การปรับปรุงการตรวจจับข้อผิดพลาดด้วยการเข้ารหัสเชิงเลขคณิต

นายสมภพ โชคชัยธรรม

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2547

ISBN 974-17-6117-1

IMPROVEMENT OF ERROR DETECTION USING ARITHMETIC CODING

Mr. Somphop Chokchaitam

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2004
ISBN 974-17-6117-1

หัวข้อวิทยานิพนธ์	การปรับปรุงการตรวจจับข้อผิดพลาดด้วยการเข้ารหัสเชิงเลขคณิต
โดย	นาย สมภพ โชคชัยธรรม
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษา	รองศาสตราจารย์ ดร. ประสิทธิ์ ทีฆพุฒิ

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

…….. คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร. ดิเรก ลาวัณย์ศิริ)

คณะกรรมการสอบวิทยานิพนธ์

…… ประธานกรรมการ
(ศาสตราจารย์ ดร. ประสิทธิ์ ประพิณมงคลการ)

……... อาจารย์ที่ปรึกษา
(รองศาสตราจารย์ ดร. ประสิทธิ์ ทีฆพุฒิ)

……กรรมการ
(รองศาสตราจารย์ ดร. ลัญฉกร วุฒิสิทธิกุลกิจ)

……กรรมการ
(ดร. ศิวรักษ์ ศิวโมกษธรรม)

สมภพ โชคชัยธรรม : การปรับปรุงการตรวจจับข้อผิดพลาดด้วยการเข้ารหัสเชิงเลขคณิต. (IMPROVEMENT OF ERROR DETECTION USING ARITHMETIC CODING) อ. ที่ปรึกษา : รศ. ดร.ประสิทธิ์ ทีฆพุฒิ, 80 หน้า. ISBN 974-17-6117-1.

วิทยานิพนธ์นี้นำเสนอวิธีการตรวจจับข้อผิดพลาดซึ่งมีการใช้คุณสมบัติการแพร่กระจายของข้อผิดพลาดของการเข้ารหัสเชิงเลขคณิตโดยอาศัยพื้นฐานความจริงที่ว่าถ้าหากเกิดข้อผิดพลาดเกิดขึ้นกับข้อมูลที่ผ่านการเข้ารหัสด้วยการเข้ารหัสเชิงเลขคณิต ข้อผิดพลาดนั้นจะแพร่กระจายไปทั่วข้อมูลทั้งหมดที่ถอดรหัสจากข้อมูลที่เสียหาย คุณสมบัติอันนี้เองทำให้สามารถใช้วิธีการตรวจจับข้อผิดพลาดโดยการใส่สัญลักษณ์ที่รู้ว่าจะต้องเกิดขึ้นหรือที่เรียกว่าเครื่องหมายเข้าไปในข้อมูล ที่ตัวถอดรหัสหากสัญลักษณ์ที่ใส่เข้าไปนี้ไม่ปรากฏอยู่ที่ตำแหน่งที่ถูกต้องแสดงว่ามีข้อผิดพลาดเกิดขึ้น วิธีการที่นำเสนอเลือกสัญลักษณ์ที่มีความถี่ในการเกิดสูงที่สุดมาใช้เป็นเครื่องหมายซึ่งหมายความว่าตอนนี้การตัดสินใจเลือกเอาสัญลักษณ์ใดมาเป็นเครื่องหมายจะถูกตัดสินโดยดูจากความน่าจะเป็นในการเกิดของสัญลักษณ์ ซึ่งเป็นเหตุให้การตรวจจับข้อผิดพลาดที่นำเสนอสามารถแลกเปลี่ยนความซ้ำซ้อนเพียงเล็กน้อยกับความสามารถในการตรวจจับข้อผิดพลาดได้ นอกจากนี้วิธีการที่นำเสนอยังใช้ความซับซ้อนในการคำนวณน้อยกว่าวิธีการตรวจจับข้อผิดพลาดต่อเนื่อง

ภาควิชา วิศวกรรมไฟฟ้า ลายมือชื่อนิสิต .....
สาขาวิชา วิศวกรรมไฟฟ้า ลายมือชื่ออาจารย์ที่ปรึกษา .....
ปีการศึกษา 2547 ลายมือชื่ออาจารย์ที่ปรึกษาร่วม .....

4570746621 : MAJOR ELECTRICAL ENGINEERING

KEY WORD: ARITHMETIC CODING / ERROR DETECTION / ERROR PROPAGATION

SOMPHOP CHOKCHAITAM : IMPROVEMENT OF ERROR DETECTION USING ARITHMETIC CODING. THESIS ADVISOR : ASSOC. PROF. PRASIT TEEKAPUT, Ph.D., 80 pp. ISBN 974-17-6117-1.

This thesis proposes an error detection method which utilizes error propagation property of arithmetic coding. This method based on the fact that if an error occurred in the data encoded by arithmetic coding. The error will propagate through the end of the decoded data. This property provides a method to detect errors by inserting known symbol called marker into the data. At the decoder, if the known symbol is not present at its location. Then an error has been occurred. The proposed method employs the most frequent symbol as marker which means the marker is now determined by its probability of occurrence. Therefore the proposed error detection can trade-off small amount of redundancy for error detection capability. The proposed method also consumes less computational complexity than continuous error detection scheme.

Department Electrical Engineering Student's signature .....

Field of study Electrical Engineering Advisor's signature .....

Academic year 2004 Co-advisor's signature .....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยคำแนะนำและความช่วยเหลืออย่างดียิ่งของอาจารย์ที่ปรึกษาวิทยานิพนธ์คือ รศ.ดร. ประสิทธิ์ ทีฆพุฒิ คำปรึกษาที่ได้นั้นเป็นประโยชน์ทั้งต่องานวิจัย และในการดำรงชีวิตซึ่งมีส่วนให้วิทยานิพนธ์นี้ลุล่วงไปด้วยดี

ขอขอบคุณโครงการเสริมสร้างความเชื่อมโยงระหว่างภาควิชาวิศวกรรมไฟฟ้าและภาคเอกชนทางด้านการวิจัยและพัฒนา (Research and Development Cooperation Project between Department of Electrical Engineering and Private Sector) ที่ช่วยสนับสนุนเงินทุนสำหรับการทำงานวิจัยนี้

ขอขอบคุณ อาจารย์ภาควิชาไฟฟ้า จุฬาลงกรณ์มหาวิทยาลัยทุกท่านที่ช่วยให้คำแนะนำแนวทางการดำเนินงานและให้ข้อคิดต่างๆ กับงานวิจัย

นอกจากนี้ขอขอบคุณ เพื่อนๆ พี่ๆ น้องๆ ทุกคนที่คอยให้การช่วยเหลือ ให้กำลังใจ ช่วยผ่อนคลายความเครียดจากการทำงานวิจัย ให้คำแนะนำ และเป็นแรงกระตุ้นให้วิทยานิพนธ์นี้ลุล่วงไปด้วยดี

สุดท้ายผู้วิจัยขอขอบคุณบิดามารดาและครอบครัวที่คอยให้กำลังใจ และให้การสนับสนุนเป็นอย่างดีแก่ผู้วิจัยเสมอมา

สารบัญ

สารบัญตาราง

สารบัญภาพ

ภาพประกอบ หน้า

ภาพประกอบ หน้า

บัญชีคำศัพท์

Arithmetic Coding	การเข้ารหัสเชิงเลขคณิต
Adaptive model	แบบจำลองแบบปรับตัวได้
Algorithm	ขั้นตอนวิธี
Bandwidth	แบนด์วิดท์
Binary Decoder Tree	ต้นไม้ถอดรหัสฐานสอง
Bit Error Rate	อัตราข้อผิดพลาดบิต
Channel Coding	การเข้ารหัสช่องสัญญาณ
Checksum	ผลรวมตรวจสอบ
Codeword	คำรหัส
Complexity	ความซับซ้อน
Compression Ratio	อัตราส่วนการบีบอัด
Computational	การคำนวณ
Continuous Error Detection	การตรวจจับข้อผิดพลาดต่อเนื่อง
CRC (Cyclic Redundancy Check)	ซีอาร์ซี(การตรวจสอบด้วยส่วนซ้ำซ้อนแบบวน)
Decoder	ตัวถอดรหัส
Decrementing Semi-Adaptive Model	แบบจำลองแบบกึ่งปรับตัวได้แบบหักออก
Delay	ประวิง
Digital	ดิจิทัล
Encoder	ตัวเข้ารหัส
End of File Symbol	สัญลักษณ์สิ้นสุดแฟ้ม
Entropy	เอนโทรปี
Error	ข้อผิดพลาด
Error Correction	การแก้ข้อผิดพลาด
Error Detection	การตรวจจับข้อผิดพลาด
Error Propagation	การแพร่กระจายของข้อผิดพลาด
File	แฟ้ม
Forbidden Symbol	สัญลักษณ์ต้องห้าม
Frequency	ความถี่
Gaussian	เกาส์เซียน
Generator Polynomial	พหุนามตัวก่อกำเนิด

Huffman Coding	การเข้ารหัสฮัฟฟ์แมน
Implementation	การนำไปปฏิบัติ
Logarithm	ลอการิทึม
Marker	เครื่องหมาย
Markov Chain	มาร์คอฟเชน
Memory	หน่วยความจำ
Model	แบบจำลอง
Node	จุดต่อ
Optimal	เหมาะที่สุด
Parameter	พารามิเตอร์
Parity Bit	บิตภาวะคู่หรือคี่
Pattern	รูปแบบ
Prefix	เติมหน้า
Probability	ความน่าจะเป็น
Process	ประมวล(ผล)
Random	สุ่ม
Redundancy	ความซ้ำซ้อน
Source Coding	การเข้ารหัสต้นทาง
Static Model	แบบจำลองแบบตายตัว
Static Semi-Adaptive Model	แบบจำลองแบบกึ่งปรับตัวได้แบบตายตัว
Stream	กระแส(ข้อมูล)
Synchronization	การประสานเวลา
Transform	แปลง
Variable Length Code	รหัสความยาวแปรได้

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันการสื่อสารมีความก้าวหน้าและมีความสำคัญอย่างมาก เนื่องจากการสื่อสารเป็นหนึ่งในส่วนสำคัญของการดำเนินงานในหลายๆ ด้าน การสื่อสารที่ใช้กันมากในปัจจุบันจะเป็นการสื่อสารระบบดิจิทัลซึ่งเป็นการสื่อสารที่มีประสิทธิภาพสูงเนื่องจากสามารถที่จะทำการเข้ารหัสข้อมูลเพื่อเพิ่มประสิทธิภาพได้ ตัวอย่างเช่น การที่จะส่งข้อมูลที่มีปริมาณมากๆในช่องสัญญาณที่มีแบนด์วิดท์จำกัดจะต้องใช้เวลามาก แต่ในการสื่อสารระบบดิจิทัลนั้นสามารถที่จะทำการเข้ารหัสต้นทาง (Source Coding) เพื่อลดขนาดของข้อมูลเพื่อทำให้สามารถทำการส่งข้อมูลได้อย่างรวดเร็วยิ่งขึ้น หรือในกรณีที่ช่องสัญญาณมีสัญญาณรบกวน การสื่อสารในระบบดิจิทัลก็ยังสามารถจะทำการเข้ารหัสช่องสัญญาณ (Channel Coding) เพื่อควบคุมข้อผิดพลาดที่เกิดขึ้นจากสัญญาณรบกวนได้ การลดขนาดของข้อมูลโดยการเข้ารหัสต้นทางนั้นทำให้ข้อมูลมีความไวต่อการเกิดข้อผิดพลาดมากขึ้นกล่าวคือถ้ามีข้อผิดพลาดเกิดขึ้นเพียงบิตเดียวก็จะมีผลกับการถอดรหัสข้อมูลทั้งหมด ดังนั้นเมื่อมีการเข้ารหัสต้นทางจึงมักจะมีการเข้ารหัสช่องสัญญาณด้วยเสมอเพื่อควบคุมข้อผิดพลาดไม่ให้เกิดขึ้นกับข้อมูลที่ผ่านการเข้ารหัสต้นทางมาแล้ว

การเข้ารหัสสามารถแบ่งออกได้เป็น 2 ประเภทคือ การเข้ารหัสต้นทาง และการเข้ารหัสช่องสัญญาณ การเข้ารหัสทั้ง 2 ประเภท นี้มีความแตกต่างกันทั้งในแง่ของข้อดีและข้อเสีย คือการเข้ารหัสต้นทางนั้นจะถูกใช้ในการลดขนาดของข้อมูลที่จะทำการส่งโดยใช้วิธีการแปลงข้อมูลหรือสัญลักษณ์ต่างๆจากที่ทั่วไปจะใช้การเก็บในแบบที่มีการใช้จำนวนบิตเท่ากันเพื่อจัดเก็บสัญลักษณ์แต่ละสัญลักษณ์ไปใช้การเก็บข้อมูลแต่ละสัญลักษณ์จะใช้จำนวนบิตเพื่อเก็บแต่ละสัญลักษณ์ไม่เท่ากัน โดยสัญลักษณ์ที่มีความถี่ในการเกิดบ่อยครั้งกว่าจะใช้จำนวนบิตเพื่อเก็บสัญลักษณ์นั้นน้อยกว่าสัญลักษณ์ที่มีความถี่ในการเกิดต่ำ การจัดเก็บข้อมูลในลักษณะดังกล่าวนี้จะทำให้จำนวนบิตที่ใช้การเก็บข้อมูลโดยรวมลดลงแต่ก็มีข้อเสียคือทำให้เกิดการแพร่กระจายของข้อผิดพลาด (Error Propagation) ขึ้นเมื่อเกิดข้อผิดพลาดขึ้นที่จุดใดจุดหนึ่งของข้อมูลที่ทำการเข้ารหัสด้วยวิธีการเข้ารหัสต้นทาง ส่วนการเข้ารหัสช่องสัญญาณนั้นเป็นวิธีการที่ใช้ในการแก้ไขความไม่สมบูรณ์ของช่องสัญญาณ กล่าวคือเป็นการเข้ารหัสเพื่อทำให้ข้อมูลที่ได้รับนั้นสามารถตรวจจับหรือแก้ไขข้อผิดพลาดที่เกิดขึ้นจากความไม่สมบูรณ์ของช่องสัญญาณได้ แต่การเข้ารหัสช่องสัญญาณนั้นจำเป็นต้องมีการเพิ่มข้อมูลซ้ำซ้อนเข้าไปเพื่อช่วยในการตรวจจับหรือแก้ไขข้อผิดพลาดทำให้ข้อมูลที่ผ่านการเข้ารหัสช่องสัญญาณแล้วมีปริมาณของข้อมูลที่จะต้องทำการ

ส่งผ่านช่องสัญญาณมีมากขึ้น โดยปริมาณข้อมูลที่เพิ่มมากขึ้นนั้นจะแปรผันตามจำนวนของข้อผิดพลาดที่สามารถทำการตรวจจับหรือแก้ไขได้

การส่งข้อมูลที่มีปริมาณมากๆหรือข้อมูลที่มีอัตราบิตมากๆเช่นการส่งข้อมูลภาพผ่านช่องสัญญาณที่มีสัญญาณรบกวนนั้น ทำให้มีความจำเป็นที่ต้องใช้การเข้ารหัสทั้งสองประเภทเพื่อทำการลดขนาดของข้อมูลที่จะทำการส่งและทำให้สามารถตรวจจับหรือแก้ไขข้อผิดพลาดได้ ในระบบสื่อสารปกติกระบวนการเข้ารหัสต้นทางและกระบวนการเข้ารหัสช่องสัญญาณนั้นจะทำงานแยกออกจากกัน โดยจะทำการเข้ารหัสต้นทางให้เรียบร้อยก่อนจึงนำข้อมูลนั้นไปทำการเข้ารหัสช่องสัญญาณอีกครั้งหนึ่ง การทำงานในลักษณะนี้มักทำให้เกิดการประวิงเวลาขึ้นในระบบการสื่อสารเนื่องจากการทำงานของการเข้ารหัสทั้งสองส่วนแยกออกจากกัน การนำการเข้ารหัสทั้ง 2 ประเภทรวมเข้าด้วยกันทำให้การเข้ารหัสทั้ง 2 ทำการเข้ารหัสไปด้วย การทำเช่นนี้ทำให้เกิดการประวิงเวลาน้อยลง

การเข้ารหัสการเข้ารหัสเชิงเลขคณิต (Arithmetic Coding) [1] [2] [3] เป็นวิธีการเข้ารหัสต้นทางซึ่งมีการใช้กันอย่างแพร่หลายมากขึ้นในปัจจุบันเป็นการเข้ารหัสที่นำมาใช้แทนที่การเข้ารหัสฮัฟฟ์แมน (Huffman Coding) [4] ซึ่งมีประสิทธิภาพการเข้ารหัสที่ต่ำกว่าเนื่องจากการเข้ารหัสฮัฟฟ์แมนนั้นรหัสที่ใช้ในการแสดงถึงสัญลักษณ์แต่ละสัญลักษณ์ไม่สามารถที่จะมีขนาดเล็กกว่า 1 บิตได้ ในขณะที่การเข้ารหัสเชิงเลขคณิตสามารถใช้รหัสที่มีขนาดเล็กกว่า 1 บิตในการแทนสัญลักษณ์แต่ละสัญลักษณ์ได้ โดยมากแล้วการเข้ารหัสเชิงเลขคณิตมักถูกใช้เป็นขั้นตอนสุดท้ายของการเข้ารหัสภาพและเสียง การเข้ารหัสเชิงเลขคณิตถูกนำมาใช้กับการเข้ารหัสภาพและเสียงเนื่องจากการเข้ารหัสเชิงเลขคณิตเป็นการเข้ารหัสซึ่งใช้สถิติของการเกิดของสัญลักษณ์แต่ละตัวให้เป็นประโยชน์ในการเข้ารหัสและสามารถทำการเข้ารหัสและปรับแบบจำลองของข้อมูลได้อย่างต่อเนื่องได้ จึงทำให้เหมาะกับการเข้ารหัสภาพและเสียง เนื่องจากข้อมูลภาพและเสียงนั้นมักจะมีการนำไปใช้กับการส่งข้อมูลแบบต่อเนื่องซึ่งจะมีการถอดรหัสที่ภาครับไปพร้อมๆ กับการส่งข้อมูลที่ภาคส่ง ซึ่งจำเป็นต้องใช้วิธีการเข้ารหัสต้นทางซึ่งสามารถเข้าและถอดรหัสแบบต่อเนื่องได้ แต่การเข้ารหัสเชิงเลขคณิตนั้นก็มีข้อเสียเช่นเดียวกับการเข้ารหัสข้อมูลต้นทางอื่นๆคือจะมีการแพร่กระจายของข้อผิดพลาดเกิดขึ้น แม้จะเกิดข้อผิดพลาดขึ้นกับบิตเพียงบิตเดียวข้อมูลที่ทำการถอดรหัสโดยใช้ข้อมูลจากบิตที่เกิดข้อผิดพลาดขึ้นข้อมูลที่ได้จากการถอดรหัสนั้นจะไม่ใช่ข้อมูลที่ถูกต้อง หากไม่มีการใส่รหัสตรวจจับข้อผิดพลาดเข้าไปในการเข้ารหัสเชิงเลขคณิตก็จะไม่รู้ว่ามีข้อผิดพลาดเกิดขึ้นแล้ว ดังนั้นการเข้ารหัสควบคุมข้อผิดพลาดจึงมีความจำเป็นอย่างมากเมื่อมีเข้ารหัสต้นทางเพื่อลดขนาดของข้อมูลเนื่องจากเมื่อมีข้อผิดพลาดเกิดขึ้น แม้เพียงบิตเดียวในข้อมูลที่ทำการเข้ารหัสต้นทางก็จะส่งผลทำให้ข้อมูลที่ทำการถอดรหัสจากข้อมูลที่ทำการเข้ารหัสต้นทางที่มีข้อผิดพลาดเกิดขึ้นแล้วนั้นไม่ถูกต้อง

จากเหตุผลข้างต้นวิทยานิพนธ์นี้จะนำเสนอวิธีตรวจจับข้อผิดพลาดที่ใช้กระบวนการเข้ารหัสเชิงเลขคณิต โดยไม่จำเป็นต้องมีการเข้ารหัสช่องสัญญาณ วิธีการตรวจจับข้อผิดพลาดที่ได้จะสามารถทำการตรวจจับข้อผิดพลาดได้บ่อยครั้ง

1.2 วัตถุประสงค์ของงานวิจัย

เพื่อพัฒนาการเข้ารหัสต้นทางให้สามารถตรวจจับข้อผิดพลาดได้โดยไม่ทำให้ความซับซ้อนในวิธีการเข้ารหัสเพิ่มขึ้นมากและมีผลกับอัตราการบีบอัดน้อย โดยกรรมวิธีที่นำเสนอจะช่วยให้สามารถรู้ตำแหน่งของการเกิดข้อผิดพลาดขึ้นอย่างคร่าวๆได้ ซึ่งจะเป็นประโยชน์ช่วยในการแก้ข้อผิดพลาด

1.3 ขอบเขตของงานวิจัย

1. พัฒนาการเข้ารหัสเชิงเลขคณิตให้สามารถตรวจจับข้อผิดพลาดที่เกิดขึ้นได้โดยทำให้มีผลกระทบกับอัตราส่วนการบีบอัดน้อยและมีความซับซ้อนในการเข้ารหัสน้อย
2. พัฒนาการเข้ารหัสเชิงเลขคณิตซึ่งสามารถตรวจจับข้อผิดพลาดได้ให้สามารถใช้กับข้อมูลที่มีรูปแบบการกระจายแบบต่างๆ กันได้อย่างมีประสิทธิภาพ
3. การเข้ารหัสเชิงเลขคณิตจะทำโดยใช้แบบจำลองแบบปรับตัวได้
4. แบบจำลองจะจำลองข้อมูลเป็นมาร์คอฟเชนลำดับที่ 0 (0^{th} order Markov Chain)
5. วิธีการตรวจจับข้อผิดพลาดจะทำโดยการเพิ่มสัญลักษณ์ที่ใช้เป็นเครื่องหมายเข้าไปในข้อมูลเพื่อตรวจหาข้อผิดพลาดที่ฝั่งปลายทาง

1.4 วิธีการดำเนินงานวิจัย

1. ศึกษาและค้นคว้าเกี่ยวกับการเข้ารหัส โดยมีรายละเอียดดังนี้
   1.1 ศึกษาการเข้ารหัสต้นทางแบบต่างๆ
   1.2 ศึกษาการตรวจจับข้อผิดพลาดแบบต่างๆ
   1.3 การเข้ารหัสต้นทางที่มีความสามารถในการตรวจจับข้อผิดพลาดแบบอื่นๆ
2. วิเคราะห์และทดสอบประสิทธิภาพของกรรมวิธีที่ใช้ในการเข้ารหัสต้นทางที่มีความสามารถในการตรวจจับข้อผิดพลาดจากงานที่มีผู้เสนอแล้ว
3. คิดหาแนวทางใหม่ที่สามารถให้ประสิทธิภาพที่ดีขึ้น
4. ทดสอบประสิทธิภาพของวิธีการที่ได้จากการศึกษาและพัฒนาขึ้น
5. สรุป วิจารณ์และจัดทำวิทยานิพนธ์ฉบับสมบูรณ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. วิธีการเข้ารหัสต้นทางที่มีความสามารถตรวจจับข้อผิดพลาดซึ่งมีประสิทธิภาพที่สูงขึ้น
2. สามารถนำไปประยุกต์กับการเข้ารหัสภาพและเสียงที่มีการใช้การเข้ารหัสต้นทางแบบการเข้ารหัสเชิงเลขคณิต
3. เป็นแนวทางในการวิจัยการเข้ารหัสต้นทางที่มีความสามารถในการตรวจจับข้อผิดพลาดต่อไป

