซึ่งมีข้อดีคือสามารถที่จะสร้างเมทริกซ์ $\mathbf{H}$ ที่สามารถกำหนดขนาดของเกิร์ตที่ต่ำที่สุดได้ในเมทริกซ์ $\mathbf{H}$ โดยกระบวนทั้งหมดนั้นมีดังต่อไปนี้ เริ่มแรกนั้นต้องกำหนดเกิร์ตที่ต้องการจะได้นิยามด้วย g_t และความยาวของคำรหัสที่ต้องการคือเท่ากับ N จากนั้นสร้างกราฟที่มีลักษณะเป็นแบบ finite-geometry ให้เป็นลักษณะของวงแหวน ซึ่งวงแหวนแรกเริ่มนี้ต้องมีจำนวนจุด (vertex) ที่เท่ากับ $g_t / 2$ เป็นอย่างน้อย ซึ่งจำนวนขั้น (layer) ที่จะได้คือ

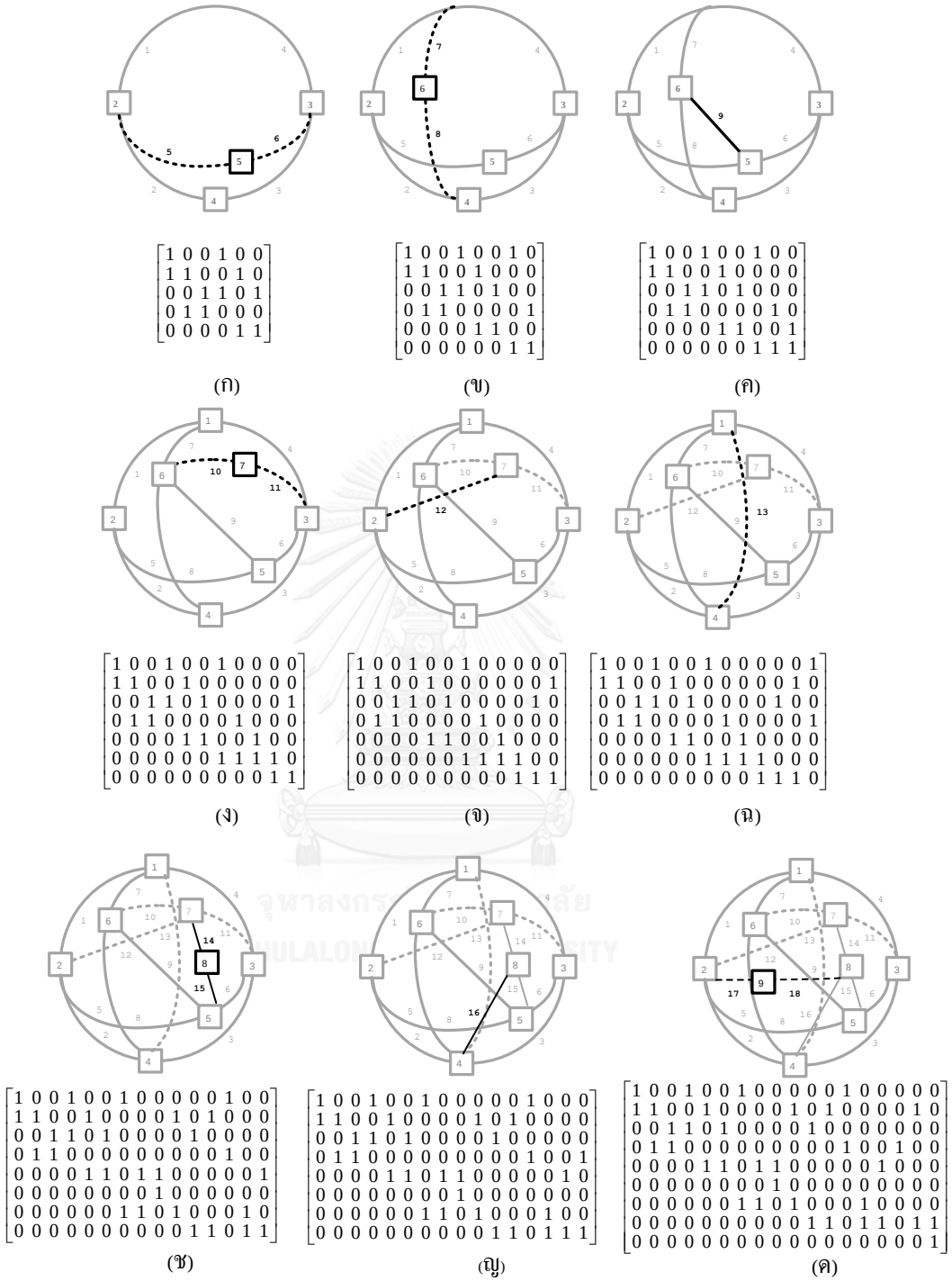
$$l_{\max} = \frac{|\mathrm{P}_M| + 1}{2} \quad \text{if } |\mathrm{P}_M| \text{ is odd} \tag{5.2}$$

และ

$$l_{\max} = \frac{|\mathrm{P}_M|}{2} + 1 \quad \text{if } |\mathrm{P}_M| \text{ is even} \tag{5.3}$$

โดยจะขอเรียกกราฟที่เป็นกราฟแบบ finite geometry นี้ว่ากราฟ G และกราฟเริ่มต้นนี้เองจะมีขนาดเป็น $|\mathrm{P}_M| \times |\mathrm{P}_M|$ จากนี้จะเห็นได้ว่าหากต้องการจะสร้างเมทริกซ์ $\mathbf{H}$ ที่มีจำนวนหลักเท่ากับ N จะต้องใส่จำนวน $M - |\mathrm{P}_M|$ เข้าไปในกราฟ G เริ่มต้นนี้ ซึ่งอัลกอริทึมในการใส่หลัก (เส้นในกราฟ G) เป็นจำนวน $M - |\mathrm{P}_M|$ โดยที่ยังรักษา g_t เอาไว้ได้จะถูกอธิบายนับจากนี้