1.6 ภาพรวมของวิทยานิพนธ์

เนื้อหาของวิทยานิพนธ์ฉบับนี้ แบ่งออกเป็น 5 บท บทที่ 1 จะเป็นบทนำ บทที่ 2 จะกล่าวถึงทฤษฎีพื้นฐานที่เกี่ยวข้อง วิธีการเข้ารหัสเชิงเลขคณิต วิธีการตรวจจับข้อผิดพลาดที่มีการใช้กันอยู่ในปัจจุบัน ปัญหาที่เกิดในการตรวจจับข้อผิดพลาด และ การเข้ารหัสตรวจจับข้อผิดพลาดที่ใช้กระบวนการเข้ารหัสเชิงเลขคณิตที่มีการนำเสนอไปแล้ว บทที่ 3 จะกล่าวถึงแนวคิดและวิธีการที่ใช้ในงานวิจัย และเสนอวิธีการแก้ปัญหาที่เกิดขึ้น ส่วนในบทที่ 4 จะเป็นส่วนของผลการทดสอบที่ได้จากการทดสอบใช้รหัสที่ได้รับการพัฒนาขึ้นโดยจะเปรียบเทียบกับผลการทดสอบที่ได้จากการใช้วิธีการอื่นๆ ที่ได้กล่าวถึงในบทที่ 2 และในบทที่ 5 จะเป็นส่วนสรุปและอภิปรายข้อเสนอแนะข้อคิดเห็นตลอดจนแนวทางในการพัฒนาวิธีการต่อไป

บทที่ 2

เอกสารและงานวิจัยที่เกี่ยวข้อง

เนื้อหาในบทนี้กล่าวถึงทฤษฎีและงานวิจัยต่างๆที่เกี่ยวข้องกับวิทยานิพนธ์ซึ่งประกอบด้วยทฤษฎีข่าวสาร วิธีการเข้ารหัสต้นทางชนิดต่างๆ วิธีการเข้ารหัสตรวจจับข้อผิดพลาด วิธีการตรวจจับข้อผิดพลาดต่อเนื่อง และ วิธีการตรวจจับข้อผิดพลาดโดยใช้เครื่องหมาย

2.1 เอนโทรปี

เอนโทรปีเป็นแนวความคิดที่สำคัญในทฤษฎีข่าวสารและทฤษฎีสื่อสารของแชนนอน [5] ในการบีบอัดข้อมูลเราจะเริ่มจากการกำหนดขนาดของข่าวสารให้กับสัญลักษณ์แต่ละสัญลักษณ์ จากนั้นจึงสามารถที่จะทำการหาเอนโทรปีซึ่งมีค่าเท่ากับข่าวสารโดยเฉลี่ยได้ เพื่อใช้กับแหล่งข้อมูลแต่ละแหล่ง

2.1.1 การวัดข่าวสาร

ดังที่ได้กล่าวไปแล้วข้างต้น ข่าวสารได้ถูกนิยามให้เป็น ความรู้ ข้อเท็จจริงและข่าว โดยข่าวสารนั้นสามารถวัดให้อยู่ในรูปของปริมาณได้ โดยพาหะของข่าวสารนั้นจะได้มาในรูปของสัญลักษณ์ สัญลักษณ์แต่ละตัวก็จะนำข่าวสารมาในปริมาณที่มากน้อยไม่เท่ากัน ยกตัวอย่างเช่น ให้สัญลักษณ์ตัวหนึ่งมีความน่าจะเป็นในการเกิดของสัญลักษณ์นั้นมีค่าเท่ากับ p ปริมาณของข่าวสารของสัญลักษณ์นี้จะมีค่าเท่ากับ

$$I = \log_2(1/p) \text{ หรือ } I = -\log_2 p \qquad \text{บิต} \qquad (2.1)$$

เมื่อบิตเป็นปริมาณการวัดข่าวสารในระบบเลขฐานสองเนื่องจากในสมการที่ 2.1 ได้กำหนดค่าของลอการิทึมให้มีค่าเท่ากับ 2 สมการที่ 2.1 นั้นสามารถใช้หาปริมาณของข่าวสารเมื่อทำการเข้ารหัสโดยการใช้รหัสเลขโดดที่มีขนาดเท่ากับ r โดยสมการที่ 2.1จะกลายเป็น

$$I = -\log_r 2.\log_2 p \qquad \text{บิต} \qquad (2.2)$$

จากสมการที่ 2.1 จะเห็นได้ว่าปริมาณของข่าวสารนั้นจะมีค่าเท่าส่วนกลับของลอการิทึมของความน่าเป็นของการเกิดของสัญลักษณ์นั้น ยิ่งมีความน่าจะเป็นในการเกิดของสัญลักษณ์ต่ำข่าวสารที่ได้มาจากสัญลักษณ์นั้นก็จะมีค่ามาก ในทางกลับกันหากความน่าจะเป็นในการเกิดของสัญลักษณ์นั้นมีมากข่าวสารที่ได้จากการรับสัญลักษณ์นั้นก็จะมีค่าต่ำ ความน่าจะเป็นของการเกิดของสัญลักษณ์นั้นจะขึ้นอยู่กับความไม่แน่นอนของการเกิดสัญลักษณ์ที่เกิดจาก

แหล่งกำเนิดที่ไม่เหมือนกัน ความน่าจะเป็นของการเกิดของสัญลักษณ์ที่มีค่าน้อยจะมีความไม่แน่นอนในการเกิดที่มาก เช่นเดียวกันถ้าความน่าจะเป็นในการเกิดของสัญลักษณ์มีค่ามาก สัญลักษณ์นั้นจะมีความแน่นอนในการเกิดมาก ซึ่งทำให้เห็นได้ว่าปริมาณของข่าวสารที่ได้จากแต่ละสัญลักษณ์นั้นจะแปรผันกับความไม่แน่นอนในการเกิดของแต่ละสัญลักษณ์นั่นเอง

2.1.2 ปริมาณข่าวสารโดยเฉลี่ย

สมมติให้มีแหล่งข้อมูลที่มีสัญลักษณ์ที่สามารถเกิดจากแหล่งข้อมูลนี้ได้มีจำนวนทั้งหมดเท่ากับ l สัญลักษณ์ โดยการเกิดขึ้นของสัญลักษณ์แต่ละตัวนั้นจะไม่ขึ้นกับสัญลักษณ์ที่มีการเกิดขึ้นก่อน จะได้ว่าแหล่งข้อมูล $\{s_i : i = 1, 2, 3, ..., l\}$ สัญลักษณ์ที่เกิดขึ้นแต่ละตัวนั้นจะมีความน่าจะเป็นในการเกิดของสัญลักษณ์เป็นของตัวเองโดยกำหนดให้เป็น $\{p(s_i) : i = 1, 2, 3, ..., l\}$ จากที่ได้กล่าวไปแล้วข้างต้นว่าปริมาณข่าวสารนั้นจะสามารถหาได้ถ้ารู้ความน่าจะเป็นในการเกิดของสัญลักษณ์แต่ละตัว เมื่อรู้ปริมาณข่าวสารที่ได้จากแต่ละสัญลักษณ์ก็จะทำให้สามารถหาปริมาณข่าวสารโดยเฉลี่ยของแหล่งข้อมูลนั้นได้ โดยจะได้ว่าปริมาณข่าวสารโดยเฉลี่ยของแหล่งข้อมูลมีค่าเท่ากับ

$$H(p) = -\sum_{i=1}^{l} p(s_i) \log_2 p(s_i) \quad \text{บิตต่อสัญลักษณ์} \qquad (2.3)$$

ปริมาณข่าวสารโดยเฉลี่ยของแหล่งข้อมูลสามารถเรียกได้อีกชื่อหนึ่งว่า เอนโทรปี จะเห็นได้ว่าเอนโทรปีนั้นก็เป็นฟังก์ชันของความน่าจะเป็นของการเกิดของสัญลักษณ์เช่นเดียวกับปริมาณของข่าวสาร อีกทั้งสามารถเห็นได้ว่าเอนโทรปีนั้นจะมีค่ามากที่สุดก็ต่อเมื่อสัญลักษณ์ที่เกิดจากแหล่งข้อมูลนั้นมีความน่าจะเป็นในการเกิดที่เท่ากัน โดยเมื่อความน่าจะเป็นการเกิดของสัญลักษณ์แต่ละสัญลักษณ์มีค่าเท่ากันหมดหรือเป็นแหล่งข้อมูลที่มีความไม่แน่นอนในการเกิดของแต่ละสัญลักษณ์สูงที่สุดนั่นเอง ในกรณีนี้จะได้ว่าเอนโทรปีมีค่าเท่ากับ

$$H_{\max} = -\log_2(1/l) \quad \text{บิตต่อสัญลักษณ์} \qquad (2.4)$$

2.1.3 ทฤษฎีการเข้ารหัสต้นทางของแชนนอน

ในการเข้ารหัสต้นทางคำรหัสจะถูกกำหนดให้กับสัญลักษณ์แต่ละสัญลักษณ์ จำนวนของบิตที่ใช้ในแต่ละคำรหัสจะเรียกว่าความยาวของคำรหัส ความยาวโดยเฉลี่ยของคำรหัสจะเรียกว่าอัตราบิตซึ่งมีหน่วยเป็น บิตต่อสัญลักษณ์ จากทฤษฎีของแชนนอนอัตราบิตที่น้อยที่สุดที่ใช้สำหรับเข้ารหัสแหล่งข้อมูลหนึ่งจะมีค่าโดยเฉลี่ยเท่ากับเอนโทรปีของแหล่งข้อมูลนั้น ทฤษฎีของแชนนอนนั้นทำให้เรารู้ถึงขอบล่างของการเข้ารหัสต้นทาง แชนนอนได้แสดงไว้ด้วยว่าการเข้ารหัส

ให้ได้เท่ากับเอนโทรปีของข้อมูลนั้นจะทำได้ต่อเมื่อมีการเข้ารหัสที่มีการประวิงเวลาไม่จำกัด หรืออีกนัยหนึ่งคือการเข้ารหัสโดยทำการเขียนสัญลักษณ์ทั้งหมดทีเดียวหลังจากได้รับข้อมูลที่จะทำการส่งทั้งหมดแล้ว โดยในทางปฏิบัตินั้นเราสามารถทำการเข้ารหัสได้โดยใช้ความหน่วงเวลาที่จำกัดได้และยังสามารถทำการเข้ารหัสที่มีประสิทธิภาพระดับหนึ่งได้ ในหัวข้อต่อไปจะอธิบายถึงวิธีการสร้างรหัสเพื่อใช้ในการเข้ารหัสต้นทาง

2.2 การเข้ารหัสฮัฟฟ์แมน

การเข้ารหัสฮัฟฟ์แมนเป็นการเข้ารหัสที่สร้างรหัสความยาวแปรได้ที่มีความยาวของรหัสแต่ละรหัสเป็นเลขจำนวนเต็ม สัญลักษณ์ที่มีความน่าจะเป็นในการเกิดสูงจะมีความยาวของรหัสที่สั้นกว่าสัญลักษณ์ที่ทีความน่าจะเป็นในการเกิดต่ำ รหัสฮัฟฟ์แมนจะมีรหัสเติมหน้าเฉพาะตัวซึ่งทำให้สามารถทำการถอดรหัสได้อย่างถูกต้องถึงแม้ว่ารหัสจะเป็นรหัสความยาวแปรได้ การถอดรหัสของรหัสฮัฟฟ์แมนมักถูกทำโดยการใช้ต้นไม้ถอดรหัสฐานสอง (Binary Decoder Tree)

การสร้างต้นไม้ถอดรหัสฐานสองจะถูกสร้างโดยเริ่มจากใบของต้นไม้และสร้างต้นไม้กลับลงไปยังรากของต้นไม้ กระบวนการสร้างต้นไม้ทำได้โดยเริ่มจากสัญลักษณ์แต่ละสัญลักษณ์จะถูกวางไว้ที่ใบแต่ละใบของต้นไม้และจะถูกเชื่อมต่อโดยต้นไม้ฐานสอง ที่จุดต่อแต่ละจุดจะมีน้ำหนักซึ่งมีค่าเป็นจำนวนครั้งในการเกิดหรือความน่าจะเป็นในการเกิดของสัญลักษณ์แต่ละสัญลักษณ์ ต้นไม้จะถูกสร้างโดยใช้กระบวนการดังต่อไปนี้

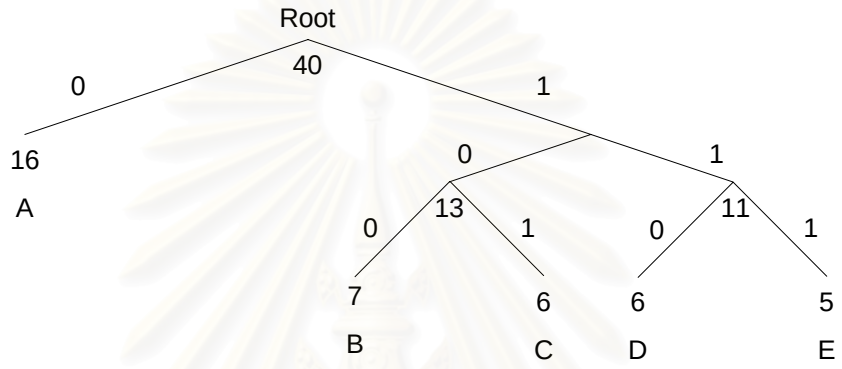
1. หาจุดต่อที่มีน้ำหนักน้อยที่สุด 2 จุด
2. สร้างจุดต่อแม่ให้กับจุดต่อทั้งสองจุดนั้น และกำหนดให้มีน้ำหนักเท่ากับน้ำหนักของจุดต่อทั้งสองรวมกัน
3. เพิ่มจุดต่อแม่เข้าไปในรายการของจุดต่อที่ยังว่างอยู่ และตัดจุดต่อลูกทั้งสองออกจากรายการ
4. จุดต่อลูกข้างหนึ่งจะถูกเลือกเป็นสัญลักษณ์เมื่อทำการถอดรหัส 0 ส่วนอีกตัวหนึ่งจะถูกใช้เมื่อทำการถอดรหัส 1
5. ทำซ้ำข้อ 1-4 จนกว่าจะเหลือจุดต่อที่ว่างอยู่เพียงจุดเดียวโดยจะกำหนดให้จุดต่อที่เหลือนี้เป็นรากของต้นไม้

ตัวอย่างการเข้ารหัสฮัฟฟ์แมนจะทำได้โดยการกำหนดสัญลักษณ์และจำนวนครั้งในการเกิดของแต่ละสัญลักษณ์ ดังตารางที่ 2.1

ตารางที่ 2.1 สัญลักษณ์และจำนวนครั้งในการเกิด

สัญลักษณ์	A	B	C	D	E
จำนวนครั้งในการเกิด	16	7	6	6	5

หลังจากทำตามขั้นตอนที่กล่าวไปแล้วจนเสร็จสิ้นเราจะได้ต้นไม้ถอดรหัสฐานสองซึ่งสามารถใช้ในการถอดรหัสสัญลักษณ์ทั้ง 5 สัญลักษณ์ได้อย่างถูกต้อง โดยผลลัพธ์ของการใช้กระบวนการดังที่กล่าวไปแล้วข้างต้นอาจได้ผลลัพธ์ดังที่แสดงไว้ในรูปที่ 2.1



รูปที่ 2.1 ตัวอย่างต้นไม้ถอดรหัสฐานสอง

เพื่อให้ได้รหัสจากต้นไม้ถอดรหัสฐานสองที่ถูกสร้างขึ้น เราทำโดยการเดินจากใบไปยังรากของต้นไม้โดยแต่ละครั้งที่เดินผ่านจุดต่อแม่จะต้องทำการรวมเอาบิตนั้นเข้ามาด้วย แต่บิตที่เราได้จากวิธีการนี้จะอยู่ในลำดับที่กลับกับลำดับที่เราต้องการ เราต้องทำการกลับลำดับของบิตที่เราได้ออกมาก่อน โดยหลังจากทำจนเสร็จสำหรับทุกสัญลักษณ์แล้วเราจะได้รหัสสำหรับแต่ละสัญลักษณ์ดังตาราง ที่ 2.2

ตารางที่ 2.2 รหัสฮัฟฟ์แมน

สัญลักษณ์	รหัส
A	0
B	100
C	101
D	110
E	111

จากตารางที่ 2.2 จะเห็นได้ว่ารหัสแต่ละตัวจะมีรหัสเติมหน้าที่แตกต่างกัน และเนื่องจากไม่มีรหัสตัวใดมีรหัสเติมหน้าเหมือนกับรหัสตัวอื่นทำให้รหัสฮัฟฟ์แมนยังสามารถทำการ

ถอดรหัสได้อย่างถูกต้องแม้จะเป็นรหัสความยาวแปรได้ อีกทั้งจะเห็นได้ว่าสัญลักษณ์ A ที่มีความน่าจะเป็นในการเกิดสูงที่สุดจะถูกกำหนดให้ใช้รหัสที่มีจำนวนบิตน้อยที่สุด และสัญลักษณ์ E ซึ่งมีความน่าจะเป็นในการเกิดต่ำที่สุดจะถูกกำหนดให้ใช้รหัสที่มีความยาวมากที่สุด จะเห็นว่าขั้นตอนวิธีการสร้างรหัสแปรความยาวได้ของฮัฟฟ์แมนนั้นสามารถที่จะสร้างรหัสแปรความยาวได้โดยความยาวสั้นที่สุดของรหัสของแต่ละสัญลักษณ์จะมีขนาดเล็กกว่า 1 บิตไม่ได้อีกทั้งรหัสจะมีขนาดเป็นเลขจำนวนเต็มหรือกล่าวคือการเข้ารหัสฮัฟฟ์แมนจะให้ผลลัพธ์ที่มีขนาดของรหัสโดยเฉลี่ยเท่ากับเอนโทรปีก็ต่อเมื่อความน่าจะเป็นของการเกิดของสัญลักษณ์ทุกตัวมีค่าเท่ากับค่ายกกำลังลบของ 2 เท่านั้น ด้วยเหตุผลข้อนี้ทำให้ รหัสฮัฟฟ์แมนไม่สามารถทำการเข้ารหัสข้อมูลบางประเภทได้อย่างมีประสิทธิภาพ เช่นการเข้ารหัสข้อมูลที่มีสัญลักษณ์ใดสัญลักษณ์หนึ่งเกิดขึ้นมากๆ หรือในกรณีที่มีสัญลักษณ์เพียง 2 สัญลักษณ์ปรากฏขึ้นในข้อมูล ในหัวข้อถัดไปจะอธิบายถึงวิธีการสร้างรหัสแปรความยาวได้อีกวิธีหนึ่งโดยวิธีการนั้นจะสามารถสร้างรหัสความยาวแปรได้ที่มีขนาดไม่เป็นจำนวนเต็มได้

2.3 การเข้ารหัสเชิงเลขคณิต

การเข้ารหัสเชิงเลขคณิตเป็นวิธีการเข้ารหัสต้นทางแบบหนึ่งที่สามารถให้ประสิทธิภาพการเข้ารหัสได้ใกล้เคียงกับเอนโทรปีที่สุด โดยการเข้ารหัสเชิงเลขคณิตนั้นอาจอธิบายได้ว่าเป็นการแปลงจากลำดับของสัญลักษณ์ไปเป็นเศษส่วนของเลขจำนวนจริงซึ่งมีค่าระหว่าง 0 กับ 1 การเข้ารหัสเชิงเลขคณิตจะสามารถทำงานได้ดีกว่าการเข้ารหัสฮัฟฟ์แมนเมื่อใช้กับข้อมูลที่มีค่าความน่าจะเป็นของการเกิดของแต่ละสัญลักษณ์แตกต่างกันมากๆ หรือมีความน่าเป็นของการเกิดสัญลักษณ์ใดมากๆ เนื่องจากการเข้ารหัสเชิงเลขคณิตสามารถที่จะเข้ารหัสข้อมูลแต่ละสัญลักษณ์โดยใช้บิตน้อยกว่า 1 บิตได้ซึ่งการเข้ารหัสฮัฟฟ์แมนไม่สามารถทำได้

ถ้าให้ N เป็นจำนวนของสัญลักษณ์ที่ได้รับเข้ามา และให้ $x_1, x_2, x_3, ..., x_N$ เป็นสัญลักษณ์ตัวที่ 1 ถึงตัวที่ N และให้ $P_1, P_2, P_3, ..., P_N$ เป็นความน่าจะเป็นในการเกิดของแต่ละสัญลักษณ์ เมื่อสัญลักษณ์แต่ละตัวถูกนำไปประมวลผลลัพธ์จะถูกจำกัดให้อยู่ในช่วง $[y, y+R)$ ซึ่งจะสามารถแบ่งออกเป็นช่วงย่อยที่ ไม่ซ้อนทับกันได้ เป็น N ช่วงดังนี้

$$
\begin{aligned}
I_1 &= [y, y + P_1 R) \\
I_2 &= [y + P_1 R, y + P_1 R + P_2 R) \\
&\vdots \\
I_N &= [y + \sum_{i=1}^{N-1} P_i R, y + R)
\end{aligned}
\tag{2.5}
$$

ซึ่งทำให้ขนาดของ I_i เท่ากับ P_iR_i เมื่อสัญลักษณ์ถัดไปคือ x_i ผลลัพธ์ที่ได้จะถูกจำกัดอยู่ในช่วง I_i และจะเห็นได้ว่าการที่ช่วงแต่ละอันไม่มีการซ้อนทับกันนั้นทำให้ผลลัพธ์ที่ได้จากการเข้ารหัสนั้นไม่ซ้ำซ้อนกันซึ่งทำให้สามารถที่จะทำการถอดรหัสกลับได้อย่างสมบูรณ์

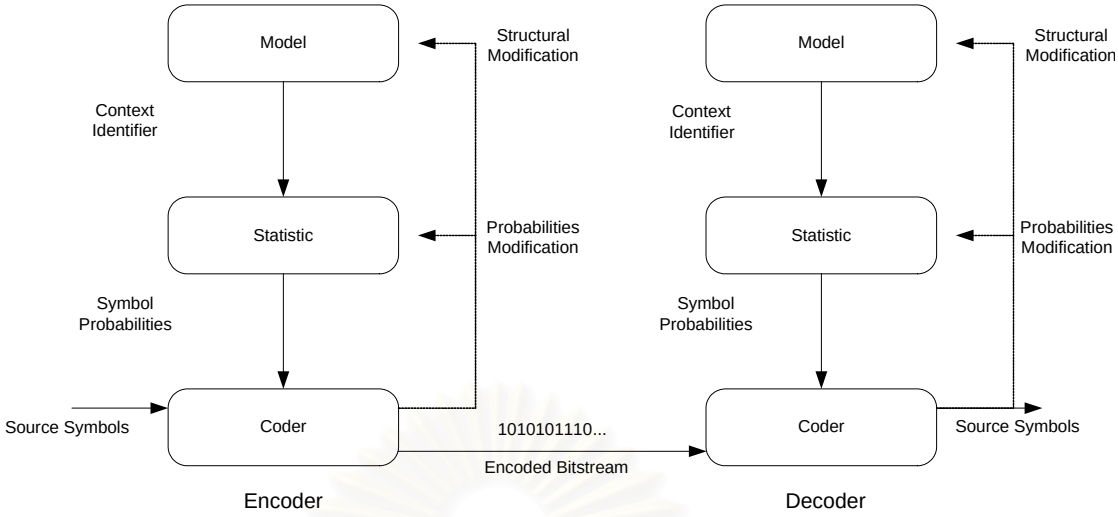
เนื่องจากตัวเข้ารหัสเชิงเลขคณิตมักถูกสร้างโดยใช้กับระบบที่ใช้เลขฐานสองทำให้ผลลัพธ์ที่ได้จากตัวเข้ารหัสจะเป็นชุดลำดับของเลขฐานสองที่สั้นที่สุดที่สามารถจะแสดงถึงจำนวนเลขสัดส่วนที่อยู่ในช่วงสุดท้ายของการเข้ารหัส

สมมติให้กระแสข้อมูลที่เข้ามาทั้งหมดมีจำนวน M สัญลักษณ์ เมื่อ x_i เกิดขึ้น P_iM ครั้งจะทำให้ขนาดของช่วงเมื่อจบขั้นตอนสุดท้ายกลายเป็น

$$\begin{aligned} R_f &= \prod_{i=1}^{N} P_i^{P_iM} \\ -\log_2 R_f &= -\log_2 \prod_{i=1}^{N} P_i^{P_iM} \\ &= \sum_{i=1}^{N} -\log_2 P_i^{P_iM} \\ &= \sum_{i=1}^{N} -P_iM \log_2 P_i \end{aligned} \tag{2.6}$$

ซึ่งจะมีขนาดเท่ากับเอนโทรปีของข้อมูลต้นฉบับ ดังนั้นจะเห็นได้ว่าการเข้ารหัสเชิงเลขคณิตเป็นการเข้ารหัสที่สามารถเข้ารหัสได้ใกล้เคียงกับเอนโทรปีที่สุด

การเข้ารหัสเชิงเลขคณิตและการถอดรหัสเชิงเลขคณิตประกอบด้วยส่วนประกอบหลักๆ 2 ส่วนคือ แบบจำลองและตัวเข้ารหัสเชิงเลขคณิตดังแสดงไว้ในรูปที่ 2.2 ส่วนประกอบทั้งสองส่วนมีหน้าที่และการทำงานที่แยกออกจากกันโดยอิสระคือ ส่วนแบบจำลองจะมีหน้าที่ในการบริหารและจัดการการเก็บข้อมูลเพื่อสร้างตารางความน่าจะเป็นเพื่อใช้ในการเข้ารหัสโดยข้อมูลตารางความน่าจะเป็นจะถูกส่งไปที่ตัวเข้ารหัสเชิงเลขคณิตซึ่งจะมีหน้าที่ในการแปลงสัญลักษณ์ที่รับเข้ามาให้กลายเป็นตัวเลขเศษส่วนที่มีความสอดคล้องกับลำดับของสัญลักษณ์และตารางความน่าจะเป็นที่ใช้ในการเข้ารหัส



รูปที่ 2.2 ส่วนประกอบของการเข้ารหัสเชิงเลขคณิต

จากรูปที่ 2.1 จะเห็นว่าแบบจำลองและตัวเข้ารหัสเชิงเลขคณิตนั้นทำงานแยกอิสระจากกันซึ่งทำให้เราสามารถใช้แบบจำลองที่ซับซ้อนหรือมีเป็นแบบจำลองที่มีลักษณะเป็นมาร์คอฟเชนลำดับสูงๆ ได้ การที่เราสามารถที่จะใช้แบบจำลองที่มีความซับซ้อนมากๆได้นั้นทำให้เราสามารถจะทำการบีบอัดข้อมูลได้อย่างมีประสิทธิภาพมากกว่าการใช้แบบจำลองซึ่งมีลักษณะเป็นมาร์คอฟเชนลำดับต่ำๆ โดยยิ่งมีลำดับของมาร์คอฟเชนสูงขึ้นเท่าไรก็จะมีความต้องการหน่วยความจำในการจัดเก็บตารางความน่าจะเป็นและใช้ความซับซ้อนในการคำนวณมากขึ้นเพื่อแลกเปลี่ยนกับอัตราส่วนการบีบอัดที่เพิ่มขึ้น แบบจำลองยังอาจแบ่งออกได้เป็น 3 ประเภทดังที่แสดงไว้ในหัวข้อ 2.3.1 ส่วนตัวเข้ารหัสเชิงเลขคณิตที่มีหน้าที่ในการแปลงข้อมูลให้กลายเป็นเลขเศษส่วนจำนวนจริงจะอธิบายในรายละเอียดในหัวข้อ 2.3.2

2.3.1 แบบจำลอง

แบบจำลองที่ใช้กับการเข้ารหัสต้นทางที่ใช้ข้อมูลทางสถิติจะสามารถแบ่งประเภทของแบบจำลองออกได้เป็น 3 ประเภทดังที่ได้อธิบายไว้ในบทความ[6] ดังต่อไปนี้

2.3.1.1 แบบจำลองแบบตายตัว (Static Model)

แบบจำลองแบบตายตัวจะใช้วิธีการกำหนดค่าความน่าจะเป็นของแต่ละสัญลักษณ์ไว้ล่วงหน้าและใช้ค่าความน่าจะเป็นที่กำหนดไว้แล้วกับข้อมูลทุกรูปแบบที่เข้ามาที่ตัวเข้ารหัส การทำแบบจำลองแบบตายตัวอาจทำได้โดยการสุ่มเลือกข้อมูลจำนวนมากที่เป็นข้อมูลแบบเดียวกับข้อมูลที่จะทำการเข้ารหัสมาเพื่อหาความน่าจะเป็นของการเกิดของสัญลักษณ์แต่ละสัญลักษณ์เพื่อให้ได้ความน่าจะเป็นที่ดีในกรณีที่ใช้แบบจำลองชนิดนี้ควรมีการสุ่มข้อมูลจำนวนมากเท่าที่จะสามารถทำได้เพื่อให้ได้ความน่าจะเป็นที่ใกล้เคียงกับข้อมูลที่จะทำการเข้ารหัสที่สุด

การใช้แบบจำลองแบบตายตัวมีข้อดีคือเป็นแบบจำลองที่มีความเร็วสูงที่สุด เนื่องจากไม่มีส่วนของการคิดคำนวณเพิ่มเข้ามาในการทำแบบจำลอง แต่การใช้แบบจำลองแบบตายตัวมีข้อเสียคือการใช้ค่าความน่าจะเป็นที่กำหนดไว้แล้วกับข้อมูลทุกแบบเนื่องจากข้อมูลจะให้ประสิทธิภาพของการเข้ารหัสต้นทางมีน้อยลง และในบางกรณีอาจทำให้ข้อมูลที่เข้ารหัสแล้วมีขนาดใหญ่กว่าเดิมได้หากข้อมูลที่ทำการเข้ารหัสมีความน่าจะเป็นไม่ตรงกับแบบจำลอง

2.3.1.2 แบบจำลองแบบกึ่งปรับตัวได้ (Semi-Adaptive Model)

ในแบบจำลองแบบกึ่งปรับตัวได้จะใช้วิธีการตรวจดูข้อมูลทั้งหมดก่อนที่จะเริ่มทำการเข้ารหัส โดยจะนำผลลัพธ์ที่ได้จากการตรวจดูข้อมูลเรียบร้อยแล้วนั้นไปใช้ในการสร้างแบบจำลองดังนั้นความน่าจะเป็นในแบบจำลองที่ได้จะเหมือนกับข้อมูลที่ต้องการจะทำการเข้ารหัสโดยสมบูรณ์ แบบจำลองแบบกึ่งปรับตัวได้ยังแบ่งออกเป็น 2 แบบ คือ

1. แบบจำลองแบบกึ่งปรับตัวได้แบบตายตัว (Static Semi-Adaptive Model)

แบบจำลองแบบกึ่งปรับตัวได้แบบตายตัวจะใช้ผลลัพธ์ที่ได้จากการตรวจดูข้อมูลไปใช้เป็นแบบจำลองโดยไม่มีการเปลี่ยนแปลงแบบจำลองอีกจนจบการเข้ารหัส

2. แบบจำลองแบบกึ่งปรับตัวได้แบบหักออก (Decrementing Semi-Adaptive Model)

แบบจำลองแบบกึ่งปรับตัวได้แบบหักออกจะใช้ผลลัพธ์ที่ได้จากการตรวจดูข้อมูลไปใช้เป็นแบบจำลองตั้งต้นแล้วทำการลดความน่าจะเป็นของสัญลักษณ์ตัวที่เข้ารหัสไปแล้วลง โดยการทำเช่นนี้จะทำให้ได้ความน่าจะเป็นที่ใกล้เคียงกับความน่าจะเป็นจริงมากที่สุด

แบบจำลองแบบกึ่งปรับตัวได้มีข้อดีคือให้ผลลัพธ์ในการเข้ารหัสที่มีประสิทธิภาพสูง เนื่องจากมีการตรวจดูข้อมูลทั้งหมดก่อนทำการเข้ารหัส แบบจำลองแบบกึ่งปรับตัวได้มีข้อเสียคือการตรวจดูข้อมูลก่อนนั้นไม่คุ้มค่าและไม่สามารถทำได้ในบางกรณีเช่น การถ่ายทอดสด อีกข้อหนึ่งคือค่าความน่าจะเป็นที่ได้มานั้นจะต้องส่งข้อมูลนั้นไปยังปลายทางด้วยจึงจะสามารถทำการถอดรหัสได้ และในบางกรณีข้อมูลนี้มีปริมาณมากกว่าปริมาณของข้อมูลที่จะทำการเข้ารหัส

2.3.1.3 แบบจำลองแบบปรับตัวได้

แบบจำลองแบบปรับตัวได้นั้นจะมีการปรับตัวอยู่ตลอดเวลาที่มีการเข้ารหัส โดยมากจะมีการปรับตัวทุกครั้งที่ได้รับสัญลักษณ์ใหม่เข้ามา โดยความน่าจะเป็นที่ได้จากการใช้แบบจำลองแบบนี้จะใกล้เคียงกับความน่าจะเป็นของจริงของข้อมูลมากเนื่องจากมีการปรับเปลี่ยน

ตารางความน่าจะเป็นให้สอดคล้องกับข้อมูลที่ได้รับอยู่ตลอดเวลา ซึ่งจะมักให้ได้ผลลัพธ์การเข้ารหัสที่ดีกว่าแบบจำลองอื่นๆ

แบบจำลองแบบปรับตัวได้มีข้อดีคือให้ประสิทธิภาพในการเข้ารหัสสูง นอกจากนี้ยังไม่จำเป็นต้องมีการส่งข้อมูลสถิติไปยังตัวถอดรหัสก่อนจึงจะทำการถอดรหัสได้ เนื่องจากหากตั้งให้ค่าเริ่มต้นของแบบจำลองทั้งสองฝั่งตรงกันก็จะทำให้สามารถถอดรหัสได้ แบบจำลองแบบปรับตัวได้มีข้อเสียคือมีความซับซ้อนของการเข้ารหัสเพิ่มมากขึ้นเนื่องจากมีการปรับตัวอย่างต่อเนื่องทำให้มีโอกาสที่แบบจำลองกับข้อมูลที่รับเข้ามาเพื่อทำการเข้ารหัสจะไม่สัมพันธ์กันลดลง แบบจำลองชนิดนี้จึงเป็นแบบจำลองที่ได้รับความนิยม

2.3.2 ตัวเข้ารหัสเชิงเลขคณิตและตัวถอดรหัสเชิงเลขคณิต

ตัวเข้ารหัสเชิงเลขคณิตมีหน้าที่ในการแปลงข้อมูลสัญลักษณ์ที่เข้ามาให้กลายเป็นเลขเศษส่วนที่มีค่าอยู่ในช่วง 0 ถึง 1 โดยเลขเศษส่วนที่ทำการแปลงไปนั้นจะสอดคล้องกับข้อมูลที่ได้จากตารางความน่าจะเป็นที่ได้จากแบบจำลอง กล่าวคือประสิทธิภาพของการเข้ารหัสจะขึ้นกับแบบจำลองข้อมูลเนื่องจากหากแบบจำลองสามารถที่จะทำนายการเกิดของข้อมูลตัวถัดไปได้ถูกต้องมากขึ้นเพียงใดก็จะทำให้ข้อมูลถูกเข้ารหัสได้อย่างมีประสิทธิภาพมากขึ้นเท่านั้นแต่ก็ต้องแลกกับความซับซ้อนในการคำนวณที่ต้องใช้มากขึ้น

ตัวเข้ารหัสเชิงเลขคณิตมีกระบวนการเข้ารหัสดังต่อไปนี้คือ

1. ทำการกำหนดช่วงเริ่มต้นให้มีค่าเท่ากับ [0,1)
2. ทำการแบ่งช่วงปัจจุบันให้แบ่งออกเป็นช่วงย่อยที่มีความสอดคล้องกับความน่าเป็นที่ได้รับมาจากแบบจำลอง
3. เลือกช่วงย่อยสำหรับสัญลักษณ์ตัวที่เข้ามา และกำหนดให้ช่วงนั้นกลายเป็นช่วงปัจจุบัน
4. ทำซ้ำข้อ 2 และข้อ 3 สำหรับทุกๆสัญลักษณ์ที่ได้รับมา
5. เมื่อข้อมูลที่ได้รับเข้ามาทั้งหมดถูกประมวลด้วยวิธีข้างต้น ผลลัพธ์ที่ได้คือรหัสใดๆที่สามารถจะอธิบายถึงช่วงปัจจุบันได้ หรือกล่าวคือรหัสที่ได้จะมีค่าอยู่ในช่วงปัจจุบัน

ส่วนตัวถอดรหัสเชิงเลขคณิตจะมีหน้าที่ในการเปลี่ยนจากคำรหัสที่ได้รับซึ่งแสดงถึงเลขจำนวนจริง โดยการถอดรหัสเชิงเลขคณิตจะมีลักษณะเป็นการเลียนแบบตัวเข้ารหัสเชิงเลขคณิต การถอดรหัสของการเข้ารหัสเชิงเลขคณิตสามารถทำได้ดังต่อไปนี้

1. ทำการกำหนดช่วงเริ่มต้นให้มีค่าเท่ากับ [0,1)
2. ทำการแบ่งช่วงปัจจุบันให้แบ่งออกเป็นช่วงย่อยที่มีความสอดคล้องกับความน่าเป็นที่ได้รับมาจากแบบจำลอง

3. เลือกช่วงย่อยของสัญลักษณ์โดยดูจากรหัสเชิงเลขคณิตที่ได้รับซึ่งมีข้อกำหนดคือช่วงที่ทำการเลือกจะต้องเป็นช่วงที่บรรจุตัวเลขที่แปรจากรหัสเชิงเลขคณิตเอาไว้ในช่วง และทำการกำหนดให้ช่วงที่ทำการเลือกนั้นเป็นช่วงปัจจุบัน

4. ทำซ้ำข้อ 2 และ ข้อ 3 จนกว่าจะพบสัญลักษณ์สิ้นสุดแฟ้ม (End of File Symbol)

เมื่อทำการประมวลสัญลักษณ์แต่ละสัญลักษณ์ช่วงจะมีขนาดเล็กลงเรื่อยๆ ซึ่งทำให้จำเป็นต้องใช้จำนวนบิตมากขึ้นสำหรับการแสดงถึงตัวเลขในที่อยู่ในช่วงปัจจุบัน การเข้ารหัสเชิงเลขคณิตอาจทำได้ดังตัวอย่างต่อไปนี้ ตัวอย่างเช่น มีความต้องการจะส่งข้อมูลที่มีสัญลักษณ์ดังต่อไปนี้คือ {a,e,i,o,u,!} และมีการใช้แบบจำลองตายตัวโดยมีการกำหนดความน่าจะเป็นไว้ดังตารางที่ 2.3

ตารางที่ 2.3 แบบจำลองตายตัวสำหรับสัญลักษณ์ {a,e,i,o,u,!}

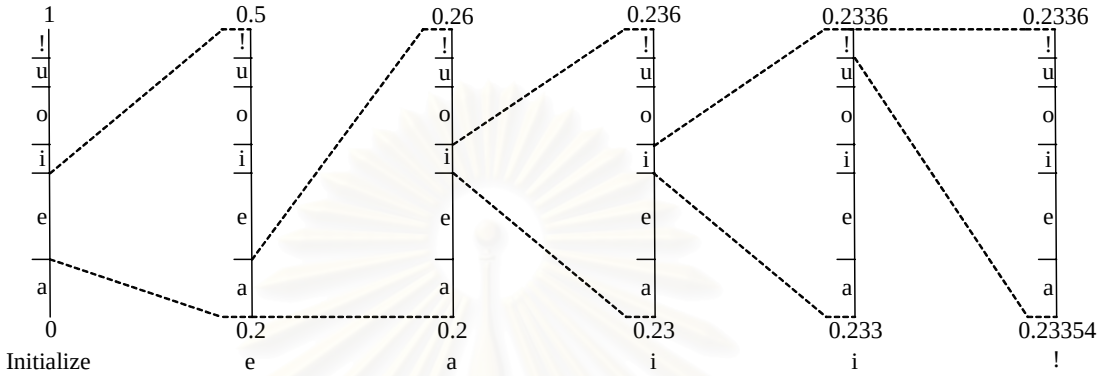
สัญลักษณ์	ความน่าจะเป็น	ช่วง
a	0.2	[0.0,0.2)
e	0.3	[0.2,0.5)
i	0.1	[0.5,0.6)
o	0.2	[0.6,0.8)
u	0.1	[0.8,0.9)
!	0.1	[0.9,1.0)

สมมติว่าข้อมูลที่จะทำการส่งคือ “eaii!” ในตอนเริ่มต้นทั้งตัวเข้ารหัสและตัวถอดรหัสจะรู้ว่าช่วงคือ [0,1) หลังจากได้รับสัญลักษณ์ตัวแรก คือ e จะทำให้ช่วงเล็กลงกลายเป็น [0.2,0.5) เมื่อได้รับสัญลักษณ์ตัวที่ 2 คือ a ซึ่งอยู่ในช่วง [0.0,0.2) จะทำให้ช่วงกลายเป็น [0.2,0.26) เมื่อได้รับสัญลักษณ์ตัวที่ 3 คือ I ซึ่งอยู่ในช่วง [0.5,0.6) จะทำให้ช่วงกลายเป็น [0.23,0.236) ทำเช่นเดียวกันนี้ต่อไป จะสามารถเขียนได้ดังตารางที่ 2.4

ตารางที่ 2.4 ช่วงที่ใช้ในการเข้ารหัสเชิงเลขคณิตของข้อมูล eaii!

เริ่มต้น	[0,1)
หลังจากได้รับ e	[0.2, 0.5)
หลังจากได้รับ a	[0.2, 0.26)
หลังจากได้รับ i	[0.23, 0.236)
หลังจากได้รับ i	[0.233, 0.2336)
หลังจากได้รับ !	[0.23354, 0.2336)

หลังจากที่เข้ารหัสเสร็จเรียบร้อยแล้วส่งไปที่ฝั่งตัวเข้ารหัสจะต้องสร้างรหัสฐานสองที่เป็นตัวเลขที่มีค่าอยู่ในช่วง [0.2334, 0.2336) เพื่อส่งไปให้กับทางฝั่งตัวถอดรหัสการเข้ารหัสเชิงเลขคณิตสามารถเขียนได้อีกรูปแบบหนึ่งคือตามรูปที่ 2.3 จะทำให้เห็นภาพรวมของวิธีการเข้ารหัสได้ชัดเจนยิ่งขึ้น ดังนี้



รูปที่ 2.3 ขั้นตอนการเข้ารหัสเชิงเลขคณิต

การเข้ารหัสเชิงเลขคณิตของข้อมูล “eaii!” เมื่อทำจนถึงข้อมูลตัวสุดท้าย เราจะได้ว่ารหัสผลลัพธ์ที่ได้จะสามารถแปรเป็นตัวเลขที่อยู่ภายในช่วง [0.2334, 0.2336) ซึ่งรหัสที่สั้นที่สุดคือ 00111011110011 เมื่อตัวเลขฐานสองที่มีค่า 1 แต่ละหลักแทนด้วยค่า $1/2^{-n}$ เมื่อ n คือเลขหลักของแต่ละหลัก หากต้องการจะทำการถอดรหัสเราก็จะทำตามขั้นตอนดังที่ได้กล่าวไปแล้วข้างต้นเราก็จะได้ข้อมูลกลับคืนมา เมื่อตัวถอดรหัสได้รับรหัสดังกล่าวจะสามารถรู้ได้ทันทีว่าสัญลักษณ์ตัวแรกคือ e เนื่องจากช่วงที่ได้รับอยู่ในช่วงที่จองไว้สำหรับสัญลักษณ์ e ซึ่งจะทำให้ตัวถอดรหัสนั้นสามารถที่จะจำลองการทำงานของตัวเข้ารหัสได้ ทำการถอดรหัสในลักษณะเดียวกันนี้ต่อไปจะทำให้สามารถได้เข้ามูลที่ถูกเข้ารหัสไว้กลับมาได้ทั้งหมดการถอดรหัสของรหัสเชิงเลขคณิตที่ไม่มีข้อผิดพลาดเกิดขึ้นเราจะได้ผลลัพธ์เป็นข้อมูลชุดเดียวกับก่อนที่จะทำการเข้ารหัสหรือได้ว่าการถอดรหัสจะทีใช้ช่วงย่อยดังต่อไปนี้

ตารางที่ 2.5 ช่วงที่ใช้ในการถอดรหัสเชิงเลขคณิตของข้อมูล eaii!