1. เลือกจุดที่มีน้ำหนักน้อยที่สุดบน G (หากเป็น G เริ่มต้นนั้นคือเลือกจุดใดก็ได้) โดยจุดที่เลือกมานี้จะเป็นจุดที่อยู่ใน l_1
2. ขยายกราฟในรูปแบบกราฟของต้นไม้หาจุดที่อยู่ใน $l_{\max}$ ซึ่งจะเกิด 2 เงื่อนไขคือ
   - ถ้า $g_t \le 2l_{\max}$ ให้เริ่มใส่เส้นใหม่ไปใน G โดยที่เส้นที่พิ่มขึ้นมาใหม่นี้จะเป็นตัวที่เชื่อมกันระหว่างจุดใน l_1 (ซึ่งแน่นอนว่ามีจุดเดียว) ซึ่งเกิร์ตที่เกิดจากเส้นที่เกิดขึ้นใหม่ (เนื่องจากเส้นนั้นหมายถึงหลักใน $\mathbf{H}$ เมทริกซ์หรือโนดบิตนั้นเอง) จะมีค่าเท่ากับ $2l_{\max}$
   - ถ้า $g_t \ge 2l_{\max}$ ให้เพิ่มจุดใหม่เป็นจำนวน $(g_t / 2) - l_{\max}$ จุดและเพิ่มจำนวนเส้นใหม่เป็นจำนวน $((g_t / 2) - l_{\max}) + 1$ เส้น ซึ่งจุดและเส้นที่เพิ่มมาทั้งหมดจะเรียงต่อกันเป็นลักษณะแบบเส้นที่สลับกันไประหว่างเส้นกับจุดโดนที่ปลายข้างหนึ่งต่ออยู่กับจุดใน l_1
3. กลับไปเริ่มต้นที่ 1 จนกว่าจะสามารถมีเส้นบน G ครบ N เส้น
4. แปลงจากกราฟ G ให้เป็นกราฟแทนเนอร์ ซึ่งก็จะสามารถได้ $\mathbf{H}$ ที่มีทุกหลักเท่ากับ 2 จากกราฟแทนเนอร์นี้



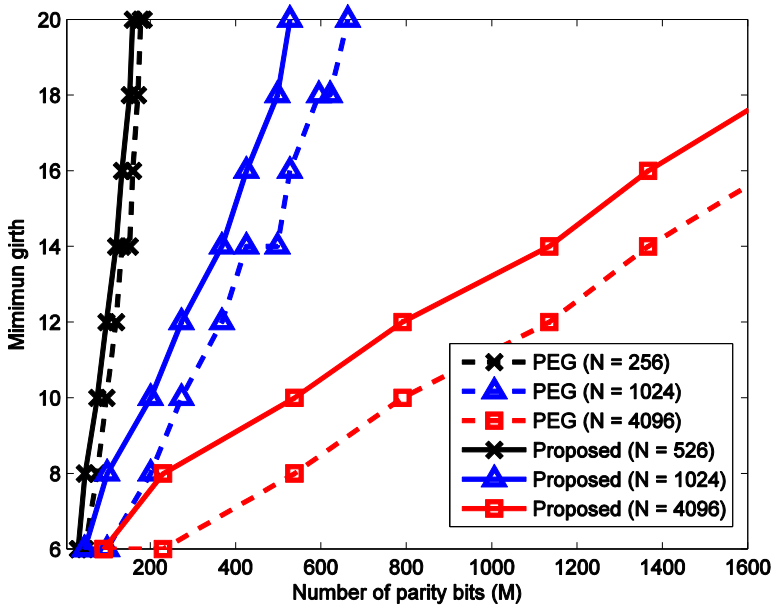
รูปที่ 5.4 ความสัมพันธ์ระหว่างเมทริกซ์พาริตี และวงแหวน G หลังจากดำเนินการไป 6 ขั้น

ตัวอย่าง : หากต้องการได้เมทริกซ์ $\mathbf{H}$ ที่มีเกิร์ตเท่ากับ 8 และจำนวนหลักเท่ากับ 18 หลัก จะต้องกำหนดตัวแปรดังนี้ $g_t = 8, N = 18$ ขั้นตอนแรกให้สร้างวงแหวนเริ่มแรก G สามารถสร้างได้ตาม

รูปที่ 5.3 (ข) หลังจากนั้นขยายกราฟแบบต้นไม้จากจุดที่ 2 จะได้ว่า จุดที่ 2 จะอยู่ที่ l_1 และจุดที่ 2 อยู่ใน $l_{\max}$ ซึ่ง $l_{\max} = l_3$ จะเห็นได้ว่าตรงกับเงื่อนไข $g_t \geq 2l_{\max}$ หมายความว่าต้องเพิ่ม 2 เส้นและ 1 จุดใน G คือเส้นที่ 5 และเส้นที่ 6 ละจุดที่ 5 ซึ่งจะแสดงได้ดังรูปที่ 5.4 (ก) ขั้นตอนที่ 2 เลือกจุดที่มีน้ำหนักน้อยที่สุดอย่างสุ่มสมมุติว่าเลือกจุดที่ 1 เพื่อขยายกราฟ หลังจากนั้นขยายกราฟแบบต้นไม้จากจุดที่ 1 นี้จะได้ว่า จุดที่ 1 จะอยู่ที่ l_1 และจุดที่ 4 อยู่ใน $l_{\max}$ ซึ่ง $l_{\max} = l_3$ ซึ่งตรงกับเงื่อนไข $g_t \geq 2l_{\max}$ หมายความว่าต้องเพิ่ม 2 เส้นและ 1 จุดใน G คือเส้นที่ 7 และเส้นที่ 8 ละจุดที่ 6 ซึ่งจะแสดงได้ดังรูปที่ 5.4 ข) ขั้นตอนที่ 3 เลือกจุดที่มีน้ำหนักน้อยที่สุดอย่างสุ่มในที่นี้เลือกจุดที่ 6 เพื่อขยายกราฟ หลังจากนั้นขยายกราฟแบบต้นไม้จากจุดที่ 6 นี้จะได้ว่า จุดที่ 6 จะอยู่ที่ l_1 และจุดที่ 5 อยู่ใน $l_{\max}$ ซึ่ง $l_{\max} = l_4$ ซึ่งตรงกับเงื่อนไข $g_t \leq 2l_{\max}$ กระบวนการนี้จะแตกต่างจาก 2 กระบวนการแรกคือเพิ่มเส้นที่ 9 มาเพียงเส้นเดียวและเชื่อมกันระหว่างจุดที่ 6 และจุดที่ 5 ซึ่งจะแสดงได้ดังรูปที่ 5.4 ค) ขณะนี้กราฟวงแหวน G มีเส้นอยู่จำนวน 9 เส้นเท่ากับเมทริกซ์ $\mathbf{H}$ ตอนนี้มี 9 หลักซึ่งยังขาดอีก 9 หลัก รูปที่ 5.4 (ง), รูปที่ 5.4 (จ) และรูปที่ 5.4 (ฉ) คือ 3 ขั้นตอนหลังจาก 3 ขั้นตอนแรกจนกราฟ G มีครบ 13 เส้นและ 7 จุด จะได้เมทริกซ์ $\mathbf{H}$ ที่มีแถว 7 แถวและหลัก 13 หลักและเกิร์ตมีขนาดอย่างต่ำเท่ากับ 8 ทุกหลัก ขั้นตอนที่เหลือดังจะเห็นได้ในรูปที่ 5.4 (ช), 5.4 (ญ) และ 5.4 (ด) ซึ่งเป็นการดำเนินการในแบบเดียวกัน จนกระทั่งได้หลักครับทั้ง 13 หลักตามต้องการ ซึ่งเมื่อจบกระบวนการทั้งหมดจะใช้โนดพาริตีเพื่อให้ได้เกิร์ตทั้งหมดมีค่าความยาวเท่ากับ 8 นั้นเท่ากับ 9 โนด