เริ่มต้น	[0,1)
หลังจากเลือก e	[0.2, 0.5)
หลังจากเลือก a	[0.2, 0.26)
หลังจากเลือก i	[0.23, 0.236)
หลังจากเลือก i	[0.233, 0.2336)
หลังจากเลือก !	[0.23354, 0.2336)

จากตารางที่ 2.4 และตารางที่ 2.5 จะเห็นได้ว่าช่วงที่ถูกใช้ในการเข้ารหัสและการถอดรหัสเป็นช่วงเดียวกันและการถอดรหัสก็เป็นกระบวนการย้อนกลับของการเข้ารหัสนั่นเอง ซึ่งส่วนที่ต่างกันคือการเข้ารหัสนั้นเราจะเลือกช่วงที่จะใช้จากข้อมูลที่จะทำการส่งส่วนการถอดรหัสเราจะเลือกช่วงที่เราจะใช้จากรหัสเชิงเลขคณิตที่ได้รับ โดยขั้นการถอดรหัสก็สามารถแสดงเป็นรูปได้เช่นเดียวกับรูปที่ 2.3 ซึ่งแสดงขั้นตอนการเข้ารหัส แต่ในกรณีที่รหัสที่ส่งไปแล้วภาครับไม่สามารถรับได้อย่างถูกต้องแล้วทำให้เกิดมีข้อผิดพลาดเกิดขึ้นขั้นตอนและช่วงที่ใช้ในการถอดรหัสจะต่างออกไปโดยสิ้นเชิง ซึ่งทำให้ข้อมูลที่ได้จากการถอดรหัสของข้อมูลที่มีข้อผิดพลาดจะเป็นข้อมูลที่ได้นั้นถูกต้องและไม่สามารถนำมาใช้งานได้ ในส่วนต่อไปจะอธิบายถึงการแพร่กระจายของข้อผิดพลาดของการเข้ารหัสเชิงเลขคณิตซึ่งทำให้เกิดความเสียหายกับข้อมูลที่ถอดรหัสมาจากรหัสเชิงเลขคณิตที่มีข้อผิดพลาด

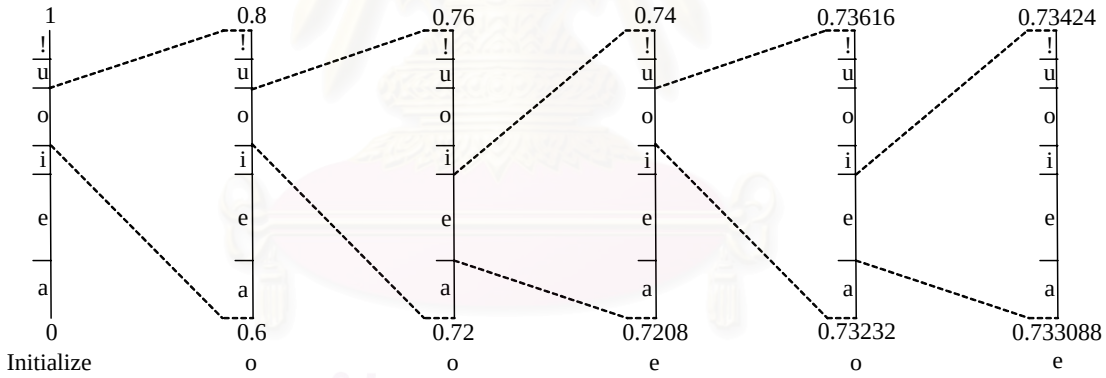
2.3.3 การแพร่กระจายของข้อผิดพลาดของการเข้ารหัสเชิงเลขคณิต

การเข้ารหัสเชิงเลขคณิตสามารถทำการเข้ารหัสได้ใกล้เคียงกับเอนโทรปีมากที่สุด แต่การเข้ารหัสเชิงเลขคณิตเองก็ยังมีจุดอ่อนอยู่เนื่องจากการเข้ารหัสเชิงเลขคณิตจะเป็นวิธีการที่ทำการแปลงลำดับข้อมูลทั้งหมดที่รับเข้ามาให้กลายเป็นรหัสตัวเดียวที่สามารถอธิบายถึงข้อมูลทั้งหมดได้ดังนั้นหากรหัสที่ได้รับนั้นไม่ตรงกับรหัสที่สร้างจากตัวเข้ารหัสข้อมูลที่ได้จากการถอดรหัสจะมีความถูกต้องถึงส่วนของข้อมูลก่อนที่จะมีข้อผิดพลาดเกิดขึ้นครั้งแรกเท่านั้น การแพร่กระจายของข้อผิดพลาดของการเข้ารหัสเชิงเลขคณิตจะทำให้ข้อผิดพลาดที่เกิดขึ้นกับรหัสเชิงเลขคณิตที่อาจเกิดขึ้นกับบิตเพียงบิตเดียวสร้างความเสียหายให้กับข้อมูลที่ทำการถอดรหัสทั้งหมด การแพร่กระจายของข้อผิดพลาดจะสามารถทำความเสียหายให้กับข้อมูลได้มากที่สุดเมื่อเกิดข้อผิดพลาดกับบิตข้อมูลแรกของรหัสเพราะจะทำให้การตัดสินใจเลือกสัญลักษณ์ผิดตั้งแต่ตัวแรก การแพร่กระจายของข้อผิดพลาดของการเข้ารหัสเชิงเลขคณิตอาจสามารถยกตัวอย่างให้ดูได้ดังนี้ จากการเข้ารหัสที่ผ่านมาเราได้ทำการเข้ารหัสเชิงเลขคณิตกับข้อมูล “eaii!” จากการเข้ารหัสนั้นเราจะได้รหัสที่จะทำการส่งว่าเป็นกระแสข้อมูลฐานสองคือ 00111011110011 หรือข้อมูลที่แปรเป็นตัวเลขแล้วคือ 0.23358154296875 หากมีข้อผิดพลาดเกิดขึ้นที่บิตแรกของรหัสจะทำให้ข้อมูลที่ฝั่งรับกลายเป็น 10111011110011 หรือข้อมูลที่แปรเป็นตัวเลขแล้วกลายเป็น 0.73358154296875 ซึ่งทำให้มีข้อผิดพลาดเกิดขึ้นทำให้ไม่สามารถทำการถอดรหัสข้อมูลที่ถูกต้องทั้งหมดได้ การถอดรหัสของข้อมูลชุดนี้อาจทำได้ดังต่อไปนี้ ขั้นแรกทำการแปรจะรหัสฐานสองที่ได้รับเป็นตัวเลขจะได้เป็น 0.73358154296875 ซึ่งจะต้องทำการเลือกสัญลักษณ์จากตัวเลขนี้โดยมีข้อจำกัดว่าจะต้องเป็นช่วงที่มีตัวเลขที่แปรออกมาอยู่ในช่วงนั้นโดยสัญลักษณ์ที่ถูกเลือกออกมาและช่วงที่ถูกเลือกจะเป็นดังที่แสดงไว้ในตารางที่ 2.6

ตารางที่ 2.6 ช่วงที่ถูกเลือกในการถอดรหัสเชิงเลขคณิตที่มีข้อผิดพลาดเกิดขึ้นที่บิตแรก

เริ่มต้น	[0, 1)
หลังจากเลือก o	[0.6, 0.8)
หลังจากเลือก o	[0.72, 0.76)
หลังจากเลือก e	[0.7208, 0.74)
หลังจากเลือก o	[0.73232, 0.73616)
หลังจากเลือก e	[0.733088, 0.73424)

จะเห็นได้ว่าการถอดรหัสเชิงเลขคณิตที่มีข้อผิดพลาดเกิดขึ้นที่บิตเดียวจะทำให้เกิดความเสียหายให้กับข้อทั้งที่มีความเกี่ยวพันธ์กับรหัสตัวเดิมดังนั้นโดยในกรณีนี้การที่เกิดข้อผิดพลาดขึ้นที่ข้อมูลบิตแรกทำให้ตัวเลขซึ่งใช้เป็นรหัสเปลี่ยนจาก 0.23358154296875 ไปเป็น 0.73358154296875 ส่งผลให้ตัวถอดรหัสไม่สามารถทำการถอดรหัสข้อมูลที่ถูกต้องออกมาได้ ข้อมูลที่ถอดรหัสออกมาได้นั้นจะมีความน่าจะเป็นในการเกิดสอดคล้องกับแบบจำลองข้อมูลที่ตัวถอดรหัสใช้อยู่โดยในกรณีนี้ข้อมูลที่ถอดรหัสได้จะเปลี่ยนจาก “eaii!” ไปเป็น “ooeoe” โดยขั้นตอนการถอดรหัสเป็นไปตามที่ได้แสดงไว้ในรูปที่ 2.4



รูปที่ 2.4 ขั้นการถอดรหัสเชิงเลขคณิตที่มีข้อผิดพลาดเกิดขึ้นที่บิตแรก

ตัวอย่างแรกนั้นข้อผิดพลาดได้เกิดขึ้นที่บิตข้อมูลบิตแรกทำให้ข้อมูลทั้งหมดเสียหายไป แต่หากข้อผิดพลาดเกิดขึ้นที่บิตข้อมูลที่มีความสำคัญน้อยลงไปความรุนแรงของความเสียหายที่เกิดขึ้นก็จะลดน้อยลงเนื่องจากจะมีข้อมูลบางส่วนที่ไม่จำเป็นต้องใช้บิตข้อมูลที่เกิดข้อผิดพลาดในการตัดสินใจเลือกช่วงที่จะใช้ซึ่งทำให้มีข้อมูลที่สามารถนำไปใช้งานได้จะเท่ากับ

$$M = \frac{L}{H} - 1 \tag{2.7}$$

เมื่อ L เป็นจุดที่ข้อผิดพลาดตัวแรกที่เกิดขึ้นกับรหัสเชิงเลขคณิต

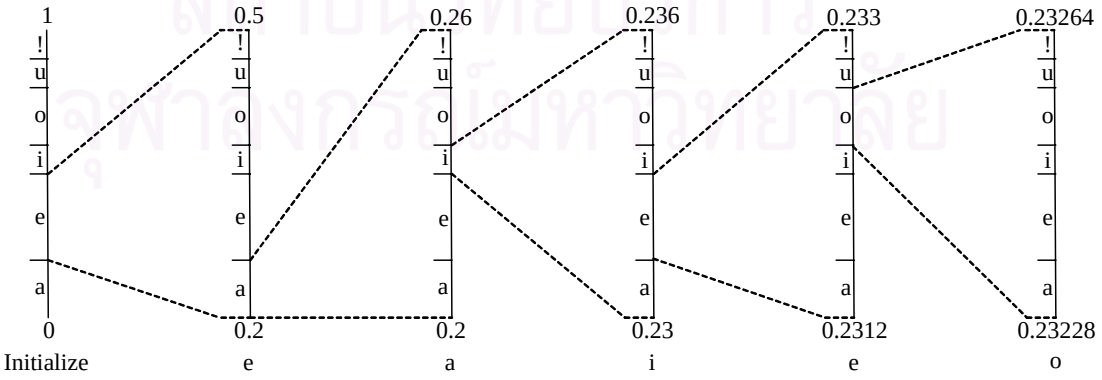
H เป็นเอนโทรปีของข้อมูล

กล่าวคือยิ่งจุดที่เกิดข้อผิดพลาดขึ้นที่แรกอยู่ไกลเพียงไรข้อมูลที่สามารถถอดรหัสได้อย่างถูกต้องก็จะมีมากขึ้นทั้งนี้ข้อมูลที่สามารถถอดรหัสได้อย่างถูกต้องนี้ จะขึ้นอยู่กับเอนโทรปีของข้อมูลนั้นด้วยเนื่องจากหากเอนโทรปีของข้อมูลมีค่ามากหรือข้อมูลมีการกระจายของสัญลักษณ์ที่เท่าเทียมกันจะทำให้ข้อมูลที่สามารถถอดรหัสได้อย่างถูกต้องมีจำนวนลดลง การคำนวณข้อมูลที่สามารถถอดรหัสได้อย่างถูกต้องอาจทำได้ดังตัวอย่างต่อไปนี้ สมมติให้มีการส่งข้อมูลเดิมที่ใช้ในตัวอย่างก่อนๆ โดยครั้งนี้ข้อมูลที่ได้รับคือ 00111011100011 ซึ่งข้อมูลที่ควรจะได้รับคือ 00111011110011 นั่นคือเกิดข้อผิดพลาดขึ้นที่บิตที่ 10 ทำให้เลขที่แปรจากข้อมูลที่ได้รับเปลี่ยนจาก 0.23358154296875 ไปเป็น 0.23260498046875 ทำให้การถอดรหัสในครั้งนี้สามารถถอดรหัสได้ดังต่อไปนี้

ตารางที่ 2.7 ช่วงที่ถูกเลือกในการถอดรหัสเชิงเลขคณิตที่มีข้อผิดพลาดเกิดขึ้นที่บิตที่ 10

เริ่มต้น	[0,1)
หลังจากเลือก e	[0.2, 0.5)
หลังจากเลือก a	[0.2, 0.26)
หลังจากเลือก i	[0.23, 0.236)
หลังจากเลือก e	[0.2312, 0.233)
หลังจากเลือก o	[0.23228, 0.23264)

จากตารางที่ 2.7 จะเห็นได้ว่าสัญลักษณ์ที่ได้จากการถอดรหัสของข้อมูลที่มีความผิดพลาดเกิดขึ้นที่บิตที่ 10 นี้จะมีข้อมูลบางส่วนซึ่งไม่เกี่ยวข้องกับข้อมูลส่วนที่เกิดข้อผิดพลาดขึ้นทำให้ข้อมูลส่วนนั้นสามารถทำการถอดรหัสออกมาได้อย่างถูกต้องแต่ข้อมูลที่ทำการถอดรหัสออกมาจากข้อมูลส่วนที่เกิดข้อผิดพลาดขึ้นแล้วสัญลักษณ์ที่ได้ออกมาจะเป็นสัญลักษณ์ที่ไม่ถูกต้อง การถอดรหัสนี้สามารถเขียนเป็นขั้นตอนได้ดังนี้



รูปที่ 2.5 ขั้นการถอดรหัสเชิงเลขคณิตที่มีข้อผิดพลาดเกิดขึ้นที่บิตที่ 10

จาก ความน่าจะเป็นในตารางที่ 2.3 เอนโทรปีของข้อมูลชุดนี้จะมีค่าเท่ากับ $H = 0.3\times\log_2(0.3) + 2\times 0.2\log_2(0.2) + 3\times 0.1\log_2(0.1) = 2.44643934467102$ ซึ่งทำให้สามารถคำนวณปริมาณข้อมูลที่สามารถถอดรหัสได้อย่างถูกต้องซึ่งมีค่าเท่ากับ $M = 10/2.44643934467102 - 1 = 3.08757324058845 \approx 3$ ซึ่งจะเท่ากับที่ได้เห็นจากตัวอย่างที่ถอดรหัสได้สัญลักษณ์ “eaieo” จะเห็นว่าข้อมูลที่สามารถถอดรหัสได้อย่างถูกต้องมีเพียง 3 สัญลักษณ์เท่านั้น แต่ก็แสดงให้เห็นว่าตำแหน่งของการเกิดข้อผิดพลาดครั้งแรกมีผลกับระดับของการแพร่กระจายของข้อผิดพลาดคือยิ่งมีข้อผิดพลาดเกิดขึ้นเร็วเพียงไรการแพร่กระจายของข้อผิดพลาดก็จะเกิดขึ้นเร็วขึ้นเท่านั้น

จากตัวอย่างที่ผ่านมาซึ่งเกิดข้อผิดพลาดขึ้นที่บิตแรกของข้อมูลที่ผ่านการเข้ารหัสเชิงเลขคณิตแล้ว ข้อผิดพลาดดังกล่าวทำให้เกิดข้อผิดพลาดขึ้นกับข้อมูลที่ทำการถอดรหัสออกมาทั้งหมด แต่ความผิดพลาดนั้นอาจไม่เกิดขึ้นเสมอไปเนื่องจากหากพิจารณาให้ดีจะให้ว่าผลลัพธ์ที่ได้จากการถอดรหัสจากรหัสเชิงเลขคณิตที่เกิดข้อผิดพลาดขึ้นจะให้ผลลัพธ์เป็นชุดสัญลักษณ์ที่มีความน่าเป็นในการเกิดของสัญลักษณ์ตรงกับความน่าจะเป็นที่มีอยู่ในตารางซึ่งถูกนำไปใช้โดยตัวถอดรหัส หรือพูดได้ว่าโอกาสที่เกิดข้อผิดพลาดขึ้นแล้วสัญลักษณ์ที่ถอดรหัสออกมาได้ยังเป็นสัญลักษณ์เดิมจะมีความน่าเป็นเท่ากับความน่าจะเป็นในการเกิดของสัญลักษณ์ที่ตำแหน่งนั้นโดยสามารถเขียนเป็นสมการได้ว่า

$$P_{actual}(s) = P_{table}(s) \tag{2.8}$$

เมื่อ s เป็นสัญลักษณ์ตัวที่กำลังทำการถอดรหัส

จากสมการนี้หมายความว่าหากข้อมูลที่จะทำการส่งมีความน่าเป็นในการเกิดเท่ากับค่าที่ถูกกำหนดไว้สำหรับใช้ในการเข้ารหัสในตารางความน่าจะเป็น แล้วจะได้ว่าหากเกิดข้อผิดพลาดขึ้นในข้อมูลที่ผ่านการเข้ารหัสเชิงเลขคณิตข้อมูลที่มีความน่าจะเป็นในการเกิดสูงจะมีโอกาสที่จะถอดรหัสได้สัญลักษณ์ที่ถูกต้องที่ตำแหน่งที่ถูกต้อง

2.3.4 การนำการเข้ารหัสเชิงเลขคณิตไปใช้งานจริง

การเข้ารหัสเชิงเลขคณิตที่ได้อธิบายไปแล้วในส่วนที่ผ่านมาเป็นการเข้ารหัสเชิงเลขคณิตที่ไม่สามารถสร้างขึ้นเพื่อใช้งานจริงได้เนื่องจากเมื่อเราทำการเลือกช่วงที่เราจะต้องใช้ในการสร้างตัวเลขฐานสองเพื่อใช้เป็นรหัสเราไม่สามารถที่จะทำการจัดเก็บช่วงของข้อมูลที่มีความละเอียดมากได้เนื่องจากข้อจำกัดของฮาร์ดแวร์ เนื่องจากฮาร์ดแวร์จะมีการใช้หน่วยความจำที่มีขนาดจำกัดทำให้ไม่สามารถที่จะทำการจัดเก็บช่วงที่มีความละเอียดมากๆ ได้ ดังนั้นจำเป็นต้องมีการปรับขนาดของช่วงให้อยู่ในขนาดที่หน่วยความจำสามารถจะทำการจัดเก็บได้โดยทำได้

ดังต่อไปนี้ เมื่อช่วงที่ใช้ในการเข้ารหัสมีขนาดเล็กลงจะมีโอกาสเป็นไปได้ที่ช่วงจะเป็นไปตามลักษณะดังต่อไปนี้

1. ช่วงที่ใช้ทั้งหมดอยู่ในครึ่งล่างของช่วงหนึ่งหน่วย $[0, 0.5)$
2. ช่วงที่ใช้ทั้งหมดอยู่ในครึ่งบนของช่วงหนึ่งหน่วย $[0.5,1)$
3. ช่วงที่ใช้อยู่ตรงกลางของช่วงหนึ่งหน่วย

ในกรณีที่ 1 และกรณีที่ 2 จะเห็นได้ว่าบิตที่มีนัยสำคัญสูงที่สุดจะมีค่าเป็น 0 และ 1 ตามลำดับดังนั้นเมื่อมาถึงกรณีนี้จะเห็นได้ว่าไม่ว่าต่อไปกระบวนการเข้ารหัสจะได้รับข้อมูลใดเข้ามาบิตที่มีนัยสำคัญสูงที่สุดนี้ก็จะไม่มีการเปลี่ยนแปลงค่าเกิดขึ้นดังนั้นเราสามารถที่จะทำการดึงบิตที่มีนัยสูงสุดออกมาทำการส่งจากนั้นจึงทำการปรับขนาดของช่วงโดยการปรับขนาดของช่วงเก่า x_{n-1} ไปเป็นช่วงใหม่ x_n จะเป็นดังนี้คือ

$$x_n = 2 \times x_{n-1} \qquad \text{สำหรับกรณีที่1}$$

และ $$x_n = 2(x_{n-1} - 0.5) \qquad \text{สำหรับกรณีที่ 2}$$

ส่วนในกรณีที่ 3 นั้นไม่มีความจำเป็นที่จะต้องมีการปรับช่วงเนื่องจากช่วงยังสามารถที่เกิดการเปลี่ยนแปลงบิตที่มีนัยสำคัญสูงสุดได้อยู่ ดังนั้นการปรับช่วงโดยไม่มีการสูญเสียข้อมูลจึงไม่สามารถเกิดขึ้นได้ จากวิธีการดังที่ได้กล่าวมานี้ทำให้การเข้ารหัสเชิงเลขคณิตสามารถหลีกเลี่ยงปัญหาเกี่ยวกับหน่วยความจำที่จำกัดได้ อีกทั้งการเข้ารหัสยังสามารถทำการเข้ารหัสแบบไปข้างหน้าได้เนื่องจากไม่มีความจำเป็นที่จะต้องทำการเข้ารหัสข้อมูลทั้งหมดก่อนแล้วจึงจะสามารถเริ่มต้นส่งข้อมูลได้ เนื่องจากข้อมูลที่ได้ออกมาจากวิธีการนี้สามารถที่จะทำการส่งไปได้ทันทีเพราะจะไม่มีการเปลี่ยนแปลงข้อมูลในส่วนนั้นอีกแล้ว

ปัญหาอีกข้อหนึ่งของการนำการเข้ารหัสเชิงเลขคณิตไปใช้คือการหยุดการถอดรหัสเนื่องจากการถอดรหัสเชิงเลขคณิตนั้นสามารถดำเนินต่อไปได้ไม่มีที่สิ้นสุดเนื่องจากเป็นการเลือกช่วงที่ตรงกับข้อมูลที่ได้รับเท่านั้นแม้ไม่มีข้อมูลใดๆ เข้ามาก็ยังสามารถให้ผลการถอดรหัสออกมาได้ตลอด ดังนั้นจึงจำเป็นต้องมีวิธีการในการบอกให้การถอดรหัสนั้นหยุดลงซึ่งอาจทำได้โดยการใช้สัญลักษณ์สิ้นสุดแฟ้มซึ่งเป็นสัญลักษณ์พิเศษเพื่อบอกให้ตัวถอดรหัสหยุดทำการถอดรหัสเมื่อพบสัญลักษณ์ตัวนี้ แต่ผลจากการใช้สัญลักษณ์พิเศษนี้คือจำเป็นจะต้องมีการจองเนื้อที่บางส่วนในตารางความน่าจะไปใช้สำหรับสัญลักษณ์สิ้นสุดแฟ้มซึ่งทำให้ประสิทธิภาพโดยรวมลดลงอีกทั้งยังมีความจำเป็นต้องมีการเข้ารหัสสัญลักษณ์สิ้นสุดแฟ้มเพิ่มเข้าไปหลังข้อมูลที่จะทำการเข้ารหัสอีกด้วย และเนื่องจากสัญลักษณ์สิ้นสุดแฟ้มนั้นถูกใช้เพียงครั้งเดียวในการเข้ารหัสข้อมูลแต่ละครั้งปริมาณของบิตที่ถูกใช้ในการแทนสัญลักษณ์สิ้นสุดแฟ้มก็จะมีมากไปด้วยเนื่องจาก

สัญลักษณ์สิ้นสุดแฟ้มจะมีความน่าจะเป็นในการเกิดต่ำมากซึ่งทำให้สูญเสียประสิทธิภาพไปอีกบางส่วน

2.4 การตรวจจับข้อผิดพลาด

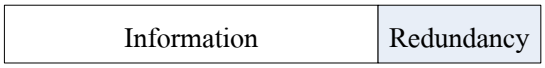
จุดมุ่งหมายของการเข้ารหัสตรวจจับข้อผิดพลาดมีขึ้นเพื่อทำให้ภาครับสามารถที่จะทำการตรวจสอบได้ว่าข้อมูลที่ทำการส่งผ่านช่องสัญญาณที่มีสัญญาณรบกวนนั้นเกิดความเสียหายขึ้นหรือไม่ และควบคุมระดับของข้อผิดพลาดที่เกิดขึ้นให้อยู่ในระดับที่ยอมรับได้ โดยการเข้ารหัสตรวจจับข้อผิดพลาดอาจทำได้โดยการส่งข้อมูลขึ้นมาอีกชุดหนึ่งซึ่งเรียกว่า ผลรวมตรวจสอบ(Checksum) ซึ่งคิดคำนวณขึ้นมาจากฟังชั่นของข้อมูลที่ทำการส่ง เมื่อสร้างผลรวมตรวจสอบขึ้นมาแล้วเราจะต้องทำการส่งผลรวมตรวจสอบที่ได้นั้นไปกับข้อมูลด้วย เมื่อภาครับได้รับข้อมูลทั้งหมดเรียบร้อยแล้วภาครับก็จะสามารถทำการตรวจสอบข้อมูลที่ได้รับกับผลรวมตรวจสอบได้โดยการสร้างผลรวมตรวจสอบโดยการใช้ฟังก์ชั่นเดียวกับที่ภาคส่งแล้วทำการเปรียบเทียบกับผลรวมตรวจสอบที่ได้รับเพื่อตรวจสอบว่าผลรวมตรวจสอบที่ได้นั้นเท่ากันหรือไม่ ถ้าผลรวมตรวจสอบทั้งสองเท่ากันจะได้ว่าข้อมูลทั้งสองเป็นข้อมูลชุดเดียวกัน ตัวอย่างเช่นถ้าเราเลือกฟังก์ชั่นผลรวมตรวจสอบซึ่งเป็นฟังก์ชั่นง่ายๆ ตัวอย่างเช่นเป็นผลรวมของตัวเลขของข้อมูล ดังนั้นการตรวจจับข้อผิดพลาดอาจเป็นดังต่อไปนี้ เมื่อตัวเลขในตัวอย่างเป็นเลขฐานสิบ