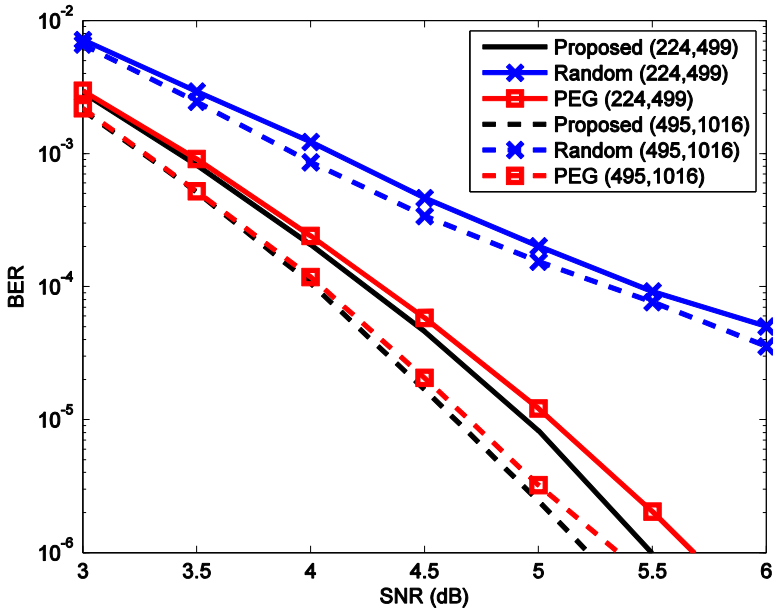
5.4 ผลการทดลองและการสรุปผล

ดังเช่นที่แล้วมาการแสดงผลจะอยู่ในรูปเทอมของ BER (Bit-Error Rate) โดยใช้ช่องสัญญาณเป็นสัญญาณรบกวนแบบเกาต์สีขาวแบบบวก ที่จำนวน k บิตข้อมูลแบบไบนารี $a_k = \{0,1\}$ เข้ารหัสแอลดีพีซีจะได้คำรหัสจำนวน N บิต โดยได้เป็นสัญญาณแบบ $b_k = \{-1,1\}$ และ ณ ภาครับได้รับสัญญาณเป็น $y_k = b_k + n_k$ โดย n_k คือสัญญาณรบกวนเป็นสัญญาณรบกวนแบบเกาต์สีขาวแบบบวกที่มีค่าเฉลี่ย (mean) เป็น 0 และมีความแปรปรวน (veriance) เท่ากับ σ^2 และที่ภาครับนี้เองถูกถอดรหัสด้วยรหัส แอลดีพีซีด้วยวิธีแบบซอฟต์ และการส่งข้อมูลเป็นจำนวน 50000 บล็อก และการถอดรหัสวนซ้ำจำนวน 30 รอบ วัดค่าไปจนกว่ามีจำนวนบิตผิดพลาด 1000 บิต



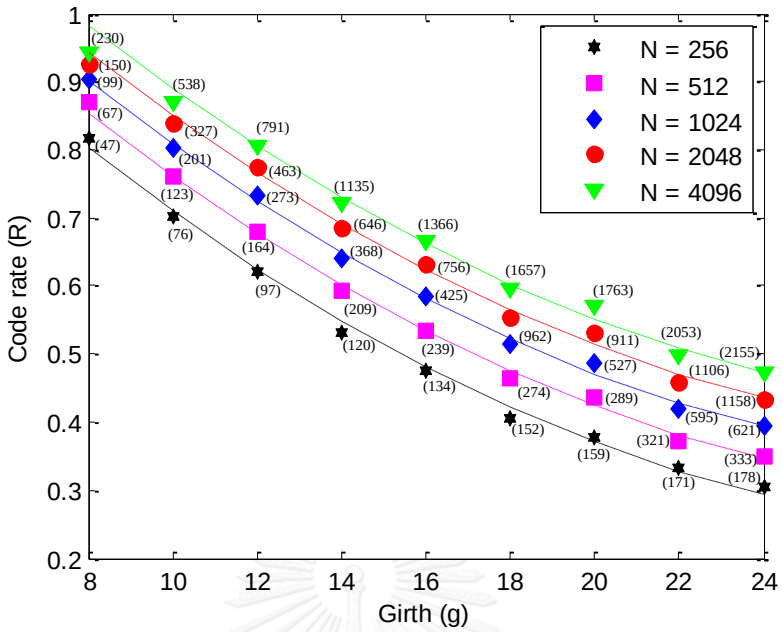
รูปที่ 5.5 ค่าของเกิร์ตที่น้อยที่สุดระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG ที่ขนาดของ code แตกต่างกัน

จากรูปที่ 5.5 แสดงค่าเกิร์ตที่น้อยที่สุดระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG ที่ขนาดของ code แต่ละขนาดคือ $N = 256, N = 1024$ และ $N = 2048$ โดยที่แกน x คือจำนวนของโนดพาริตีที่ใช้ แกน y คือค่าเกิร์ตที่น้อยที่สุด (minimum girth) จะเห็นได้ว่าทุกขนาดของคำรหัสหากมีขนาดของเกิร์ตที่น้อยที่สุดเท่ากันอัลกอริทึมที่นำเสนอนั้นใช้จำนวนโนดพาริตีที่น้อยกว่าอัลกอริทึม PEG หรืออีกนัยหนึ่งคือหากเปรียบเทียบที่ขนาดจำนวนของโนดพาริตีและโนดบิตที่เท่ากันอัลกอริทึมที่นำเสนอจะให้เกิร์ตที่สูงกว่าอัลกอริทึม PEG นั้นเอง



รูปที่ 5.6 BER ระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และแบบสุ่ม

จากรูปที่ 5.6 แสดงค่า BER ระหว่างเมทริกซ์ที่มาจากอัลกอริทึมที่นำเสนอและอัลกอริทึม PEG รวมไปถึงเมทริกซ์ที่ได้มาจากการสุ่มและถอดรหัสเป็นจำนวน 30 รอบ ซึ่งแสดงให้เห็นว่าเมทริกซ์ H ที่มาจากอัลกอริทึมที่นำเสนอนั้นสามารถลบความผิดพลาดได้ดีกว่าอัลกอริทึม PEG และแบบสุ่มทุกขนาด โดยขนาดของคำรหัสที่ใช้เป็น 2 ขนาดคือ 499 และ 1016 ที่อัตราโนดบิตต่อจำนวนโนดทั้งหมด (code rate) ที่ 0.5



รูปที่ 5.7 อัตราโนดบิตต่อจำนวนโนดทั้งหมดหากกำหนดเกิร์ตต่าง ๆ ของอัลกอริทึมที่นำเสนอ