ข้อมูล	6 23 4
ข้อมูลและผลรวมตรวจสอบ	6 23 4 33
ข้อมูลหลังจากการส่ง	6 27 4 33

รูปที่ 2.5 ตัวอย่างการตรวจจับข้อผิดพลาดอย่างง่าย

ในตัวอย่างด้านบนข้อมูลไบต์ที่สองถูกทำให้เปลี่ยนจาก 23 เป็น 27 โดยช่องสัญญาณ อย่างไรก็ดีในกรณีนี้ภาครับสามารถที่จะทำการตรวจสอบข้อมูลชุดนี้ได้โดยการเปรียบเทียบผลรวมตรวจสอบ ซึ่งในกรณีนี้เท่ากับ 33 กับผลรวมตรวจสอบที่ทำการคำนวณขึ้นใหม่ซึ่งมีค่าเท่ากับ 6+27+4 เท่ากับ 37 ภาครับจะสามารถรู้ได้ว่าข้อมูลที่รับมาได้นั้นมีข้อผิดพลาดเกิดขึ้น ในกรณีที่ถ้าตัวผลรวมตรวจสอบเองเกิดความเสียหายขึ้นจากการส่งผ่านช่องสัญญาณแต่ข้อมูลที่ทำการส่งไปนั้นไม่มีความเสียหายข้อมูลนั้นก็จะถูกตัดสินให้เป็นข้อมูลที่เสียหายด้วยเช่นเดียวกับข้อมูลที่เกิดความเสียหายขึ้นกับตัวข้อมูลเอง ความล้มเหลวในการตรวจจับข้อผิดพลาดอาจเกิดขึ้นในกรณีที่มีควาล้มเหลวเกิดขึ้นในตัวข้อมูลแต่ผลรวมตรวจสอบที่คำนวณได้และผลรวมตรวจสอบที่ได้รับยังคงมีค่าเท่ากันในกรณีนี้การที่จะหลีกเลี่ยงเหตุการณ์นี้อาจทำได้โดยการขยายขนาดของตัวผลรวมตรวจสอบให้มีขนาดใหญ่ยิ่งขึ้นซึ่งจะทำให้มีโอกาสที่ผลรวมตรวจสอบจะมีค่าเท่ากันลดลง แต่ก็ต้องมีการส่งข้อมูลมากขึ้นไปด้วย การเข้ารหัสตรวจจับข้อผิดพลาดอาจทำได้หลากหลายวิธีแต่

วิธีการที่มักถูกใช้จะมีลักษณะเป็นการส่งข้อมูลแล้วตามด้วยข้อมูลที่ใช้ในการควบคุมข้อผิดพลาด กล่าวคือการตรวจจับข้อผิดพลาดแต่ละครั้งจะทำได้ก็ต่อเมื่อภาครับได้รับข้อมูลและรหัสตรวจจับข้อผิดพลาดเรียบร้อยทั้งหมดแล้ว การตรวจจับข้อผิดพลาดโดยวิธีการนี้เราสามารถเขียนข้อมูลที่ทำการส่งในแต่ละครั้งได้ดังนี้



รูปที่ 2.6 รูปแบบของข้อมูลที่ทำการส่งแต่ละครั้ง

จากรูปที่ 2.6 จะเห็นว่าข้อมูลทั้งหมดที่ต้องทำการส่งในกรณีที่มีการใช้รหัสตรวจจับข้อผิดพลาดจะประกอบด้วยส่วนแรกจะเป็นข้อมูลเดิมที่ยังไม่ได้มีการเปลี่ยนแปลงใดๆ รวมกับรหัสตรวจจับข้อผิดพลาดซึ่งเป็นส่วนซ้ำซ้อนของข้อมูล

จากตัวอย่างของผลรวมตรวจสอบที่ได้กล่าวไปแล้ว เราได้แสดงให้เห็นแล้วว่าเราสามารถที่จะทำการตรวจจับข้อผิดพลาดที่เกิดขึ้นในข้อมูลได้อย่างไร ปัญหาที่เกิดขึ้นกับวิธีการนี้คือวิธีการนี้ง่ายเกินไป เนื่องจากถ้าข้อผิดพลาดเกิดขึ้นมากกว่า 1 จุดก็จะมีโอกาส 1 ใน 256 ที่ข้อผิดพลาดที่เกิดขึ้นจะไม่ถูกตรวจพบเมื่อใช้ความซ้ำซ้อนที่มีความยาว 8 บิต ดังตัวอย่างต่อไปนี้

ข้อมูล	6 23 4
ข้อมูลและผลรวมตรวจสอบ	6 23 4 33
ข้อมูลหลังจากการส่ง	8 20 4 33

รูปที่ 2.7 ตัวอย่างการตรวจจับข้อผิดพลาดอย่างง่ายที่ตรวจจับข้อผิดพลาดล้มเหลว

เพื่อทำให้ความสามารถในการตรวจจับข้อผิดพลาดมีโอกาสที่จะตรวจไม่พบข้อผิดพลาดน้อยลงเราอาจทำได้โดยการเพิ่มขนาดของส่วนซ้ำซ้อนจาก 8 บิตเป็น 16 บิตซึ่งจะทำให้โอกาสที่จะเกิดความล้มเหลวในการตรวจจับข้อผิดพลาดลดลงจาก 1/256 เป็น 1/65536 หรือเขียนได้เป็นสมการดังต่อไปนี้

$$P_{Failure} = \frac{1}{2^R} \tag{2.9}$$

เมื่อ R เป็นจำนวนของบิตที่ใช้เป็นส่วนซ้ำซ้อน

แต่การเพิ่มขนาดของส่วนซ้ำซ้อนเพียงอย่างเดียวนั้นไม่สามารถที่จะทำให้โอกาสที่จะเกิดการตรวจจับล้มเหลวได้ได้โดยสมบูรณ์เนื่องจากหากวิธีการที่เราใช้ในการคำนวณค่าที่ใช้ในการตรวจสอบนั้นไม่มีความสุ่มที่เพียงพออันเนื่องมาจากวิธีการคำนวณที่ง่ายเกินไปหรือวิธีการคำนวณนั้นไม่ได้ใช้ประโยชน์จากจำนวนส่วนซ้ำซ้อนที่เพิ่มขึ้นได้ ปัญหาอันนี้อาจแก้ได้โดยการ

แทนวิธีการคำนวณที่ง่ายนั้นด้วยวิธีการที่ซับซ้อนกว่าแต่สามารถใช้ประโยชน์จากส่วนซ้ำซ้อนที่เพิ่มขึ้นได้สูงสุด ดังนั้นเราจะเห็นได้ว่ามีสิ่งที่จำเป็น 2 อย่างเพื่อที่จะได้ผลรวมตรวจสอบที่ดี ดังนี้

1. ขนาดของส่วนซ้ำซ้อนยิ่งขนาดของส่วนซ้ำซ้อนมีมากเท่าไรโอกาสที่จะเกิดการตรวจจับที่ล้มเหลวก็จะมีน้อยลงเท่านั้น

2. ระดับความสุ่มของวิธีการคำนวณหรือความสามารถในการเปลี่ยนแปลงบิตแต่ละบิตของส่วนซ้ำซ้อน

ดังที่ได้กล่าวมาแล้วข้างต้นการสร้างรหัสตรวจจับข้อผิดพลาดที่ดีต้องการระดับของความสุ่มที่สูงพอ เราสามารถใช้วิธีการต่างๆ ได้มากมายตัวอย่างเช่น การใช้ค่าของตัวเลขในสมุดโทรศัพท์และใช้ข้อมูลที่ต้องการส่งเป็นตัวชี้ หรืออาจใช้วิธีการอื่นๆ ได้อีกมากแต่เราไม่จำเป็นต้องใช้วิธีการที่ยุ่งยากขนาดนั้นแต่ในเมื่อการบวกเป็นวิธีการที่ง่ายและไม่มีประสิทธิภาพเพียงพอวิธีการต่อไปจากการบวกก็คือการคูณหรือหาร เมื่อตัวหารมีขนาดยาวใกล้เคียงกับขนาดของส่วนซ้ำซ้อน วิธีการหนึ่งที่ใช้เป็นวิธีการคำนวณส่วนซ้ำซ้อนซึ่งใช้ในการตรวจจับข้อผิดพลาดคือวิธีการซีอาร์ซี แนวคิดพื้นฐานของซีอาร์ซีคือการปฏิบัติกับข้อมูลแต่ละตัวในรูปแบบของเลขฐานสองแล้วทำการหารเลขตัวนี้ด้วยตัวเลขอื่นแล้วทำให้เศษเหลือของการหารกลายเป็นผลรวมตรวจสอบของข้อมูลชุดนี้ เมื่อทำการส่งผ่านข้อมูลชุดนี้เรียบร้อยแล้วที่ภาครับก็สามารถทำการหารแบบเดียวกันเพื่อหาผลรวมตรวจสอบเพื่อเปรียบเทียบกับผลรวมตรวจสอบที่ได้รับ ตัวอย่างของการคำนวณผลรวมตรวจสอบอาจทำได้ดังนี้ สมมติให้ข้อมูลที่ต้องการส่งมีขนาด 2 ไบต์คือ 6 และ 23 เช่นเดียวกับในตัวอย่างที่ผ่านมา ซึ่งสามารถเขียนเป็นเลขฐานสิบหก ได้เท่ากับ 06 17 และสามารถเขียนเป็นเลขฐานสอง 0000011000010111 สมมติว่าเราใช้ส่วนซ้ำซ้อนยาวเท่ากับ 1 ไบต์และใช้ตัวหารคงที่ซึ่งมีค่าเท่ากับ 1001 ซึ่งมีค่าเท่ากับ 9 หากทำการคำนวณในรูปแบบของเลขฐานสิบเราจะได้ว่า 1559 หารด้วย 9 จะเท่ากับ 173 เศษ 2 สิ่งที่เราต้องการในที่นี้คือเศษ 2 ซึ่งสามารถเขียนในรูปแบบ ฐานสองได้เป็น 0010 จะเห็นได้ว่าเมื่อมีข้อมูลเข้ามามากกว่านี้ค่าของเศษเหลือจะเปลี่ยนแปลงไปมากซึ่งต่างจากการบวกซึ่งไม่สามารถทำให้เกิดการเปลี่ยนแปลงที่มากเพียงพอเมื่อส่วนซ้ำซ้อนมีขนาดใหญ่และสามารถอ้างถึงจำนวนตัวเลขได้เป็นจำนวนมากๆ

การที่จะทำการคำนวณหาค่าของซีอาร์ซี เราจะต้องเริ่มจากการเลือกตัวหารหรือที่เรียกว่าพหุนามตัวก่อกำเนิด (Generator Polynomial) ซึ่งเป็นตัวแปรสำคัญของขั้นตอนวิธีซีอาร์ซี เราสามารถที่จะเลือกพหุนามตัวก่อกำเนิดใดก็ได้ แต่อย่างไรก็ดีพหุนามตัวก่อกำเนิดบางตัวสามารถทำงานได้ดีกว่าพหุนามตัวก่อกำเนิดตัวอื่นๆ อีกส่วนหนึ่งของที่เป็นตัวแปรสำคัญของขั้นตอนวิธีซีอาร์ซีคือการเลือกความกว้างของพหุนามตัวก่อกำเนิดซึ่งจะมีค่าเท่ากับตำแหน่งของบิตที่มีค่าเท่ากับ 1 ที่สูงที่สุด โดยปกติแล้วเรามักจะเลือกความกว้างของพหุนามตัวก่อกำเนิดเท่ากับ 16 หรือ 32 ซึ่งพหุนามตัวก่อกำเนิดที่มีความกว้างเท่านี้มักจะถูกในไปประยุกต์ใช้งานกับคอมพิวเตอร์ ความกว้างของพหุนามตัวก่อกำเนิดจะเท่ากับตำแหน่งของบิตสูงสุด ตัวอย่างเช่น พหุนาม $x^4 + x + 1$ ซึ่ง

จะแทนด้วย 10011 จะมีความกว้างของพหุนามเท่ากับ 4 หลังจากเลือกพหุนามตัวก่อกำเนิดเรียบร้อยแล้วเราจะสามารถเริ่มการหารข้อมูลด้วยพหุนามตัวก่อกำเนิดที่ได้เลือกไว้ โดยก่อนเริ่มทำการหารเราจะเพิ่มบิต 0 ตามหลังข้อมูลเข้าไปอีกเท่ากับความกว้างของพหุนามตัวก่อกำเนิดเช่นตัวอย่างต่อไปนี้

ข้อมูลต้นฉบับ	1101011011
พหุนามตัวก่อกำเนิด	10011
ข้อมูลหลังจากเติมบิต 0	11010110110000

รูปที่ 2.8 วิธีการเติมบิต 0 ให้กับข้อมูลก่อนการหาร

หลังจากทำการเติมบิต 0 แล้วเราสามารถเริ่มการหารได้โดยการหารเพื่อหาค่าของซีอาร์ซีจะทำได้ดังต่อไปนี้

```
          1100001010   = Quotient
10011 ) 11010110110000   = Data Message  (1101011011+0000)
        10011
        10011
        10011
        00001
        00000
        00010
        00000
        00101
        00000
        01011
        00000
        10110
        10011
        01010
        00000
        10100
        10011
        01110
        00000
        1110   = Remainder = CRC
```

รูปที่ 2.9 การหาค่าของซีอาร์ซี

ผลลัพธ์ที่ได้จากการทำการหารข้อมูลด้วยพหุนามตัวก่อกำเนิดประกอบด้วยผลหารซึ่งเป็นส่วนที่เราไม่สนใจกับส่วนของเศษเหลือซึ่งเป็นค่าของซีอาร์ซีที่ต้องการนำไปใช้ต่อท้ายจากข้อมูลที่จะทำการส่ง หลังจากที่ได้ทำการหาซีอาร์ซีและทำการเติมซีอาร์ซีต่อท้ายข้อมูลเป็นที่เรียบร้อยแล้วเราจะได้ข้อมูลที่จะทำการส่งทั้งหมดกลายเป็น 11010110111110 เมื่อเราทำการส่งข้อมูลชุดนี้ไปยังภาครับ สิ่งที่ภาครับสามารถทำได้มี 2 วิธีเพื่อตรวจจับข้อผิดพลาดคือ

1. ทำการแยกข้อมูลที่ได้รับออกเป็น 2 ส่วนคือ ส่วนที่เป็นข้อมูลต้นฉบับและส่วนที่ใช้เป็นซีอาร์ซี หลังจากนั้นทำการคำนวณหาซีอาร์ซีจากข้อมูลที่แยกออกมาเพื่อทำการเปรียบเทียบกับซีอาร์ซีที่ได้รับจากภาคส่ง
2. ทำการคำนวณหาซีอาร์ซีจากข้อมูลทั้งหมดที่ได้รับโดยหากข้อมูลที่ได้รับนั้นไม่มีข้อผิดพลาดเกิดขึ้นซีอาร์ซีที่หาได้จากข้อมูลทั้งหมดที่ได้รับจะต้องมีค่าเท่า 0

วิธีการตรวจสอบทั้งแบบจะให้ผลที่เหมือนกันเสมอแต่ในการนำไปใช้จริงแล้ววิธีการที่ 2 เป็นที่นิยมมากกว่าเนื่องจากไม่จำเป็นต้องขั้นตอนการแยกข้อมูลออกเป็น 2 ส่วน ก่อนที่จะทำการตรวจสอบ ทำให้การตรวจสอบสามารถทำได้อย่างรวดเร็วกว่าวิธีการแรก

การเลือกพหุนามตัวก่อกำเนิดเราจะเลือกโดยคำนึงถึงรูปแบบของข้อผิดพลาดที่เกิดขึ้น พหุนามตัวก่อกำเนิดที่ถูกนำมาใช้กันอย่างแพล่งหลายมีดังต่อไปนี้

ในกรณีความกว้างของพหุนามก่อกำเนิดเท่ากับ 16

มาตรฐาน X25 $x^{16}+x^{12}+x^{5}+1$

ซีอาร์ซี 16 $x^{16}+x^{12}+x^{2}+1$

ในกรณีความกว้างของพหุนามก่อกำเนิดเท่ากับ 32

ซีอาร์ซี 32 $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$

การเข้ารหัสตรวจจับข้อผิดพลาดโดยการใช้ขั้นตอนวิธีซีอาร์ซีจำเป็นต้องมีการรอจนกระทั่งข้อมูลทั้งหมดรวมทั้งส่วนซ้ำซ้อนได้ถูกส่งมาถึงภาครับจึงจะสามารถทำการตรวจจับข้อผิดพลาดได้วิธีการที่จะนำเสนอในส่วนต่อไปจะเป็นวิธีการตรวจจับข้อผิดพลาดที่มีการใช้กระบวนการเข้ารหัสของการเข้ารหัสเชิงเลขคณิตให้เป็นประโยชน์ทำให้การตรวจจับข้อผิดพลาดสามารถทำได้บ่อยครั้งมากขึ้น

2.5 การตรวจจับข้อผิดพลาดต่อเนื่อง (Continuous Error Detection)

จากที่เราได้แสดงตัวอย่างการตรวจจับข้อผิดพลาดแบบที่นิยมใช้กันนั้นไม่สามารถที่จะทำการตรวจจับข้อผิดพลาดได้บ่อยครั้ง เนื่องจากการตรวจจับข้อผิดพลาดแต่ละครั้งจำเป็นต้องมีส่วนซ้ำซ้อนที่ใช้เพื่อการตรวจจับข้อผิดพลาดเข้าไปทำให้มีข้อมูลที่จำเป็นต้องทำการส่งทั้งหมดมี

จำนวนมากขึ้นตามไปด้วย หากต้องการทำการตรวจจับข้อผิดพลาดให้บ่อยครั้งขึ้นก็จำเป็นต้องจำนวนของบิตซ้ำซ้อนเพิ่มมากขึ้นไปด้วย วิธีการตรวจจับข้อผิดพลาดต่อเนื่องนี้จะใช้การคำนวณของการเข้ารหัสเชิงเลขคณิตเป็นฐานในการสร้างรหัสที่ใช้ในการตรวจจับเข้าผิดพลาดอีกทั้งยังใช้เป็นรหัสที่แสดงถึงข้อมูลด้วยในรหัสเดียวกันวิธีการทำเช่นนี้ทำให้การเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องนั้นมีคุณสมบัติทั้งของการเข้ารหัสต้นทางและของการเข้ารหัสช่องสัญญาณด้วย การที่เราสามารถนำการเข้ารหัสทั้งสองมารวมกันได้นั้นทำให้เราสามารถลดการประวิงเวลาที่เกิดขึ้นจากการแยกขั้นตอนของการเข้ารหัสต้นทางและการเข้ารหัสช่องสัญญาณออกจากกัน เนื่องจากโดยปกติแล้วการเข้ารหัสต้นทางและการเข้ารหัสช่องสัญญาณนั้นจะทำแยกจากกันอย่างชัดเจนและมักจำเป็นต้องรอให้ทำการเข้ารหัสต้นทางให้เสร็จสิ้นก่อนจากนั้นจึงค่อยนำผลลัพธ์ที่ได้จากการเข้ารหัสช่องสัญญาณมาทำการเข้ารหัสช่องสัญญาณอีกครั้งหนึ่ง ซึ่งจะทำให้เกิดการประวิงเวลาขึ้นในระหว่างที่รอให้กระบวนการเข้ารหัสต้นทางเสร็จสิ้น ส่วนการเข้ารหัสตรวจจับข้อผิดพลาดแบบต่อเนื่องจะทำการเข้ารหัสข้อมูลครั้งเดียวแต่การเข้ารหัสครั้งเดียวนั้นรหัสที่ได้จะมีคุณสมบัติทั้งของการเข้ารหัสต้นทางและการเข้ารหัสช่องสัญญาณ แต่การเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องเองก็จะลดความสามารถในการบีบอัดข้อมูลลงเมื่อมีการเพิ่มความซ้ำซ้อนขึ้นเพื่อให้การตรวจจับมีประสิทธิภาพสูงขึ้น

การเข้ารหัสเชิงเลขคณิตสามารถที่จะทำการถอดรหัสข้อมูลที่เข้ารหัสมาด้วยการเข้ารหัสเชิงเลขคณิตโดยการย้อนกลับกระบวนการเข้ารหัส อย่างไรก็ดีเมื่อเกิดข้อผิดพลาดขึ้นกับข้อมูลที่ถูกเข้ารหัสด้วยการเข้ารหัสเชิงเลขคณิต ข้อผิดพลาดนี้จะทำให้เกิดการสูญเสียการประสานเวลา (Synchronization) ซึ่งทำให้เกิดการแพร่กระจายของข้อผิดพลาดและทำให้ผลลัพธ์การถอดรหัสของข้อมูลหลังจากส่วนที่เกิดข้อผิดพลาดขึ้นจะเกิดข้อพลาดขึ้นด้วย และทำให้ข้อมูลที่ผ่านการถอดรหัสมาส่วนนี้ไม่สามารถนำมาใช้งานได้ แต่ก็เป็นการแพร่กระจายของข้อผิดพลาดนี้เองที่สามารถนำมาใช้ในการสร้างวิธีการตรวจจับข้อผิดพลาดได้ แนวคิดของการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องมาจากการใส่สัญลักษณ์ที่ไม่มีการใช้สัญลักษณ์นั้นภายในข้อมูลเข้าไปในตารางความน่าจะเป็นสัญลักษณ์ที่เพิ่มเข้าไปนี้จะเรียกว่า สัญลักษณ์ต้องห้าม (Forbidden Symbol) เนื่องจากตัวถอดรหัสรู้ว่าที่ตัวเข้ารหัสไม่เคยปรากฏสัญลักษณ์ต้องห้าม ดังนั้นเมื่อทำการถอดรหัสแล้วพบสัญลักษณ์ต้องห้ามตัวถอดรหัสก็จะรู้ได้ว่าข้อผิดพลาดได้เกิดขึ้นแล้ว แนวคิดอันนี้ถูกนำเสนอไว้ในบทความ [7] [8]และ [9] เมื่อทำการถอดรหัสข้อมูลที่มีข้อผิดพลาดเกิดขึ้น

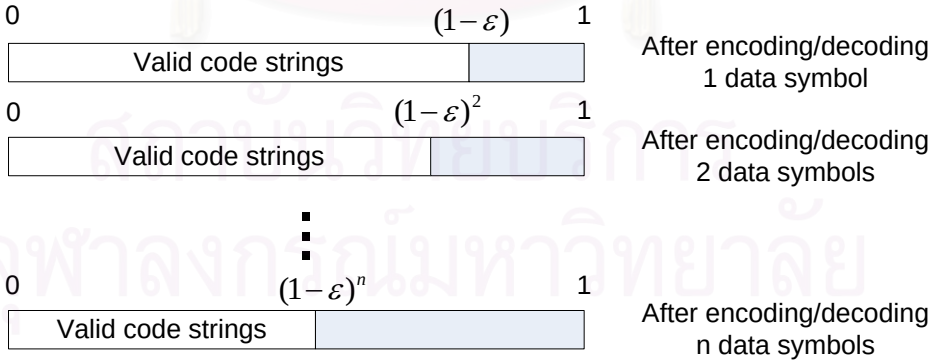
การเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องจำเป็นต้องมีการเพิ่มสัญลักษณ์ต้องห้ามเข้าไปในตารางความน่าจะเป็นและมีความน่าจะที่จะมีการเกิดสัญลักษณ์นี้โดยถ้ากำหนดให้สัญลักษณ์ต้องห้ามมีความน่าเป็นในการเกิดเท่ากับ ε จะได้ว่าเหลือความน่าจะเป็นที่สามารถนำไปใช้กับสัญลักษณ์อื่นอีก $1-\varepsilon$ การทำเช่นนี้ทำให้ความซ้ำซ้อนของข้อมูลที่ทำการเข้ารหัสโดยวิธีการตรวจจับข้อผิดพลาดต่อเนื่องเพิ่มขึ้นเท่ากับ

$$r_{forbidden} = -\log_2(\varepsilon) \qquad \text{บิตต่อสัญลักษณ์} \qquad (2.10)$$