คุณลักษณะอย่างหนึ่งของอัลกอริทึมที่นำเสนอคือสามารถที่จะกำหนด g_t และขนาดของ N ได้ แต่อย่างไรก็ตามจำนวนของโนดพาริตีที่ต้องใช้นั้นไม่สามารถที่จะกำหนดได้ขึ้นอยู่กับขนาดของ g_t และขนาดของ N ที่ต้องการ กล่าวคือยิ่งกำหนดให้ N มีขนาดใหญ่และต้องการให้ g_t มีความยาวมากก็ยิ่งต้องใช้จำนวนของโนดพาริตีเยอะมากด้วยเช่นกัน พิจารณารูปที่ 5.7 แกน y คืออัตราโนดบิตต่อจำนวนโนดทั้งหมดและแกน x คือจำนวน g_t ที่แตกต่างกัน เส้นแต่ละเส้นคือจำนวน N ที่แตกต่างกันเช่นกันคือ 256, 512, 1024, 2048 และ 4096 แสดงผลโดยเส้นทึบ โดยกราฟที่ได้เป็นการพลอตโดยใช้วิธี graph fitting เพื่อหาสมการเส้นตรงจากแกน x เพื่อให้ได้แกน y คืออัตราโนดบิตต่อจำนวนโนดทั้งหมด ซึ่งเมื่อสามารถที่จะรู้อัตราโนดบิตต่อจำนวนโนดทั้งหมดได้แล้วก็สามารถที่จะคาดคะเนจำนวนโนดพาริตีที่ใช้ไปได้หากต้องการได้เกิร์ตที่มีขนาดต่ำสุดที่ต้องการ โดยสมการ graph fitting ที่ได้ดังนี้

$$R = 0.00103g_t^2 - 0.0647g_t + 1.3532 + 0.8(\frac{x-10}{x+10}) \qquad (5.5)$$

โดยตัวแปร R นั้นคืออัตราโนดบิตต่อจำนวนโนดทั้งหมดนั้นเอง ซึ่งค่าประมาณที่ใช้จากสมการที่ 5.5 นี้จะแสดงอยู่ในรูปของเส้นประในรูปที่ 5.7 เมื่อได้ R มาเรียบร้อยแล้วนั่นจะนำมาหาค่าของโนดพาริตีที่จำเป็นต้องใช้ได้ดังนี้

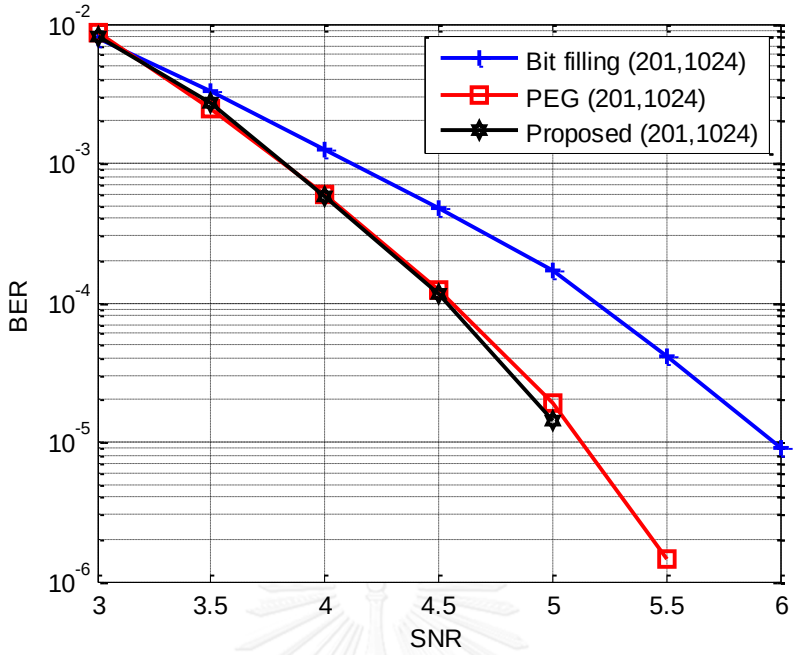
$$M = (1-R)\,\mathrm{N} \tag{5.6}$$

ตารางที่ 5-1 ตารางการเปรียบเทียบเกิร์ตของเมทริกซ์ของผู้เขียนวิทยานิพนธ์และเมทริกซ์ที่สร้างด้วยด้วยอัลกอริทึม PEG และ RDH

อัลกอริทึม	*M*	*N*	*R*	ค่าเกิร์ตที่น้อยที่สุด
อัลกอริทึมที่เสนอ	562	851	0.33	28
อัลกอริทึมที่เสนอ	2015	3182	0.33	32
อัลกอริทึมที่เสนอ	1148	6768	0.83	12
PEG	552	828	0.33	22
PEG	2128	3192	0.33	28
PEG	1128	6768	0.83	10
RDH	828	828	0.33	28
RDH	2128	3192	0.33	32
RDH	1128	6768	0.83	12

ตารางที่ 5-1 นี้แสดงการเปรียบเทียบเกิร์ตของเมทริกซ์ของที่ถูกสร้างมาจากอัลกอริทึมที่นำเสนอและเมทริกซ์ที่สร้างด้วยอัลกอริทึม PEG และ RDH ซึ่งอัลกอริทึม RDH หรือ recursive-design of high girth $(2,k)$ LDPC codes from (k,k) LDPC codes [25] ในปี 2011 ซึ่งแสดงให้เห็นว่าอัลกอริทึมที่นำเสนอให้ค่าเกิร์ตที่สูงกว่าอัลกอริทึม PEG เป็นอย่างมาก รวมทั้งให้ค่าเกิร์ตที่ใกล้เคียงกับอัลกอริทึม RDH หากแต่อัลกอริทึม RDH จะแบบเมทริกซ์ $\mathbf{H}$ แบบปกติในขณะที่อัลกอริทึมที่นำเสนอจะเป็นแบบไม่แบบปกติ (irregular LDPC codes) เนื่องด้วย Luby [26] ได้พิสูจน์ไว้ว่ารหัสแอลดีพีซีแบบไม่ปรกติสามารถให้สมรรถณะที่ดีกว่ารหัสแอลดีพีซีแบบปรกติได้

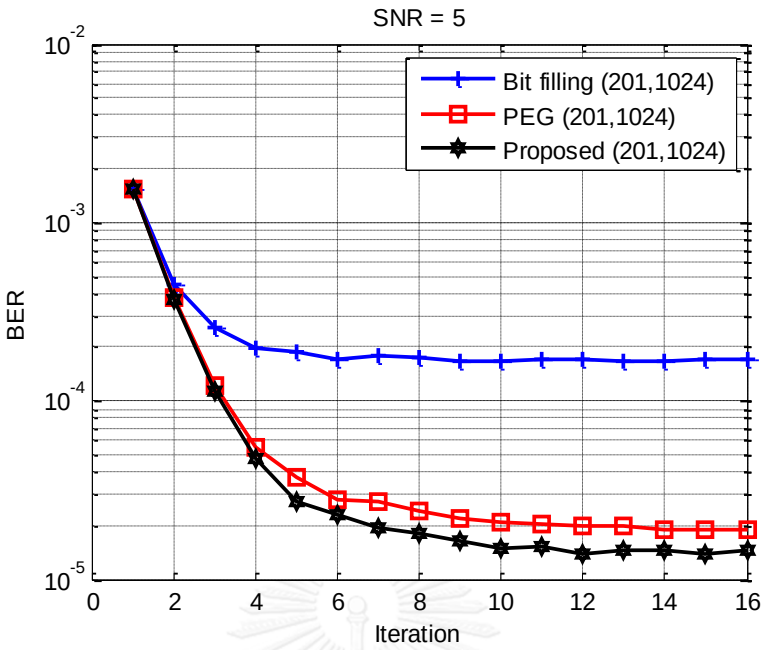
จากนี้เพื่อเป็นการวัดประสิทธิภาพของอัลกอริทึมที่เสนอนำเสนอให้ชัดเจนมากยิ่งขึ้นต่อไปนี้จึงจะเพิ่มเติมผลการทดลองและเพิ่มอัลกอริทึมอย่าง bit filling เพื่อนำมาใช้ในการเปรียบเทียบประสิทธิภาพกับเมทริกซ์ $\mathbf{H}$ ที่มาจากอัลกอริทึมที่นำเสนอซึ่งจะแสดงดังต่อไปนี้