ซึ่งทำให้วิธีการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องมีประสิทธิภาพน้อยกว่าการเข้ารหัสเชิงเลขคณิตแบบปกติเนื่องจากในการเข้ารหัสแต่ละครั้งจะมีการเพิ่มความซ้ำซ้อนเข้าไป เอนโทรปีของข้อมูลหลังจากผ่านการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องจะมีค่าเท่ากับ

$$H_{CED} = H_{AC} + r_{forbidden} \qquad \text{บิตต่อสัญลักษณ์} \qquad (2.11)$$

เพื่อให้เข้าใจการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องได้ง่ายและถูกต้องมากยิ่งขึ้น การเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องสามารถอธิบายได้ดังนี้คือ สมมติให้มีสัญลักษณ์เพียงสองสัญลักษณ์คือสัญลักษณ์ที่ใช้ได้และสัญลักษณ์ต้องห้าม สัญลักษณ์ที่ใช้ได้จะถูกกำหนดให้มีความน่าจะเป็นเท่ากับ $(1-\varepsilon)$ และสัญลักษณ์ต้องห้ามจะถูกกำหนดให้มีความน่าจะเป็นเท่ากับ ε โดย ε จะมีค่าอยู่ระหว่าง 0 ถึง 1 ทุกครั้งที่ทำการเข้ารหัสสัญลักษณ์ที่ใช้ได้ โดยถ้าช่วงปัจจุบันของการเข้ารหัสเท่ากับ α ช่วงย่อยจะถูกแบ่งเป็นสองส่วนคือ ช่วงย่อย $(1-\varepsilon)\alpha$ ซึ่งเป็นของสัญลักษณ์ที่ใช้ได้ และช่วงย่อย $\varepsilon\alpha$ ซึ่งเป็นของสัญลักษณ์ต้องห้าม เนื่องจากสัญลักษณ์ต้องห้ามไม่เคยถูกเข้ารหัสโดยภาคส่งดังนั้นช่วงย่อยซึ่งเป็นของสัญลักษณ์ต้องห้ามจะไม่ถูกแบ่งเป็นช่วงย่อยลงไปอีก ดังนั้นหลังจากทำการเข้ารหัสผ่านไป n สัญลักษณ์ ความกว้างของช่วงที่ใช้ในการสร้างรหัสที่ใช้ได้จะถูกลดลงไปมีค่าเท่ากับ $(1-\varepsilon)^n$ กล่าวคือการลดลงของความกว้างของช่วงจะลดลงไปตามรูปที่ 2.10 ถ้าข้อผิดพลาดเกิดขึ้นในข้อมูลที่ผ่านการเข้ารหัสแล้ว การถอดรหัสโดยการใช้คำรหัสที่ไม่ถูกต้องนั้นจะมีการเลือกช่วงที่ใช้ในการถอดรหัสที่ต่างออกไปจากการเข้ารหัส ซึ่งการเปลี่ยนของช่วงที่ถูกเลือกนี้จะมีการกระจายอย่างเท่าเทียมกันบนช่วง [0,1)



รูปที่ 2.10 การลดลงของขนาดของช่วงที่ใช้ได้

ถ้ากำหนดให้ตัวแปร Y แสดงถึงจำนวนของบิตที่ใช้ก่อนที่ภาครับจะสามารถตรวจพบข้อผิดพลาดจะได้ว่า Y สามารถเขียนเป็นสมการได้ดังนี้

$$P_Y(k) = (1-\varepsilon)^{k-1}\varepsilon, \qquad k = 1,2,3,...,\infty \tag{2.12}$$

เมื่อเกิดข้อผิดพลาดขึ้น ช่วงที่ใช้ทำการถอดรหัสจะไม่ถูกจำกัดให้อยู่ในช่วงส่วนที่เป็นของสัญลักษณ์ที่ใช้ได้อีกต่อไป และการที่เกิดข้อผิดพลาดขึ้นทำให้มีความน่าจะเป็นเท่ากับ ε ที่ช่วงที่ใช้ในการถอดรหัสจะไปอยู่ในช่วงที่เป็นของสัญลักษณ์ต้องห้าม และมีความน่าจะเป็นเท่ากับ $(1-\varepsilon)$ ที่ช่วงที่ใช้ในการถอดรหัสจะไปอยู่ในช่วงที่เป็นของสัญลักษณ์ที่ใช้ได้ซึ่งจะทำให้เห็นได้ว่า

$$P[Y > n] = (1-\varepsilon)^n \tag{2.13}$$

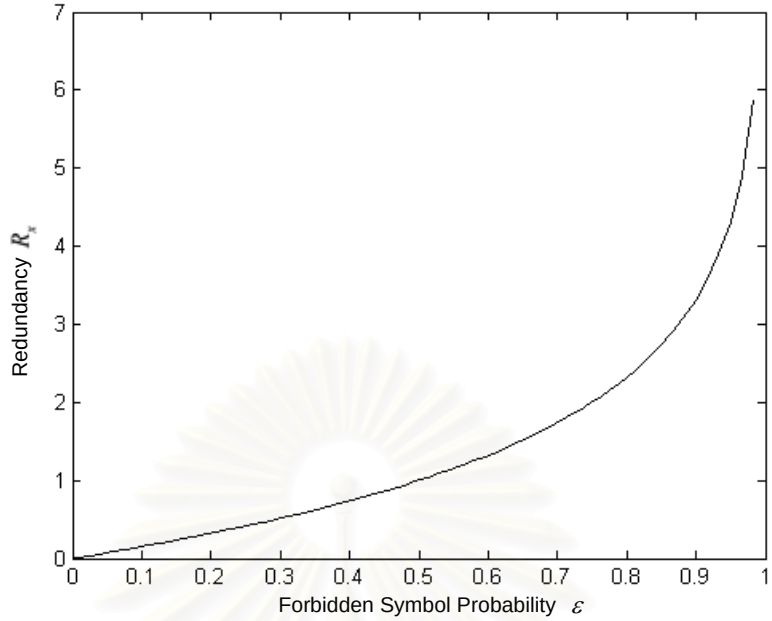
กำหนดให้ $\delta = P[Y > n]$ แล้วลอการิทึมทั้งสองข้างของสมการจะได้ว่า

$$n = \frac{\log_2(\delta)}{\log_2(1-\varepsilon)} \tag{2.14}$$

จากสมการที่ 2.10 จะได้ว่าเมื่อเกิดข้อผิดพลาดขึ้น ค่า δ จะใช้แสดงค่าความมั่นใจซึ่งสัมพันธ์กับความจริงที่ว่ามีข้อผิดพลาดเกิดขึ้นที่บิตที่ n ที่ผ่านมาสำหรับแต่ละ ε ซึ่งยังชี้ให้เห็นว่ายังคงมีความน่าจะเป็นอยู่บ้างที่ถึงแม้ว่าจะเกิดข้อผิดพลาดขึ้น รูปแบบของข้อผิดพลาดจะทำให้คำรหัสที่เกิดข้อผิดพลาดขึ้นเปลี่ยนไปเป็นคำรหัสอื่นที่ใช้ได้ซึ่งทำให้ไม่สามารถที่จะทำการตรวจจับข้อผิดพลาดได้ ซึ่งเป็นเหตุการณ์ที่มีความน่าจะเป็นในการเกิดขึ้นน้อยมาก

หลังจากที่ได้สร้างความสามารถในการตรวจจับข้อผิดพลาดขึ้นในกระบวนการเข้ารหัสเชิงเลขคณิต สิ่งที่ต้องแลกกับความสามารถในการตรวจจับข้อผิดพลาดนี้คือบิตที่เพิ่มขึ้นเนื่องจากการเพิ่มสัญลักษณ์ต้องห้ามเข้าไปในตารางความน่าจะเป็น เพื่อจะหาสมรรถภาพที่ลดลงเนื่องจากการเพิ่มสัญลักษณ์ต้องห้าม จากสมการที่ 2.1 จะได้ว่าปกติแล้วเราต้องการบิตจำนวน $-\log_2(\gamma)$ บิตในการแสดงถึงช่วงที่มีความกว้าง γ ที่อยู่บนช่วงหนึ่งหน่วย $[0,1)$ จากการเพิ่มสัญลักษณ์ต้องห้าม x เข้าไปซึ่งจะเข้าไปเป็นเจ้าของช่วงที่มีความกว้าง ε ส่วนที่เหลือจะเป็นของสัญลักษณ์ข้อมูลซึ่งจะมีความกว้างที่เหลือเท่ากับ $(1-\varepsilon)$ ดังนั้นผลลัพธ์จากการที่เพิ่มสัญลักษณ์ต้องห้ามเข้าไปจะทำให้มีความซ้ำซ้อนเนื่องจากการนี้เป็นค่าเท่ากับ

$$R_x = -\log_2(1-\varepsilon) \qquad \text{บิตต่อสัญลักษณ์} \tag{2.15}$$



รูปที่ 2.11 ความสัมพันธ์ระหว่างความซ้ำซ้อนกับความน่าจะเป็นของสัญลักษณ์ต้องห้าม

จากรูปที่ 2.11 จะเห็นว่ายิ่งเพิ่มความน่าจะเป็นของสัญลักษณ์ต้องห้ามให้มากขึ้นก็จะเป็นการเพิ่มความซ้ำซ้อนให้มากขึ้นตามไปด้วย โดยการเพิ่มความซ้ำซ้อนก็จะเป็นการเพิ่มความเร็วในการตรวจจับข้อผิดพลาดที่เกิดขึ้นด้วย จะเห็นว่าหากต้องการให้ความล้มเหลวในการตรวจจับข้อผิดพลาดมีความน่าจะเป็นที่จะเกิดขึ้นต่ำนั้นจะทำให้ขนาดของข้อมูลที่ต้องทำการส่งนั้นเพิ่มขึ้นเป็นอย่างมากเนื่องจากความซ้ำซ้อนจะเพิ่มขึ้นเป็นแบบลอการิทึม โดยสมการความสัมพันธ์ระหว่างความเร็วในการตรวจจับข้อผิดพลาดกับความซ้ำซ้อนสามารถหาได้จากสมการที่ 2.10 และสมการที่ 2.11 แทนค่าจากสมการที่ 2.11 ลงในสมการที่ 2.10 จะได้ว่า

$$n = -\frac{\log_2(\delta)}{R_x} \tag{2.16}$$

จากสมการที่ 2.12 จะเห็นได้ว่าความซ้ำซ้อนที่เกิดเนื่องจากระดับความมั่นใจหนึ่งจะแปรผกผันกับปริมาณของเวลาที่ใช้ก่อนที่จะสามารถตรวจจับข้อผิดพลาดได้การที่มีสมการนี้แสดงให้เห็นว่าเราสามารถที่จะทำการควบคุมความซ้ำซ้อนและความเร็วในการตรวจจับข้อผิดพลาดได้โดยตรง ตัวเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่อง จะมีขั้นตอนดังต่อไปนี้

1. กำหนดค่าความน่าจะเป็นของสัญลักษณ์ต้องห้าม ε อาจเลือกจากค่าความซ้ำซ้อนที่เพิ่มขึ้นหรืออาจเลือกจากจำนวนเฉลี่ยของข้อมูลก่อนที่จะตรวจพบข้อผิดพลาด ซึ่งสามารถหาได้จากสมการที่ 2.11 และสมการที่ 2.12
2. ทำการเปลี่ยนแปลงตารางความน่าจะเป็นให้เป็นไปตามลักษณะของสัญลักษณ์ต้องห้าม

3. ทำการเข้ารหัสเชิงเลขคณิตโดยใช้ตารางความน่าจะเป็นที่ได้ถูกแก้ไขแล้ว

ส่วนตัวถอดรหัสตรวจจับข้อผิดพลาดต่อเนื่องจะต้องมีการปรับค่าความน่าจะเป็นให้ตรงกับตัวเข้ารหัสด้วยทั้งยังจะมีขั้นตอนในการเปรียบเทียบสัญลักษณ์เพิ่มเข้ามา ซึ่งตัวถอดรหัสตรวจจับข้อผิดพลาดต่อเนื่องจะมีขั้นตอนดังต่อไปนี้

1. ปรับลักษณะของสัญลักษณ์ต้องห้ามให้เป็นเช่นเดียวกับฝั่งตัวเข้ารหัส
2. ทำการเปลี่ยนแปลงตารางความน่าจะเป็นให้เป็นไปตามลักษณะของสัญลักษณ์ต้องห้าม
3. ทำการตรวจสอบว่าสัญลักษณ์ที่ถอดรหัสได้มานั้นเป็นสัญลักษณ์ต้องห้ามหรือไม่ ถ้าเป็นสัญลักษณ์ต้องห้ามแสดงว่ามีข้อผิดพลาดเกิดขึ้นแล้ว

วิธีการนี้มีข้อเสียคือใช้ความซับซ้อนในการคำนวณมากขึ้นมากเนื่องจากต้องมีการปรับตารางความน่าเป็นทุกครั้งที่ทำการเข้ารหัสสัญลักษณ์แต่ละตัวอีกทั้งหากต้องการให้มีการตรวจพบข้อผิดพลาดภายในช่วงความยาวของข้อมูลที่ยอมรับได้จะต้องมีการเพิ่มความซ้ำซ้อนเข้าไปเป็นจำนวนมาก เมื่อมีข้อผิดพลาดเกิดขึ้นหลายครั้งก่อนที่จะตรวจจับได้อาจทำไม่สามารถตรวจจับข้อผิดพลาดที่เกิดขึ้นได้ แต่ข้อดีของวิธีการนี้คือความสามารถในการตรวจจับข้อผิดพลาดที่สามารถทำได้นั้นจะไม่มีข้อจำกัดในรูปแบบของข้อผิดพลาดที่จะทำการตรวจจับแต่จะมีเพียงความสัมพันธ์ระหว่างความซ้ำซ้อนกับความเร็วในการตรวจจับเท่านั้น ซึ่งในกรณีที่ต้องการความเร็วในการตรวจจับสูงก็สามารถที่จะเพิ่มความซ้ำซ้อนของข้อมูลให้มากขึ้น หรือในกรณีที่ไม่มีความจำเป็นในการตรวจจับมากหรืออยู่ในสภาวะที่ไม่มีสัญญาณรบกวนก็สามารถที่จะลดความซ้ำซ้อนลงได้ โดยการควบคุมความซ้ำซ้อนนั้นสามารถทำได้โดยตรงด้วยการเปลี่ยนความน่าจะเป็นในการเกิดของสัญลักษณ์ต้องห้าม

2.6 การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้เครื่องหมาย

จากวิธีการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องจะเห็นว่าการตรวจจับข้อผิดพลาดของวิธีการตรวจจับข้อผิดพลาดต่อเนื่องนั้นสามารถทำการตรวจจับข้อผิดพลาดได้ทุกครั้งหลังจากทำการถอดรหัสแต่ละสัญลักษณ์แต่วิธีการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องนั้นจะใช้ความซับซ้อนในการคำนวณเพิ่มขึ้น วิธีการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้เครื่องหมาย[10] จะมีความซับซ้อนในการคำนวณที่น้อยกว่าเนื่องจากไม่ทำการเข้ารหัสเพื่อทำการตรวจจับข้อผิดพลาดเมื่อทำการเข้ารหัสทุกสัญลักษณ์แต่จะมีการเว้นระยะการเข้ารหัสตรวจจับข้อผิดพลาดเป็นช่วง โดยวิธีการนี้จะมีการใส่สัญลักษณ์ที่ใช้เป็นเครื่องหมายไว้เป็นระยะๆ โดยมีความห่างของการใส่เครื่องหมายแต่ละเครื่องหมายที่แน่นอนคงที่ โดยสามารถที่จะเลือกสัญลักษณ์ใดใช้เป็นเครื่องหมายก็ได้ วิธีการนี้สามารถมองเป็นรูปแบบการเรียงข้อมูลแบบบล็อกได้ดังต่อไปนี้

รูปที่ 2.12 รูปแบบการเรียงของข้อมูล

จากรูปที่ 2.12 จะเห็นว่าการเรียงของข้อมูลจะมีการใส่เครื่องหมายเพิ่มเข้าไปในข้อมูลที่จะทำการส่งทุกบล็อกที่มี k สัญลักษณ์ดังนั้นแต่ละบล็อกจะมีขนาด $k+n$ สัญลักษณ์ การตรวจจับข้อผิดพลาดจะทำได้โดยการตกลงเครื่องหมายที่จะทำการตรวจสอบกันระหว่างภาคส่งและภาครับโดยหากเครื่องหมายที่ทำการเข้ารหัสที่ภาคส่งกับเครื่องหมายที่ถอดรหัสได้ที่ภาครับไม่ตรงกันจะทำให้ภาครับรู้ได้ว่าได้มีข้อผิดพลาดเกิดขึ้นแล้ว โดยข้อผิดพลาดที่เกิดขึ้นกับข้อมูลที่ผ่านการเข้ารหัสโดยการเข้ารหัสเชิงเลขคณิตจะเกิดการแพร่กระจายของข้อผิดพลาดทำให้ข้อมูลที่ถอดรหัสจากข้อมูลหลังบิตที่เกิดข้อผิดพลาดขึ้นจะได้รับผลจากข้อผิดพลาดนั้นด้วยทำให้สามารถที่จะตรวจจับข้อผิดพลาดจากเครื่องหมายที่ใส่เพิ่มเข้าไปได้ วิธีการเลือกเครื่องหมายสามารถทำได้หลายวิธี แต่มีวิธีที่ถูกนำเสนอไว้ 3 วิธีดังนี้

1. วิธีเลือกสัญลักษณ์ใดๆ เพื่อใช้เป็นเครื่องหมาย วิธีการนี้จะทำการเลือกสัญลักษณ์ใดๆ ที่ปรากฏอยู่ในข้อมูลที่จะทำการส่งขึ้นมาเพื่อนำมาใช้เป็นเครื่องหมาย
2. วิธีเลือกสัญลักษณ์ตัวสุดท้ายที่เกิดขึ้นเพื่อใช้เป็นเครื่องหมาย วิธีการนี้จะทำการเลือกสัญลักษณ์ตัวสุดท้ายที่เกิดขึ้นก่อนที่จะทำการใส่เครื่องหมายขึ้นมาเพื่อนำมาใช้เป็นเครื่องหมาย
3. วิธีเลือกสัญลักษณ์เฉลี่ยเพื่อใช้เป็นเครื่องหมาย วิธีการนี้จะทำการเลือกเอาสัญลักษณ์ที่เป็นสัญลักษณ์ที่มีตัวเลขประจำสัญลักษณ์น้อยกว่าหรือเท่ากับค่าเฉลี่ยของตัวเลขประจำสัญลักษณ์ภายในบล็อกที่จะทำการใส่เครื่องหมายหรือเขียนเป็นสมการได้ว่า $S^{k+1} = \sum_{i=1}^{k} S^i / k$ เมื่อ S^i เป็นตัวเลขประจำสัญลักษณ์ตัวที่ i ที่อยู่ภายในบล็อกและ S^{k+1} เป็นตัวเลขประจำสัญลักษณ์ที่ถูกเลือกเป็นเครื่องหมาย

วิธีการเลือกเครื่องหมายทั้ง 3 วิธีมีผลต่อความสามารถในการตรวจจับข้อผิดพลาดรวมทั้งความเร็วในการตรวจจับข้อผิดพลาดและความซ้ำซ้อนที่เพิ่มขึ้นอีกด้วย

2.6.1 ความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลว

ความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลว เป็นความน่าจะเป็นที่เมื่อเกิดข้อผิดพลาดขึ้นแล้วแต่การตรวจจับข้อผิดพลาดไม่สามารถที่จะตรวจจับข้อผิดพลาดนั้นได้ โดยความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลวของแต่ละวิธีการที่ใช้เลือกเครื่องหมายจะมีค่าไม่เท่ากัน โดยความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลวของแต่ละวิธีการเป็นดังต่อไปนี้

1. วิธีเลือกสัญลักษณ์ใดๆ เพื่อใช้เป็นเครื่องหมาย ความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลวของวิธีการนี้จะมีค่าเท่ากับความน่าจะเป็นของสัญลักษณ์ที่ถูกเลือกเป็นเครื่องหมาย หรือสามารถเขียนเป็นสมการได้ดังนี้

$$P_{failure} = P(s_n) \tag{2.17}$$

เมื่อ s_n เป็นสัญลักษณ์ที่ถูกเลือกเป็นเครื่องหมาย

2. วิธีเลือกสัญลักษณ์ตัวสุดท้ายที่เกิดขึ้นเพื่อใช้เป็นเครื่องหมาย ความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลวของวิธีการนี้จะมีค่าเท่ากับค่าเฉลี่ยของความน่าจะเป็นที่จะตรวจจับข้อผิดพลาดล้มเหลวของสัญลักษณ์ทั้งหมด โดยจะสามารถเขียนเป็นสมการได้ดังนี้

$$P_{failure} = \sum_{n=1}^{l} P(s_n)[P^2(s_n)] \tag{2.18}$$

เมื่อ l เป็นจำนวนของสัญลักษณ์ทั้งหมด

3. วิธีเลือกสัญลักษณ์เฉลี่ยเพื่อใช้เป็นเครื่องหมาย ความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลวของวิธีการนี้จะมีค่าเท่ากับความน่าจะเป็นที่เครื่องหมายที่ถูกเลือกใหม่โดยข้อมูลที่ถูกถอดรหัสออกมาผิดยังเป็นสัญลักษณ์เดียวกับที่ถูกเลือกเป็นเครื่องหมายโดยข้อมูลที่ถูกต้อง ซึ่งจะหาความน่าจะเป็นได้ดังต่อไปนี้

$$P_{failure} = \sum_{S^{k+1}=1}^{S_M} P(S^{k+1})P(S^{k+1}) \tag{2.19}$$

เมื่อ S_M เป็นตัวเลขสูงสุดของตัวเลขประจำสัญลักษณ์

ความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลวจะลดลงได้ถ้าทำการตรวจจับข้อผิดพลาดหลายๆ ครั้ง โดยความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลวจะลดลงจาก $p_{failure}$ ไปเป็น $p_{failure}^m$ เมื่อ m เป็นจำนวนครั้งที่ทำการตรวจจับ ดังนั้นความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลวจะมีแนวโน้มในการเข้าใกล้ศูนย์มากยิ่งขึ้นเมื่อทำการตรวจจับมากครั้งขึ้นซึ่งจะทำให้สามารถตรวจจับข้อผิดพลาดได้อย่างแน่นอนและไม่คำนึงถึงรูปแบบของข้อผิดพลาด แต่ก็อาจมีข้อผิดพลาดบางรูปแบบที่สามารถทำให้การประสานเวลากลับมาถูกต้องได้หลังจากสูญเสียการประสานเวลาขณะที่เกิดข้อผิดพลาดครั้งแรก ในกรณีนี้การตรวจจับข้อผิดพลาดโดยใช้เครื่องหมายอาจเกิดความล้มเหลวในการตรวจจับข้อผิดพลาด แต่ข้อผิดพลาดชนิดนี้จะพบได้ยากเมื่ออัตราข้อผิดพลาดบิตต่ำ

2.6.2 ขนาดของข้อมูลที่เพิ่มขึ้น

ในส่วนนี้จะอธิบายถึงขนาดของข้อมูลที่เพิ่มขึ้นเนื่องจากการเพิ่มเครื่องหมายเข้าไปในข้อมูล โดยวิธีการเลือกเครื่องหมายแต่ละวิธีก็จะส่งผลต่อการเพิ่มขนาดของข้อมูลที่ไม่เท่ากัน โดยการเพิ่มขนาดของข้อมูลอันเกิดจากวิธีการเลือกเครื่องหมายแต่ละวิธีเป็นดังต่อไปนี้

กำหนดให้ $p(s_i)$ เป็นความน่าจะเป็นของสัญลักษณ์ s_i ซึ่งจะเท่ากับ $N(s_i)/L$ เมื่อ $N(s_i)$ เป็นจำนวนครั้งทั้งหมดที่สัญลักษณ์ s_i เกิดขึ้นในข้อมูล และ L เป็นขนาดของข้อมูล จากการเพิ่มเครื่องหมายเข้าไปในข้อมูลทุกครั้งเมื่อมีข้อมูลผ่านไป k สัญลักษณ์ จะได้ว่าในกรณีที่เลือกสัญลักษณ์ใดๆเป็นเครื่องหมาย ความน่าจะเป็นใหม่สำหรับสัญลักษณ์ที่ไม่ถูกใช้เป็นเครื่องหมายเปลี่ยนไปเป็น

$$p'(s_i) = \frac{N(s_i)}{L\left(1+\frac{1}{k}\right)} = \frac{p(s_i)}{1+\frac{1}{k}} \tag{2.20}$$

และสำหรับสัญลักษณ์ที่ถูกใช้เป็นเครื่องหมาย a_n จะเปลี่ยนไปเป็น

$$p'(s_n) = \frac{N(s_n)+\frac{L}{k}}{L\left(1+\frac{1}{k}\right)} = \frac{p(s_n)+\frac{1}{k}}{1+\frac{1}{k}} \tag{2.21}$$

ความน่าจะเป็นที่เปลี่ยนไปทำให้เกิดความเปลี่ยนแปลงกับเอนโทรปีด้วยซึ่งทำให้ได้เอนโทรปีใหม่ $H'(p)$ ที่สามารถหาได้จาก $p'(s_i)$ และ $p'(s_n)$ จากสมการที่ 2.16 และสมการที่ 2.17 เพื่อที่จะหาขนาดไปเพิ่มขึ้นเราจะหาขนาดที่เปลี่ยนไปโดยผ่านอัตราการบีบอัด ซึ่งมีค่าเท่ากับอสมการ $R \le \log_2 l/H(p)$ หลังจากที่ที่มีการเพิ่มเครื่องหมายเข้าไปแล้วอัตราการบีบอัดจะมีค่าเท่ากับ $R' \le \log_2 l/H'(p)$ ซึ่งสามารถจะเท่ากับสมการ $R = \log_2 l/H(p)$ ได้เมื่อใช้วิธีการเข้ารหัสที่มีประสิทธิภาพ ซึ่งจะได้ว่าขนาดของข้อมูลที่ผ่านการเข้ารหัสแล้วจะเท่ากับ $L/R = LH(p)/\log_2 l$ หลังจากเพิ่มเครื่องหมายเข้าไปในข้อมูลขนาดของข้อมูลที่ผ่านการเข้ารหัสจะกลายเป็น

$$\frac{L\left(1+\frac{1}{k}\right)}{R'} = \frac{L\left(1+\frac{1}{k}\right)H'(p)}{\log_2 l} \tag{2.22}$$

ดังนั้นจะได้ว่าขนาดของข้อมูลที่เพิ่มขึ้นจะเท่ากับ

$$\frac{L\left(1+\frac{1}{k}\right)}{R'}-\frac{L}{R}=\frac{L}{\log_2 l}\left[\left(1+\frac{1}{k}\right)H'(p)-H(p)\right] \tag{2.23}$$

หลังจากทำการแทนค่า $H(p)$ และ $H'(p)$ ปริมาณข้อมูลที่เปลี่ยนแปลงไปจะสามารถหาได้จากสมการนี้ และหลังจากหารค่าที่หาได้ด้วย $H(p)$ เราก็จะได้เปอร์เซ็นต์ของขนาดที่เพิ่มขึ้น Δf เท่ากับ

$$\Delta f=\frac{1}{H(p)}\left[\left(1+\frac{1}{k}\right)\log_2\left(1+\frac{1}{k}\right)+p(s_n)\log_2 p(s_n)-\left(p(s_n)+\frac{1}{k}\right)\log_2\left(p(s_n)+\frac{1}{k}\right)\right] \tag{2.24}$$

สำหรับวิธีเลือกสัญลักษณ์ตัวสุดท้ายที่เกิดขึ้นเพื่อใช้เป็นเครื่องหมาย จำนวนการเกิดสัญลักษณ์ a_i หลังจากที่ทำการเพิ่มเครื่องหมายแล้วจะมีค่าเท่ากับ

$$N'(s_i)=N(s_i)+(L/k)p(s_i)=N(s_i)\left(1+(1/k)\right) \tag{2.25}$$

เมื่อ $i=1,2,3,...,l$ ดังนั้นจะได้ความน่าจะเป็นหลังจากที่เพิ่มเครื่องหมายแล้วเท่ากับ

$$p'(s_i)=\frac{N(s_i)\left(1+\frac{1}{k}\right)}{L\left(1+\frac{1}{k}\right)}=p(s_i) \tag{2.26}$$

ซึ่งหมายความว่าเอนโทรปีจะมีค่าเท่าเดิมเมื่อแทนค่าในสมการที่ 2.19 จะได้ว่า

$$\frac{L\left(1+\frac{1}{k}\right)}{R'}-\frac{L}{R}=\frac{L}{\log_2 l}\left[\left(\frac{1}{k}\right)H(p)\right] \tag{2.27}$$

และจะได้ว่าเปอร์เซ็นต์ของขนาดที่เพิ่มขึ้น Δf เท่ากับ

$$\Delta f=1/k \tag{2.28}$$

เมื่อ k เป็นขนาดของบล็อก

สำหรับวิธีเลือกสัญลักษณ์เฉลี่ยเพื่อใช้เป็นเครื่องหมาย จะได้ว่าจำนวนการเกิดใหม่จะเท่ากับ

$$N'(s_i)=N'(s_i)+(L/k)p_{mi} \tag{2.29}$$

เมื่อ $p_{mi} = p(S_{\text{marker}})$ เมื่อ S_{marker} เป็นสัญลักษณ์ที่ถูกใช้เป็นเครื่องหมาย จะได้ความน่าจะเป็นใหม่มีค่าเท่ากับ

$$p'(s_i) = \frac{N(s_i) + \frac{L}{k} p_{mi}}{1 + \frac{1}{k}} = \frac{p(s_i) + \frac{1}{k} p_{mi}}{1 + \frac{1}{k}} \tag{2.30}$$

ซึ่งทำให้ได้เอนโทรปีหลังจากที่มีการเพิ่มเครื่องหมายแล้วเท่ากับ

$$H'(p) = -\sum_{i=1}^{l} \frac{p(s_i) + \frac{L}{k} p_{mi}}{1 + \frac{1}{k}} \log_2 \frac{p(s_i) + \frac{L}{k} p_{mi}}{1 + \frac{1}{k}} \tag{2.31}$$

จากนั้นนำค่าที่ได้ไปหาว่าเปอร์เซ็นต์ของขนาดที่เพิ่มขึ้น Δf จะได้เท่ากับ

$$\Delta f = \frac{1}{H(p)} \left[\left(1 + \frac{1}{k}\right) \log_2 \left(1 + \frac{1}{k}\right) + \sum_{i=1}^{l} \left[p(s_i) \log_2 p(s_i) - \left(p(s_i) + \frac{p_{mi}}{k} \right) \log_2 \left(p(s_i) + \frac{p_{mi}}{k} \right) \right] \right] \tag{2.32}$$

บทที่ 3

กรรมวิธีที่นำเสนอ

ในบทนี้จะอธิบายถึงวิธีการที่นำเสนอและใช้ในการปรับปรุงวิธีการเข้ารหัสตรวจจับข้อผิดพลาดด้วยการเข้ารหัสเชิงเลขคณิต โดยวิธีการที่นำเสนอนี้จะสามารถนำไปใช้กับการเข้ารหัสเชิงเลขคณิตแบบปรับตัวได้ ซึ่งจะมีการเปลี่ยนแปลงของตารางความน่าจะเป็นอยู่ทุกการเข้ารหัสแต่ละสัญลักษณ์ วิธีที่นำเสนอประกอบด้วยวิธีการดังต่อไปนี้

3.1 การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุด

จากวิธีการเข้ารหัสตรวจจับข้อผิดพลาดทั้งสองวิธีที่ได้ถูกนำเสนอไปแล้วในบทที่ 2 คือ การเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องและการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์เครื่องหมายจะเห็นว่าวิธีการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องนั้นมีการกำหนดค่าของความน่าจะเป็นให้กับสัญลักษณ์ต้องห้ามและจะทำการกำหนดให้ค่าของความน่าเป็นนั้นคงที่อยู่ตลอดเวลาเพื่อให้สามารถที่จะทำการเข้ารหัสและถอดรหัสข้อมูลออกมาได้อย่างถูกต้อง การทำเช่นนี้ทำให้สามารถทำการตรวจจับข้อผิดพลาดได้โดยมีความน่าจะเป็นที่จะสามารถตรวจจับข้อผิดพลาดที่ได้เกิดขึ้นแล้วเพิ่มขึ้นด้วยอัตราที่เท่ากันในทุกการตรวจจับจะเห็นได้ว่าการตรวจจับแต่ละครั้งนั้นจะไม่แปรตามความสำคัญของข้อมูลที่จะทำการป้องกันด้วยการเข้ารหัสตรวจจับข้อผิดพลาด วิธีการตรวจจับข้อผิดพลาดต่อเนื่องนั้นอาจมีความซับซ้อนในการคำนวณมากเมื่อนำไปใช้กับการเข้ารหัสเชิงเลขคณิตที่ใช้แบบจำลองแบบปรับตัวได้เนื่องจากจำเป็นต้องมีการปรับแบบจำลองให้เป็นไปตามค่าความจะเป็นของสัญลักษณ์ต้องห้ามที่ถูกตกลงกันไว้ระหว่างตัวเข้ารหัสและตัวถอดรหัส วิธีการเข้ารหัสโดยใช้เครื่องหมายนั้นจะใช้ความซับซ้อนน้อยกว่าเมื่อนำไปใช้กับการเข้ารหัสเชิงเลขคณิตที่ใช้แบบจำลองแบบปรับตัวได้เนื่องจากจะไม่มีการปรับแบบจำลองเกิดขึ้น แต่การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้เครื่องหมายจะมีข้อจำกัดคือความซ้ำซ้อนต่ำที่สุดที่สามารถจะใช้เป็นเครื่องหมายในการตรวจจับข้อผิดพลาดได้นั้นจะเท่ากับ ความซ้ำซ้อนของสัญลักษณ์ที่มีความน่าจะเป็นในการเกิดสูงสุด ซึ่งหากใส่เครื่องหมายลงไปในข้อมูลเป็นจำนวนมากก็จะมีผลทำให้มีความซ้ำซ้อนมากจนเกินไปดังนั้นยิ่งความน่าเป็นในการเกิดของสัญลักษณ์ที่ถูกเลือกเป็นเครื่องหมายมีน้อยเพียงไรระยะห่างที่จะทำการใส่เครื่องหมายแต่ละตัวก็จะต้องมากขึ้นเท่านั้นเพื่อไม่ให้มีความซ้ำซ้อนมากจนเกินไป

วิธีการแก้ไขทางหนึ่งคือการเลือกใช้สัญลักษณ์ที่มีความน่าจะเป็นในการเกิดสูงที่สุดใช้เป็นเครื่องหมายเสมอ เพื่อทำให้ความซ้ำซ้อนโดยเฉลี่ยที่เพิ่มเข้าไปในข้อมูลที่ผ่านการ

เข้ารหัสแล้วน้อยลงเป็นผลให้สามารถที่จะทำการใส่เครื่องหมายเข้าไปในข้อมูลเพื่อทำการตรวจจับข้อผิดพลาดได้บ่อยครั้งมากขึ้น

3.2 ขั้นตอนการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุด

วิธีการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดจะใช้วิธีการเดียวกับการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้เครื่องหมายโดยจะทำการแบ่งข้อมูลออกเป็นบล็อกที่มีขนาดเท่ากัน k หลังจากนั้นจะใส่เครื่องหมายซึ่งเลือกออกมาจากสัญลักษณ์ที่เกิดขึ้นในข้อมูลในกรณีนี้จะเลือกเอาสัญลักษณ์ที่มีความน่าจะเป็นในการเกิดสูงสุดเพื่อใช้เป็นเครื่องหมายจากนั้นทำการเข้ารหัสเครื่องหมาย n ครั้ง และจะทำซ้ำกระบวนการเข้ารหัสแบบเดียวกันนี้จนถึงสัญลักษณ์ตัวสุดท้ายจากนั้นจะเข้ารหัสสิ้นสุดแฟ้มเพื่อบอกให้ตัวถอดรหัสที่ภาครับหยุดทำการถอดรหัส

ขั้นตอนของการเข้ารหัสและถอดรหัสจะสรุปเป็นขั้นตอนได้ดังนี้

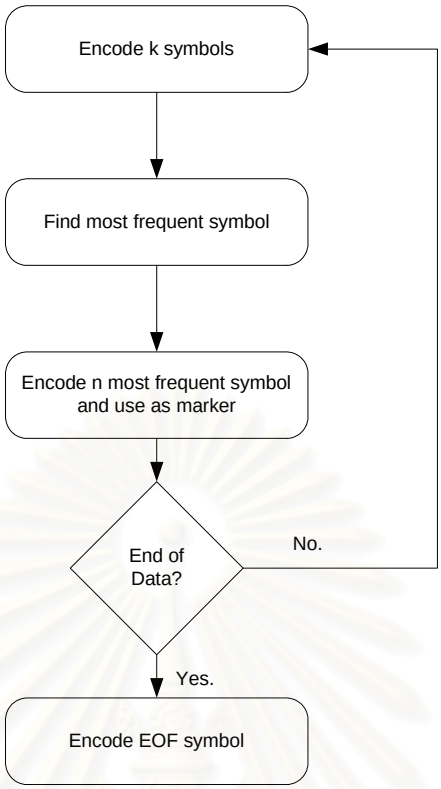
ตัวเข้ารหัส

1. เข้ารหัสสัญลักษณ์ k สัญลักษณ์โดยใช้การเข้ารหัสเชิงเลขคณิต
2. หาสัญลักษณ์ที่มีความถี่ในการเกิดสะสมมากที่สุด
3. เข้ารหัสสัญลักษณ์ที่มีความถี่ในการเกิดสะสมมากที่สุด n ครั้ง
4. ทำซ้ำข้อ 1-3 จนกระทั่งหมดข้อมูลที่ต้องการเข้ารหัส
5. เข้ารหัสสัญลักษณ์บอกจุดจบแฟ้มเพื่อบอกให้ตัวถอดรหัสหยุดทำการถอดรหัส

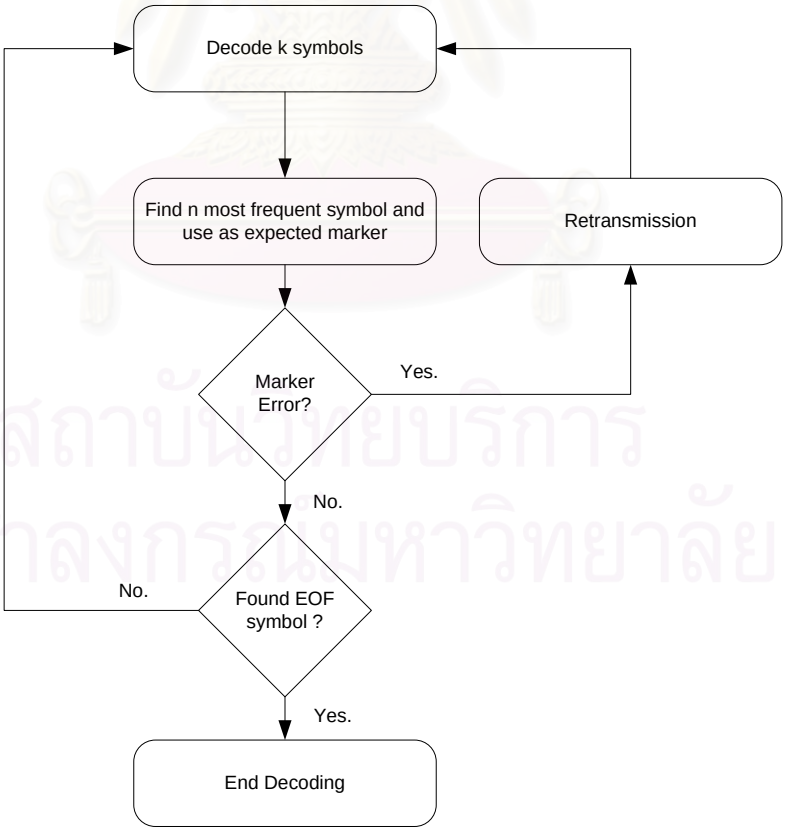
ตัวถอดรหัส

1. ถอดรหัสข้อมูลให้ได้สัญลักษณ์ k สัญลักษณ์
2. หาสัญลักษณ์ที่มีความถี่ในการเกิดสะสมมากที่สุดที่ฝั่งตัวถอดรหัส
3. ถอดรหัส n ข้อมูลตัวถัดไปซึ่งต้องเป็นสัญลักษณ์ที่มีความถี่ในการเกิดสะสมมากที่สุดจากข้อมูลที่ถูกเข้ารหัส
4. เปรียบเทียบสัญลักษณ์ที่มีความถี่ในการเกิดสะสมมากที่สุดจากข้อ 2 และข้อ 3 หากไม่ตรงกันแสดงว่ามีข้อผิดพลาดเกิดขึ้นแล้วจำเป็นต้องมีการส่งข้อมูลมาอีกครั้งเพื่อแก้ไขข้อผิดพลาด
5. ทำซ้ำข้อ 1-4 จนกว่าจะพบสัญลักษณ์บอกจุดจบแฟ้ม

จากขั้นตอนการเข้ารหัสและถอดรหัสที่กล่าวมาแล้วนั้นสามารถเขียนเป็นแผนผังการทำงานของตัวเข้ารหัสและตัวถอดรหัสได้ดังนี้



รูปที่ 3.1 แผนผังตัวเข้ารหัส



รูปที่ 3.2 แผนผังตัวถอดรหัส

3.3 ความสามารถในการตรวจจับข้อผิดพลาด

การเข้าตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดจะมีความสามารถในการตรวจจับข้อผิดพลาดเช่นเดียวกับการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องและการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้เครื่องหมายเนื่องจากการเข้ารหัสทั้ง 3 แบบจะเป็นการเข้าไปเปลี่ยนแปลงตารางความน่าจะเป็นและใช้คุณสมบัติการแพร่กระจายของข้อผิดพลาดของการเข้ารหัสเชิงเลขคณิตในการช่วยให้การตรวจจับแต่ละครั้งเป็นผลต่อเนื่องกันและทำให้ความน่าจะเป็นในการตรวจจับข้อผิดพลาดมีแนวโน้มที่จะสามารถทำการตรวจจับข้อผิดพลาดได้อย่างแน่นอน เนื่องจากการตรวจจับแต่ละครั้งหลังจากที่เกิดข้อผิดพลาดขึ้นจะมีความน่าจะเป็นในการตรวจจับที่สูงขึ้นเนื่องจากช่วงที่เป็นช่วงที่สามารถใช้งานได้จะลดลงเป็นอัตราส่วนตามค่าความน่าจะเป็นของสัญลักษณ์ที่ถูกเลือกใช้เป็นเครื่องหมาย

การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดจะมีการเข้ารหัสเครื่องหมายในแต่ละบล็อกเท่ากับ n เครื่องหมาย ดังนั้นเมื่อทำการเข้ารหัสผ่านไปแต่ละบล็อกความน่าจะเป็นที่จะสามารถทำการตรวจจับข้อผิดพลาดได้สำเร็จจะเท่ากับ

$$p_{detected} = 1 - p_{failure} \tag{3.1}$$

เมื่อ $p_{failure}$ เป็นความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลวที่จะเกิดขึ้นของแต่ละบล็อกซึ่งจะสามารถหาได้จาก

$$p_{failure} = p_{marker}^{n} \tag{3.2}$$

ดังนั้นหากทำการตรวจจับติดต่อกันหลังจากที่ข้อผิดพลาดเกิดขึ้นแล้ว m บล็อก จะได้ว่าการตรวจจับข้อผิดพลาดได้ทำการตรวจสอบผ่านไปแล้ว $m \times n$ ครั้ง ดังนั้นความน่าจะเป็นของการตรวจจับข้อผิดพลาดล้มเหลว $p_{failure}$ จะมีค่าลดลง ซึ่งจะสามารถเขียนเป็นสมการได้ว่า

$$p_{failure} = p_{marker}^{m \times n} \tag{3.3}$$

จะเห็นได้ว่าเมื่อทำการถอดรหัสผ่านไปความน่าจะเป็นที่จะสามารถตรวจจับข้อผิดพลาดได้จะเพิ่มขึ้น เนื่องจากจะมีการถอดรหัสเครื่องหมายเพื่อนำมาทำการตรวจสอบมากครั้งขึ้น การเพิ่มความน่าจะเป็นของการตรวจจับข้อผิดพลาดสามารถทำได้โดยการเพิ่มปริมาณของเครื่องหมายที่มีการใส่เข้าไปในแต่ละบล็อกหรือ ทำการลดขนาดของบล็อกแต่ละบล็อกเพื่อให้มีการตรวจจับข้อผิดพลาดเกิดขึ้นบ่อยครั้งยิ่งขึ้น วิธีการทั้งสองวิธีจะทำให้ความซ้ำซ้อนโดยรวมของระบบเพิ่มมากขึ้น

ในกรณีของสัญลักษณ์สิ้นสุดแฟ้มเพื่อให้มีความแม่นยำในการตรวจจับข้อผิดพลาดมีมากยิ่งขึ้นเนื่องจากหากตัวถอดรหัสเชิงเลขคณิตโดยปกติเมื่อพบสัญลักษณ์สิ้นสุดแฟ้มตัวถอดรหัสก็จะการทำงานทั้งหมดดังนั้นการใส่เครื่องหมายตามไปหลังจากสัญลักษณ์สิ้นสุดแฟ้มจะช่วยทำให้มีความแม่นยำในการตรวจจับข้อผิดพลาดมากขึ้น

3.4 ขนาดของข้อมูลที่เพิ่มขึ้น

ในส่วนนี้จะเป็นการวิเคราะห์วิธีการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดด้วยทฤษฎี สมมติให้สัญลักษณ์ทั้งหมดที่เกิดขึ้นในข้อมูลที่จะทำการส่งเป็น $\{s_i : i = 1,2,3,...,l\}$ ซึ่งมีความน่าจะเป็นในการเกิดของแต่ละสัญลักษณ์เป็น $\{p(s_i) : i = 1,2,3,...,l\}$ ที่ได้จากการนำจำนวนครั้งในการเกิดของสัญลักษณ์ s_i $N(s_i)$ หารด้วยจำนวนของสัญลักษณ์ทั้งหมดที่เกิดขึ้นในข้อมูล L จากที่วิธีการเข้ารหัสตรวจจับข้อผิดพลาดได้เพิ่มเครื่องหมาย n เครื่องหมายเข้าไปตามบล็อกที่มีสัญลักษณ์อยู่ k สัญลักษณ์ในแต่ละบล็อก ดังนั้น ความซ้ำซ้อนสำหรับเครื่องหมาย M แต่ละตัวจะเท่ากับ

$$r_M = \log_2(p_{marker}) \tag{3.4}$$

ขนาดของข้อมูลที่เพิ่มขึ้นจะสามารถหาได้จากความน่าจะเป็นของการเกิดของสัญลักษณ์ซึ่งจากวิธีการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้เครื่องหมายแบบแปรได้ สำหรับสัญลักษณ์ที่ไม่ได้ถูกใช้เป็นสัญลักษณ์ความน่าจะเป็นหลังจากที่มีการเพิ่มเครื่องหมายเข้าไปในข้อมูลแล้วจะมีค่าเท่ากับ

$$p'(s_i) = \frac{N(s_i)}{L\left(1+\frac{n}{k}\right)} = \frac{p(s_i)}{1+\frac{n}{k}} \tag{3.5}$$