รูปที่ 5.8 ค่า BER ระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน 0.8

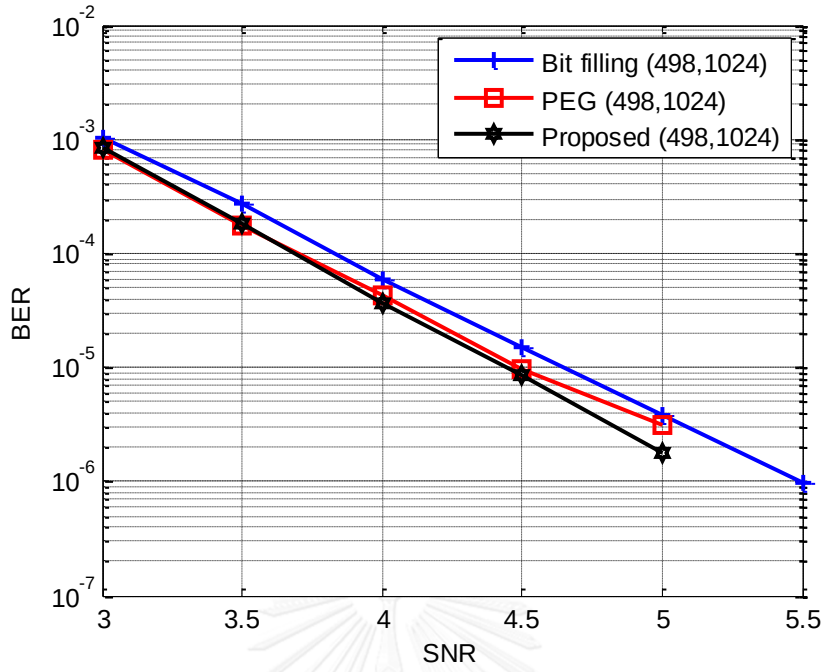
ตารางที่ 5-2 เปรียบเทียบเกิร์ตระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่ $R = 0.8$

อัลกอริทึม	เกิร์ตที่น้อยที่สุด	เกิร์ตที่มากที่สุด
อัลกอริทึมที่นำเสนอ	10	-
PEG	6	8
Bit filling	8	-



รูปที่ 5.9 ค่า BER จำนวน 16 วนซ้ำระหว่างระหว่างอัลกอริทึ่มที่นำเสนอกับอัลกอริทึ่ม PEG และ Bit filling ที่มีอัตราส่วน 0.8 ที่ SNR = 5

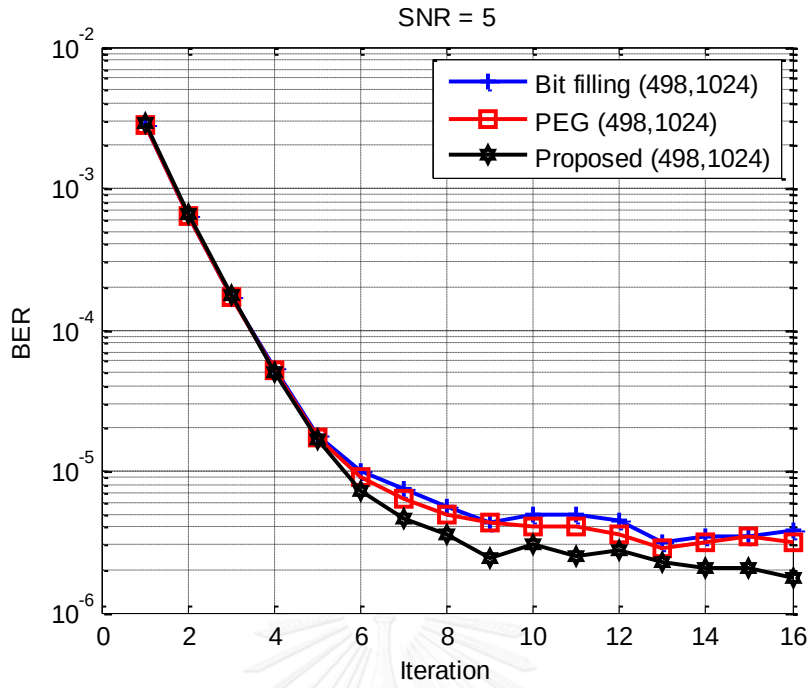
รูปที่ 5.8 แสดง ค่า BER ระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน $\boldsymbol{R}$ เท่ากับ 0.8 ที่รอบวนซ้ำสูงสุดที่ 16 รอบและขนาด 201 × 1024 จะสังเกตได้ว่าอัลกอริทึมที่นำเสนอเริ่มมีประสิทธิภาพที่ดีขึ้นเมื่อเทียบกับอัลกอริทึมอื่น ๆ เมื่อผ่าน SNR = 4 ไปแล้ว ซึ่งค่าต่ำสุดของเกิร์ตที่ได้ของทั้ง 3 มีดังนี้ อัลกอริทึมที่นำเสนอทุกหลักมีเกิร์ตเท่ากับคือ 10 ส่วนอัลกอริทึม PEG นั้นเกิร์ตจะมีการกระจายตัวอยู่ที่ 6 และ 8 ส่วนของอัลกอริทึม Bit filling สามารถมีเกิร์ตได้เพียงเท่ากับ 8 เท่านั้นที่มิติขนาดเดียวกัน ส่วนรูปที่ 5.9 แสดง BER จำนวน 16 วนซ้ำระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน 0.8 ที่ SNR = 5 จะเห็นได้ชัดว่าอัลกอริทึมที่นำเสนอสามารถแก้ความผิดพลาดได้รวดเร็วกว่าทั้งอัลกอริทึม PEG และ Bit-filling นอกจากนี้ตั้งแต่การรอบวนซ้ำรอบที่ 3 อัลกอริทึมที่เสนอก็เริ่มที่จะมีประสิทธิภาพดีกว่าอัลกอริทึมอื่น ๆ



รูปที่ 5.10 ค่า BER ระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน 0.51

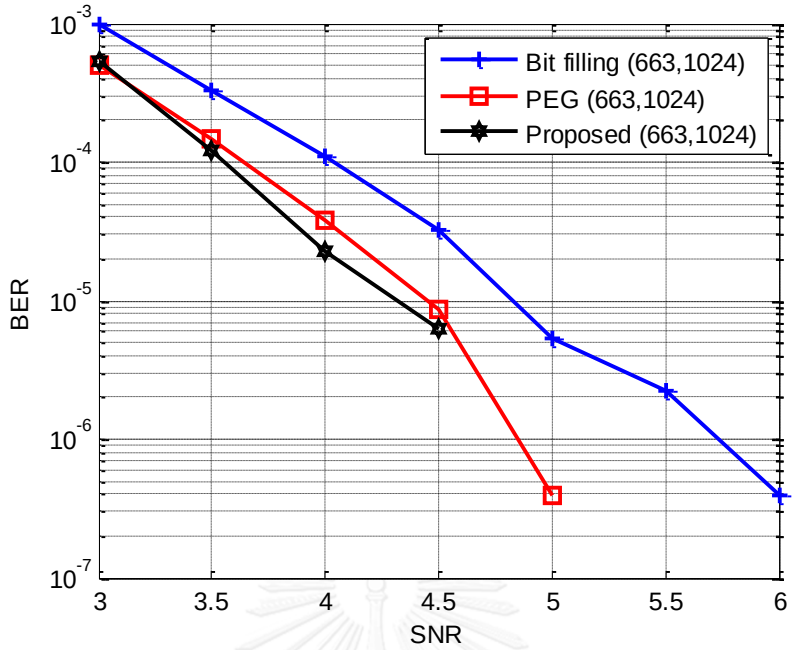
ตารางที่ 5-3 เปรียบเทียบเกิร์ตระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่ $R = 0.51$

อัลกอริทึม	เกิร์ตที่น้อยที่สุด	เกิร์ตที่มากที่สุด
อัลกอริทึมที่นำเสนอ	18	-
PEG	14	18
Bit filling	16	-



รูปที่ 5.11 ค่า BER จำนวน 16 วนซ้ำระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน 0.8 ที่ SNR = 5

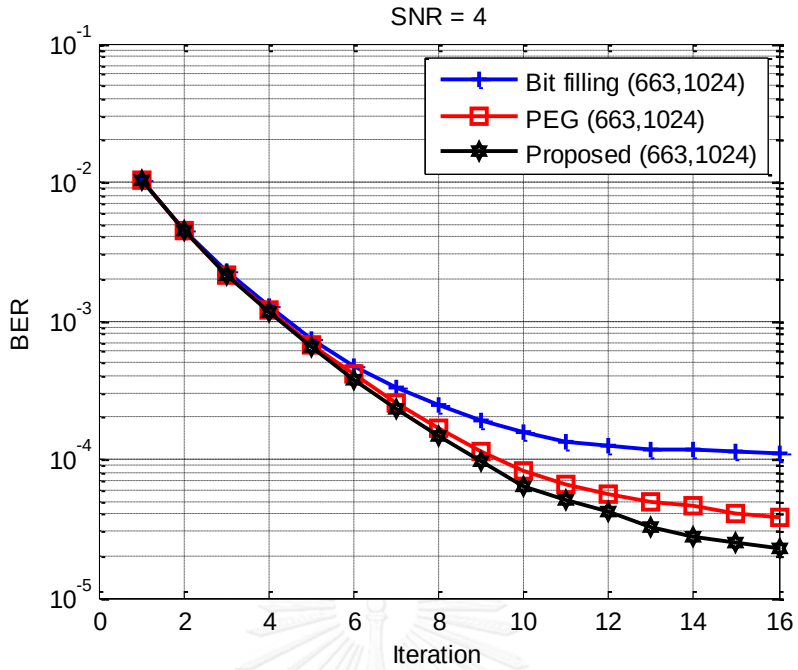
รูปที่ 5.10 แสดง ค่า BER ระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน R เท่ากับ 0.51 ที่รอบวนซ้ำสูงสุดที่ 16 รอบและขนาด 498×1024 จะสังเกตได้ว่าอัลกอริทึมที่นำเสนอเริ่มมีประสิทธิภาพที่ดีขึ้นเมื่อเทียบกับอัลกอริทึมอื่น ๆ เมื่อผ่าน SNR = 4 ไปแล้ว ซึ่งค่าต่ำสุดของเกิร์ตที่ได้ของทั้ง 3 มีดังนี้ อัลกอริทึมที่นำเสนอทุก ๆ หลักจะมีเกิร์ตเท่ากันคือ 18 ส่วน อัลกอริทึม PEG เกิร์ตจะมีการกระจายตัวอยู่ที่ 14, 16 และ 18 ส่วนของอัลกอริทึม Bit filling สามารถมีเกิร์ตได้เพียงขนาดเท่ากับ 16 ส่วนรูปที่ 5.11 แสดง BER จำนวน 16 วนซ้ำระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน 0.8 ที่ SNR = 5 จะเห็นได้ชัดว่าอัลกอริทึมที่นำเสนอสามารถแก้ความผิดพลาดได้รวดเร็วกว่าทั้งอัลกอริทึม PEG และ Bit filling นอกจากนี้ตั้งแต่รอบการวนซ้ำรอบที่ 6 อัลกอริทึมที่เสนอก็เริ่มที่จะมีประสิทธิภาพดีกว่าอัลกอริทึมอื่น ๆ



รูปที่ 5.12 ค่า BER ระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน 0.35

ตารางที่ 5-4 เปรียบเทียบเกิร์ตระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่ $R = 0.35$

อัลกอริทึม	เกิร์ตที่น้อยที่สุด	เกิร์ตที่มากที่สุด
อัลกอริทึมที่นำเสนอ	26	-
PEG	20	26
Bit filling	20	-



รูปที่ 5.13 ค่า BER จำนวน 16 วนซ้ำระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน 0.8 ที่ SNR = 4

รูปที่ 5.12 แสดง ค่า BER ระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน $\boldsymbol{R}$ เท่ากับ 0.35 ที่รอบวนซ้ำสูงสุดที่ 16 รอบและขนาด 663 × 1024 จะสังเกตได้ว่าอัลกอริทึมที่นำเสนอเริ่มมีประสิทธิภาพที่ดีขึ้นเมื่อเทียบกับอัลกอริทึมอื่น ๆ ที่มิติ 663 × 1024 นี้ จากรูปอัลกอริทึมที่นำเสนอมีประสิทธิภาพดีที่สุดตั้งแต่ SNR เท่ากับ 3 เลยทีเดียว ซึ่งค่าต่ำสุดของเกิร์ตที่ได้ของทั้ง 3 จะมีดังนี้ อัลกอริทึมที่นำเสนอทุกหลักมีเกิร์ตเท่ากับคือ 26 ส่วนอัลกอริทึม PEG เกิร์ตจะมีการกระจายตัวอยู่ที่ 20, 22, 24 และ 26 ส่วนของอัลกอริทึม Bit filling มีเกิร์ตเพียงขนาดเดียวคือเท่ากับ 20 ส่วนรูปที่ 5.13 แสดง BER จำนวน 16 วนซ้ำระหว่างระหว่างอัลกอริทึมที่นำเสนอกับอัลกอริทึม PEG และ Bit filling ที่มีอัตราส่วน 0.8 ที่ SNR = 5 จากรูปจะเห็นได้ว่าอัลกอริทึมที่นำเสนอนั้นสามารถแก้ความผิดพลาดได้รวดเร็วกว่าทั้งอัลกอริทึม PEG และ Bit filling และแต่รอบการวนซ้ำรอบที่ 8 อัลกอริทึมที่เสนอก็เริ่มที่จะมีประสิทธิภาพดีกว่าอัลกอริทึมอื่น ๆ อย่างเห็นได้ชัด

บทที่ 6
บทสรุปและข้อเสนอแนะ

วิทยานิพนธ์ฉบับนี้ได้มุ่งเน้นในการนำเสนอวิธีการสร้างเมทริกซ์พาริตีหรือเมทริกซ์ $\mathbf{H}$ บนรหัสแอลดีพีซีโดยมีเกิร์ตเป็นพารามิเตอร์หลักในการออกแบบวิธีการสร้าง อันเนื่องมาจากแทนเนอร์ได้เสนอในบทความวิจัยของตนเองว่าเกิร์ตนั้นมีความสัมพันธ์กับระยะแฮมมิงที่สั้นที่สุด กล่าวคือ เมทริกซ์ $\mathbf{H}$ ใดมีเกิร์ตที่มีขนาดยาวแล้วจะมีคำรหัสที่มีระยะแฮมมิงที่สั้นที่สุดที่ยาวตามไปด้วย ทั้งนี้จากการถอดรหัสแบบแอลดีพีซีนั้นเป็นไปในเชิงการแลกเปลี่ยนข้อมูลข่าวสารภายในคำรหัสซึ่งหากเมทริกซ์ $\mathbf{H}$ ใดมีมีเกิร์ตขนาดยาวการแลกเปลี่ยนข้อมูลนั้นก็จะเป็นไปในแบบวงกว้างทำให้การตัดสินใจของบิตข้อมูลจึงมีประสิทธิภาพมากยิ่งขึ้นด้วย อย่างไรก็ตามพารามิเตอร์เกิรต์นี้ยังเป็นที่ยอมรับในบรรดานักวิจัยทั่วไปว่าสามารถช่วยเพิ่มประสิทธิภาพให้กับรหัสแอลดีพีซีได้

บทที่ 4 ของวิทยานิพนธ์ฉบับนี้เป็นการออกแบบเมทริกซ์ $\mathbf{H}$ ของรหัสแอลดีพีซีชนิด Quasi-Cyclic หรือ Quasi-Cyclic LDPC codes เพื่อให้มีเกิร์ตที่สูงที่สุดโดยใช้อัลกอริทึมการค้นหา (search algorithm) เพื่อหาเลขที่ดีที่สุดเพื่อมาใช้ในการหมุนเมทริกซ์ย่อย ซึ่งเมทริกซ์ $\mathbf{H}$ แบบ Quasi-Cyclic นี้เป็นที่นิยมในการนำมาใช้กับระบบฮาร์ดแวร์และมีมาตรฐานของการสื่อสารมากมายที่ได้นำเมทริกซ์ H ชนิดนี้มาใช้ จากผลการทดลองจะเห็นได้ว่าเกิร์ตที่ได้จากวิธีการสร้างเมทริกซ์ $\mathbf{H}$ ที่นำเสนอนั้นให้เกิร์ตที่สูงเมื่อเทียบกับวิธีการสร้างแบบ SFT, Magic square และ MAC ที่สำคัญยังให้ประสิทธิภาพที่ดีในเทอมของ BER เมื่อนำมาเปรียบเทียบกับวิธีการสร้างแบบต่าง ๆ ดังที่ได้กล่าวไปแล้ว การทดลองนั้นยังพบอีกด้วยว่าวิธีการสร้างเมทริกซ์ $\mathbf{H}$ ที่นำเสนอนั้นยังใช้จำนวนรอบที่น้อยกว่าเพื่อแก้ไขความผิดพลาดของบิตข้อมูล

บทที่ 5 นั้นเป็นการออกแบบเมทริกซ์ $\mathbf{H}$ ที่มีลักษณะที่ทุกหลักจะมีน้ำหนักที่เท่ากับ 2 ซึ่งข้อดีของเมทริกซ์ $\mathbf{H}$ ชนิดนี้คือให้ประสิทธิภาพที่ดีเมื่อนำมาใช้ในช่องสัญญาณแบบวนกลับรวมไปถึงมีประสิทธิภาพที่ดีเมื่อเป็นเมทริกซ์ $\mathbf{H}$ แบบนอนไบนารี โดยวิธีการที่นำเสนอเป็นการเริ่มสร้างเมทริกซ์ $\mathbf{H}$ จากในมุมมองแบบกราฟแทนเนอร์ที่มีขนาดเล็ก ๆ ลักษณะเป็นรูปทรงวงแหวนและขนาดของเกิร์ตตามที่กำหนดจนเมื่อมีการเพิ่มเติมขนาดของกราฟแทนเนอร์นั้นเกิร์ตที่เกิดขึ้นใหม่จะมีขนาดไม่ลดลงจากที่กำหนด วิธีการแบบนี้สามารถที่จะสามารถกำหนดเกิร์ตที่ต่ำที่สุดที่ในกราฟของแทนเนอร์ที่สร้างขึ้นได้ แต่หากต้องการเกิร์ตที่สูงแล้วนั้นต้องแลกมาด้วยการใช้โนดพาริตีที่สูงขึ้นตามไปด้วย อย่างไรก็ตามเมื่อเปรียบเทียบเกิร์ตที่ขนาดมิติเดียวกันกับอัลกอริทึม PEG และ Bit filling

แล้ว วิธีการสร้างที่นำเสนอสามารถให้เกิร์ตที่สูงกว่า ในทิศทางเดียวกันนี้การเปรียบเทียบประสิทธิในเทอมของ BER ก็ยังแสดงให้เห็นว่าเมทริกซ์ $\mathbf{H}$ ที่ถูกสร้างมาจากวิธีการสร้างที่นำเสนอนั้นให้ประสิทธิภาพที่สูงกว่าแบบ PEG และ Bit filling ซึ่งเป็นอัลกอริทึมที่มีชื่อเสียงได้

ทั้งนี้นอกจากเกิร์ตที่เป็นพารามิเตอร์ในการออกแบบเมทริกซ์ $\mathbf{H}$ บนรหัสแอลดีพีซีแล้วยังมีพารามิเตอร์ที่น่าสนใจสำหรับการออกแบบรหัสแอลดีพีซีคือการ ”แจกแจงของน้ำหนัก (weight-distribution) ” เนื่องจากผลการทดลองอื่น ๆ แสดงให้เห็นว่า แม้จะมีเกิร์ตในเมทริกซ์ $\mathbf{H}$ ที่มีขนาดไม่มากแต่หากมีน้ำหนักของแต่ละหลักที่สูงแล้วประสิทธิภาพของรหัสแอลดีพีซีนั้นจะยังคงดีอยู่แต่ต้องแลกมาด้วยการถอดรหัสที่มีความซับซ้อนมากยิ่งขึ้นและใช้รอบในการวนซ้ำมากขึ้นด้วย ที่เป็นเช่นนี้อันเนื่องมาจากในแต่ละหลักของเมทริกซ์ $\mathbf{H}$ จะมีหลายเส้นทางที่เริ่มต้นจากหลักนั้นเดิมและกลับมาที่หลัก ๆ นั้น ส่วนเกิร์ตก็เป็นเพียงแต่เส้นทางที่สั้นที่สุด หากมีวนซ้ำหลาย ๆ รอบแล้ว การแลกเปลี่ยนข้อมูลข่าวสารก็จะเป็นไปในแบบวงกว้างมากยิ่งขึ้น ซึ่งจะเป็นการเพิ่มโอกาสในการที่จะแก้ไขข้อผิดพลาดของบิตข้อมูลได้นั้นเอง ข้อมูลในส่วนนี้อาจเป็นผลดีสำหรับการวิจัยต่อ ๆ ไปในเรื่องของการคิดค้นอัลกอริทึมเพื่อสร้างเมทริกซ์ $\mathbf{H}$ บนรหัสแอลดีพีซีโดยใช้แจกแจงของน้ำหนักเป็นพารามิเตอร์ในการออกแบบต่อไป

ภาคผนวก

รายการอ้างอิง

1. R. Gallager, *Low-density parity-check codes.* IRE Transactions on Information Theory, 1962. **8**(1): p. 21–28.
2. D. J. C. MacKay and R. M. Neal, *Near Shannon limit performance of low density parity check codes.* Electron. Lett, 1996. **32**( ): p. 1645–1646.
3. C. E. Shannon, *A mathematical theory of communication.* The Bell System Technical Journal, 1948. **27**: p. 21-28.
4. R. M. Tanner, *A recursive approach to low complexity codes.* Information Theory, IEEE Transactions on 27.5 (1981): 533-547.
5. D. J. C. MacKay, *Good error-correcting codes based on very sparse matrices.* Information Theory, IEEE Transactions on, 1999. **45**(2): p. 399-431.
6. S. J. Jahnson, *Iterative error correction Turbo, Low-Density parity-Check and repeat-Accumulate code*. 2010, United States of America Cambirdge Press.
7. X. Y. Hu, E.E. and D. M. Arnold, *Regular and irregular progressive edge-growth tanner graphs.* IEEE Trans. Inform. Theory, 2005. **51**: p. 386–398.
8. A. Venkiah, D. Declercq, and C. Poulliat, *Design of cages with a randomized progressive edge-growth algorithm.* Communications Letters, IEEE, 2008. **12**(4): p. 301-303.
9. A. Venkiah, D. Declercq, and C. Poulliat. *Randomized Progressive Edge-Growth (RandPEG)*. in *Turbo Codes and Related Topics, 2008 5th International Symposium on*. 2008.
10. J. Xueqin, X. Xiang-Gen, and L. Moon Ho, *Efficient Progressive Edge-Growth Algorithm Based on Chinese Remainder Theorem.* Communications, IEEE Transactions on, 2014. **62**(2): p. 442-451.
11. G. Srirutchataboon, et al. *PEG-like algorithm for LDPC codes*. in *Electrical Engineering Congress (iEECON), 2014 International*. 2014.
12. J. Campello, D. S. Modha, and S. Rajagopalan. *Designing LDPC codes using bit-filling*. in *Communications, 2001. ICC 2001. IEEE International Conference on*. 2001.

13. Y. Chen and K. K. Parhi, *Overlapped message passing for quasi-cyclic low-density parity check codes.* Circuits and Systems I: Regular Papers, IEEE Transactions on, 2004. **51**(6): p. 1106-1113.
14. J. L. Fan, *Array Low-density Parity-check Codes.* 2nd Int. Symp. Turbo Codes, 2000: p. 543-546.
15. โควินท์ทวีวัฒน์, ป., การประมวลผลสัญญาณสำหรับการจัดเก็บข้อมูลดิจิทัล .Vol. 1. 2554, นครปฐม.
16. T. J. Richardson, M.A. Shokrollahi, and R .L. Urbanke, *Design of capacity-approaching irregular low-density parity-check codes.* Information Theory, IEEE Transactions on, 2001. **47**(2): p. 619-637.
17. E. Eleftheriou, and S. Olcer. *Low-density parity-check codes for digital subscriber lines*. in *Communications, 2002. ICC 2002. IEEE International Conference on*. 2002.
18. W. Singhaudom, S. Noppankeepong, and P. Supnithi, *Design of high-rate modified array codes for magnetic recording system*, in *ECTI-CON*. May 2007.
19. S. Hongwei, L Jingfeng, and B.V.K.V. Kumar, *Low complexity LDPC codes for partial response channels*. in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*. 2002.
20. S. Hongwei, L Jingfeng, and B.V.K.V. Kumar, *Large girth cycle codes for partial response channels.* Magnetics, IEEE Transactions on, 2004. **40**(4): p. 3084-3086.
21. C. Poulliat, M. Fossorier, and D. Declercq, *Design of regular (2,d/sub c/)-LDPC codes over GF(q) using their binary images.* Communications, IEEE Transactions on, 2008. **56**(10): p. 1626-1635.
22. G., M. and M. Liebelt, *High girth column-weight-two LDPC codes based on distance graphs.* EURASIP Journal on Wireless Communications and Networking, 2007.
23. G. Royle, *Cages of higher valency*.
24. P. Wong, *Cage-a survey.* Journal of Graph Theory, 1982. **6**: p. 1-22.
25. T. Xiongfei, et al., *Recursive Design of High Girth (2,k) LDPC Codes from (k,k) LDPC Codes.* Communications Letters, IEEE, 2011: p. 70-72.

26. M. Luby, et al., *Improved Low-Density Parity-Check Codes Using Irregular Graphs.* IEEE Trans. Inform. Theory, 2001. **47**: p. 585-598.

ภาคผนวก

ประวัติผู้เขียนวิทยานิพนธ์

นาย กานต์ ศรีรัชตบูรณ์ เกิดเมื่อวันที่ 27 มิถุนายน พ.ศ. 2529 เข้ารับการศึกษาในหลักสูตรวิศวกรรมศาสตร์บัณฑิต สาขาคอมพิวเตอร์ในปีการศึกษา 2548 และเข้าศึกษาต่อ ในหลักสูตรวิศวกรรมศาสตร์มหาบัณฑิต สาขาวิศวกรรมไฟฟ้านปีการศึกษา 2554