สำหรับสัญลักษณ์ที่ถูกใช้เป็นสัญลักษณ์ความน่าจะเป็นหลังจากที่มีการเพิ่มเครื่องหมายเข้าไปในข้อมูลแล้วจะมีค่าเท่ากับ

$$p'(s_j) = \frac{N(s_j)+\frac{nL}{k}}{L\left(1+\frac{n}{k}\right)} = \frac{p(s_j)+\frac{n}{k}}{1+\frac{n}{k}} \tag{3.6}$$

เมื่อได้สมการที่ 3.5 และสมการที่ 3.6 มาทำให้ตอนนี้สามารถที่จะหาเปอร์เซ็นต์ขนาดของข้อมูลที่เพิ่มขึ้นได้โดยใช้วิธีการเดียวกับที่อธิบายไว้ในหัวข้อ 2.6.2 ซึ่งจะได้ผลลัพธ์เป็นเปอร์เซ็นต์ขนาดของข้อมูลที่เพิ่มขึ้น Δf ได้ดังนี้

$$\Delta_f = \frac{1}{H(p)}\left[\left(1+\frac{n}{k}\right)\log\left(1+\frac{n}{k}\right)+p(s_j)\log p(s_j)-\left(p(s_j)+\frac{n}{k}\right)\log\left(p(s_j)+\frac{n}{k}\right)\right] \quad (3.7)$$

3.5 ความซับซ้อนในการคำนวณที่เพิ่มขึ้น

จากวิธีการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดจะได้ว่าการเข้ารหัสนี้จะมีการเพิ่มความซับซ้อนในการคำนวณเท่ากับการคูณ $\frac{2an}{k}$ ครั้งและการบวก $\frac{4an}{k}$ ครั้งอันเนื่องมาจากมีการเข้ารหัสสัญลักษณ์ที่ใช้เป็นเครื่องหมายเพิ่มเข้าไป n เครื่องหมายเพิ่มเข้าไปทุกๆ บล็อกที่มี k สัญลักษณ์ เมื่อ a คือจำนวนสัญลักษณ์ทั้งหมดที่ทำการเข้ารหัส ส่วนความซับซ้อนในการคำนวณสำหรับการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องจะมีการเพิ่มความซับซ้อนในการคำนวณเท่ากับการคูณ a ครั้งและการบวก $2a$ ครั้ง ซึ่งจะเห็นได้ว่าการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดจะมีความซับซ้อนในการคำนวณที่เพิ่มขึ้นน้อยกว่าการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องเมื่อมีการใช้ปริมาณของเครื่องหมายไม่มากจนเกินไป

บทที่ 4

ผลการทดสอบ

ในบทนี้จะเป็นการทดสอบวิธีการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดที่ได้นำเสนอไว้ในบทที่ 3 โดยจะมีการนำผลที่ได้จากการทดสอบไปเปรียบเทียบกับผลที่ได้จากการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องที่ได้เคยศึกษาไว้ในบทที่ 2

4.1 การกำหนดพารามิเตอร์ในการเข้ารหัส

การทดสอบการเข้ารหัสทั้งหมดในบทนี้จะใช้พารามิเตอร์ทั้งหมดร่วมกันดังต่อไปนี้

1. การเข้ารหัสเชิงเลขคณิตที่ใช้จะใช้แบบจำลองแบบปรับตัวได้แบบจำลองแบบปรับตัวได้นี้จะตั้งค่าเริ่มต้นไว้ที่ 1 สำหรับทุกสัญลักษณ์
2. แบบจำลองที่ใช้ในการเข้ารหัสเชิงเลขคณิตจะประกอบด้วยสัญลักษณ์ 256 สัญลักษณ์รวมกับสัญลักษณ์สิ้นสุดแฟ้มอีก 1 สัญลักษณ์ รวมมีสัญลักษณ์ทั้งหมด 257 สัญลักษณ์
3. แบบจำลองที่ใช้จะเป็นแบบจำลองที่มีลักษณะการจำลองข้อมูลเป็นแบบมาร์คอฟเชนลำดับที่ 0

4.2 ข้อมูลที่ใช้ในการทดสอบ

ข้อมูลที่ใช้ในการทดสอบนี้จะเป็นข้อมูลที่ใช้ในการทดสอบการบีบอัดข้อมูล ข้อมูลชุดนี้มีชื่อเรียกว่า Calgary Corpus ข้อมูลชุดนี้สามารถหาได้จาก [11] ข้อมูลชุดนี้ประกอบไปด้วยแฟ้มข้อมูลทั้งสิ้น 18 แฟ้มข้อมูลซึ่งประกอบด้วยแฟ้ม bib book1 book2 geo news obj1 obj2 paper1 paper2 paper3 paper4 paper5 paper6 pic progc progl progp และ trans ข้อมูลทั้ง 18 แฟ้มสามารถแบ่งได้เป็นประเภทต่างกัน 7 ประเภทดังนี้

1. ภาษาอังกฤษที่มีการเขียนปกติซึ่งประเภทนี้จะประกอบด้วย book1 book2 paper1 paper2 paper3 paper4 paper5 และ paper6
2. ภาษาอังกฤษที่มีการเขียนไม่ปกติ ประกอบด้วย bib และ news
3. ภาษาคอมพิวเตอร์ ประกอบด้วย progc progl และ progp
4. บันทึกการใช้คำสั่งจากเครื่องคอมพิวเตอร์ปลายทาง อยู่ในแฟ้ม trans
5. แฟ้มข้อมูลที่เป็นโปรแกรมคอมพิวเตอร์ ประกอบด้วย obj1 และ obj2
6. ข้อมูลทางภูมิศาสตร์ อยู่ในแฟ้ม geo
7. ข้อมูลภาพขาวดำ อยู่ในแฟ้ม pic

ข้อมูลชุด Calgary Corpus นี้แต่จะแฟ้มจะมีขนาดของแต่ละแฟ้มไม่เท่ากัน อีกทั้งการกระจายของสัญลักษณ์ จำนวนของสัญลักษณ์ที่ใช้จริงในแต่ละแฟ้มไม่เท่ากัน อันเป็นผลทำให้เอนโทรปีของแต่ละแฟ้มไม่เท่ากัน ข้อมูลขนาดและเอนโทรปีจะแสดงไว้ในตารางที่ 4.1

ตารางที่ 4.1 ข้อมูลของแฟ้มข้อมูลทดสอบ

แฟ้ม	ขนาด(ไบต์)	เอนโทรปี(บิตต่อสัญลักษณ์)
bib	111,261	5.198
book1	768,771	4.525
book2	610,856	4.790
geo	102,400	5.643
news	377,109	5.187
obj1	21,504	5.945
obj2	246,814	6.257
paper1	53,161	4.980
paper2	82,199	4.599
paper3	46,526	4.662
paper4	13,286	4.697
paper5	11,954	4.933
paper6	38,105	5.007
pic	513,216	1.209
progc	39,611	5.196
progl	71,646	4.767
progp	49,379	4.866
trans	93,695	5.530

4.3 วิธีการทดสอบ

วิธีทำการทดสอบการเข้ารหัสจะมีรายละเอียดขั้นตอนการทดสอบดังต่อไปนี้

1. ทำการทดสอบข้อมูลทั้ง 18 แฟ้มกับการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุด โดยมีการเปลี่ยนแปลงขนาดของบล็อก k และจำนวนของเครื่องหมายที่เพิ่มเข้าไปในแต่ละบล็อก n เพื่อดูแนวโน้มของอัตราส่วนการบีบอัดที่พารามิเตอร์ต่างๆ กัน

2. ทดสอบการตรวจจับข้อผิดพลาดที่อัตราข้อผิดพลาดบิตต่างๆ กัน โดยข้อผิดพลาดที่ใช้จะเป็นข้อผิดพลาดแบบสุ่มและเปรียบเทียบกับการตรวจจับข้อผิดพลาดต่อเนื่องที่มีขนาดหลังการเข้ารหัสที่เท่ากันเพื่อเปรียบความสามารถในการตรวจจับโดยจะดูจากความเร็วในการตรวจจับข้อผิดพลาดซึ่งดูจำนวนบิตที่ผ่านการการถอดรหัสก่อนที่ข้อผิดพลาดจะถูกตรวจจับได้หลังจากที่เกิดข้อผิดพลาดแรกขึ้น

3. ทดสอบความซับซ้อนในการคำนวณจากการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดกับการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่อง

4.4 ผลการทดสอบ

ในส่วนนี้จะเป็นผลของการทดสอบที่ได้จากการทดสอบทั้ง 3 การทดสอบโดยมีผลการทดสอบดังต่อไปนี้

4.4.1 ผลการทดสอบเข้ารหัส

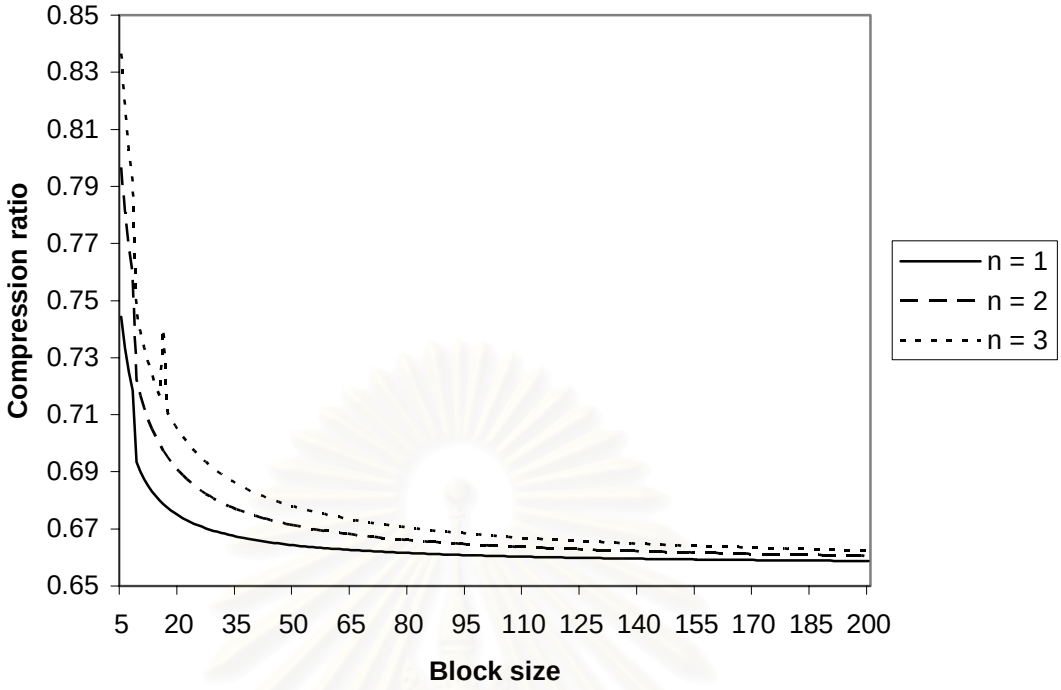
ในการทดสอบนี้จะทำการเข้ารหัสโดยใช้วิธีการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดเพื่อดูผลลัพธ์ที่ได้จากการเข้ารหัสเพื่อเทียบกับผลวิเคราะห์ที่ได้จากทฤษฎี การทดสอบจะทำการเปลี่ยนแปลงขนาดของบล็อกตั้งแต่ 5 ถึง 200 และเปลี่ยนจำนวนเครื่องหมายที่จะใส่เพิ่มเข้าไปในแต่ละบล็อก n ตั้งแต่ 1 ถึง 3 ผลการทดสอบที่ได้จะมีค่าอยู่ในรูปของอัตราส่วนการบีบอัดซึ่งเป็นไปตามสมการต่อไปนี้

$$Compression\ Ratio = \frac{L'}{L} \tag{4.1}$$

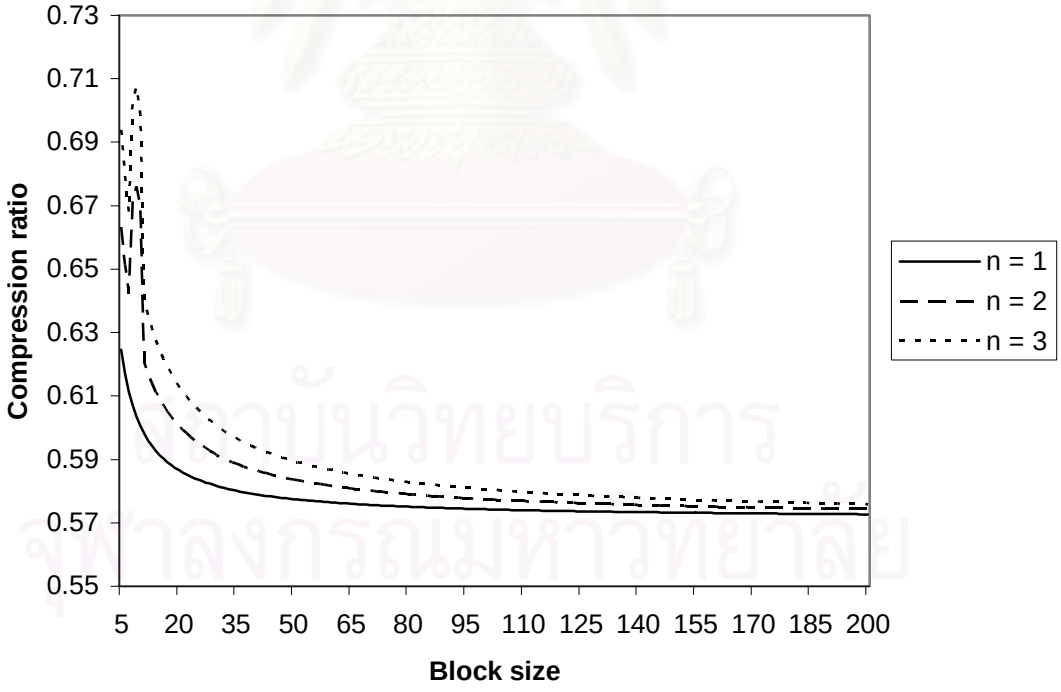
เมื่อ L' เป็นขนาดของข้อมูลที่ผ่านการเข้ารหัสแล้ว

L เป็นขนาดของข้อมูลก่อนผ่านการเข้ารหัส

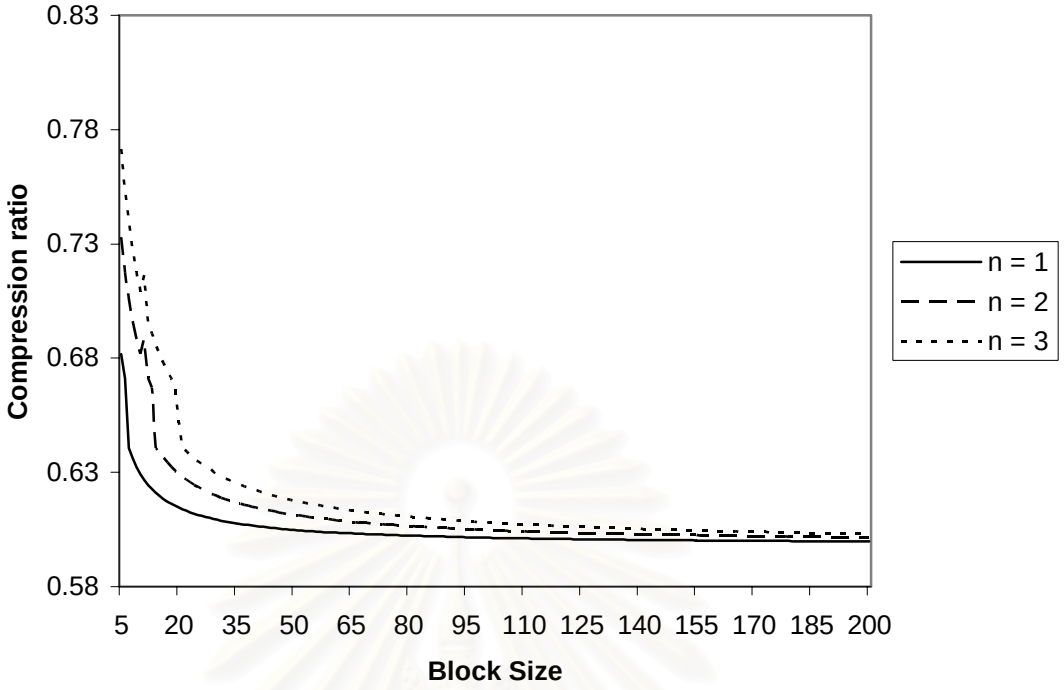
ผลที่ได้จากการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดกับแฟ้มข้อมูลทั้ง 18 แฟ้มได้ผลดังต่อไปนี้



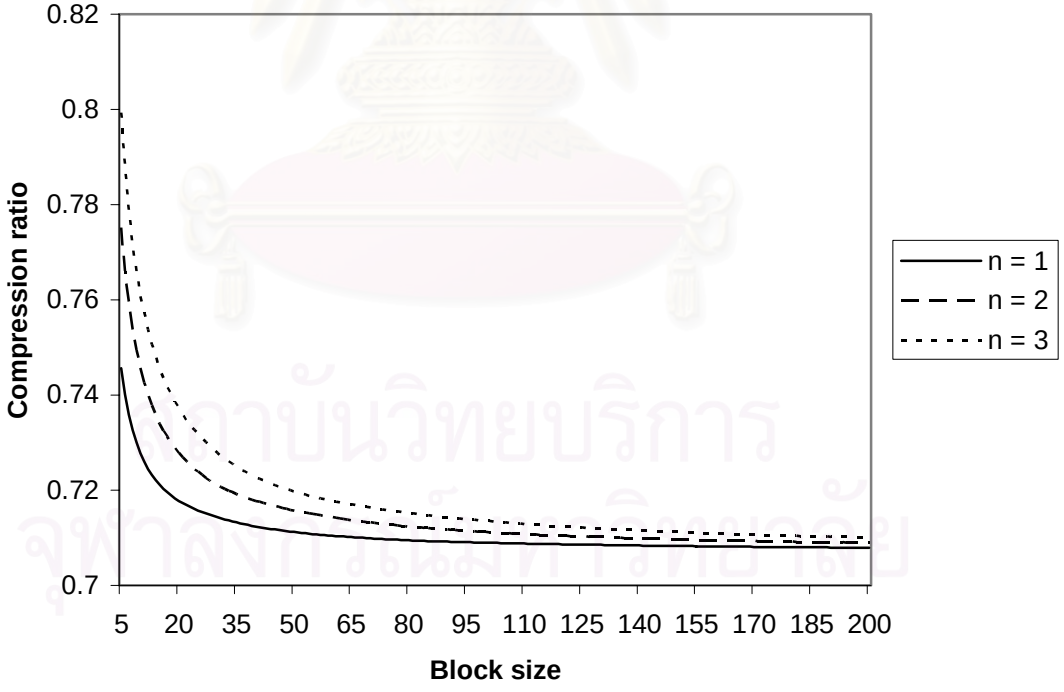
รูปที่ 4.1 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม bib



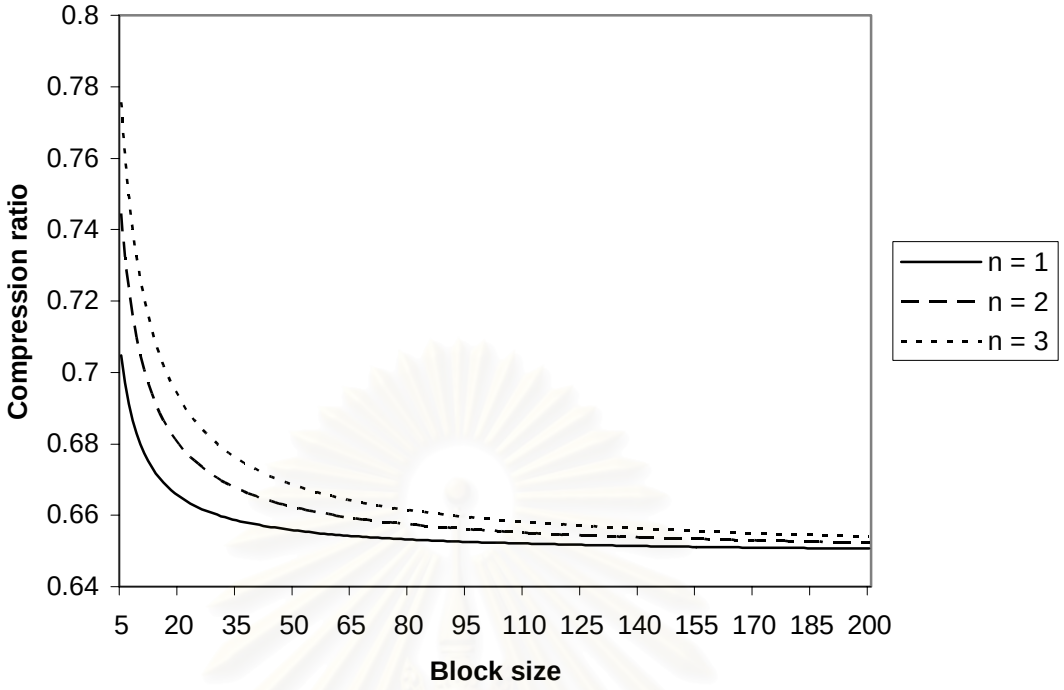
รูปที่ 4.2 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม book1



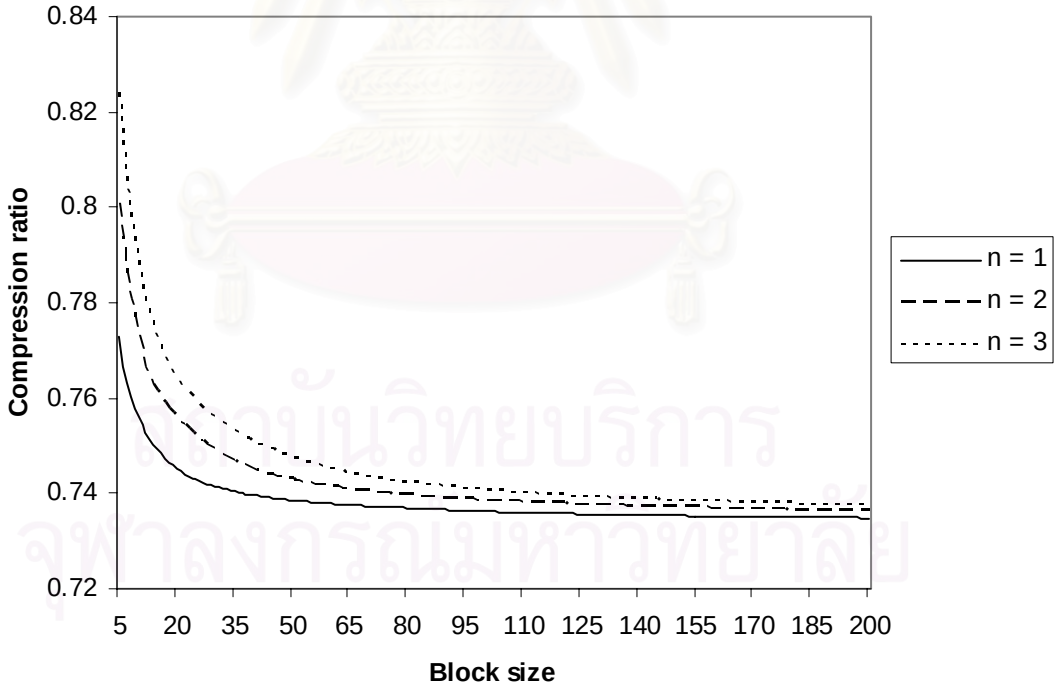
รูปที่ 4.3 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม book2



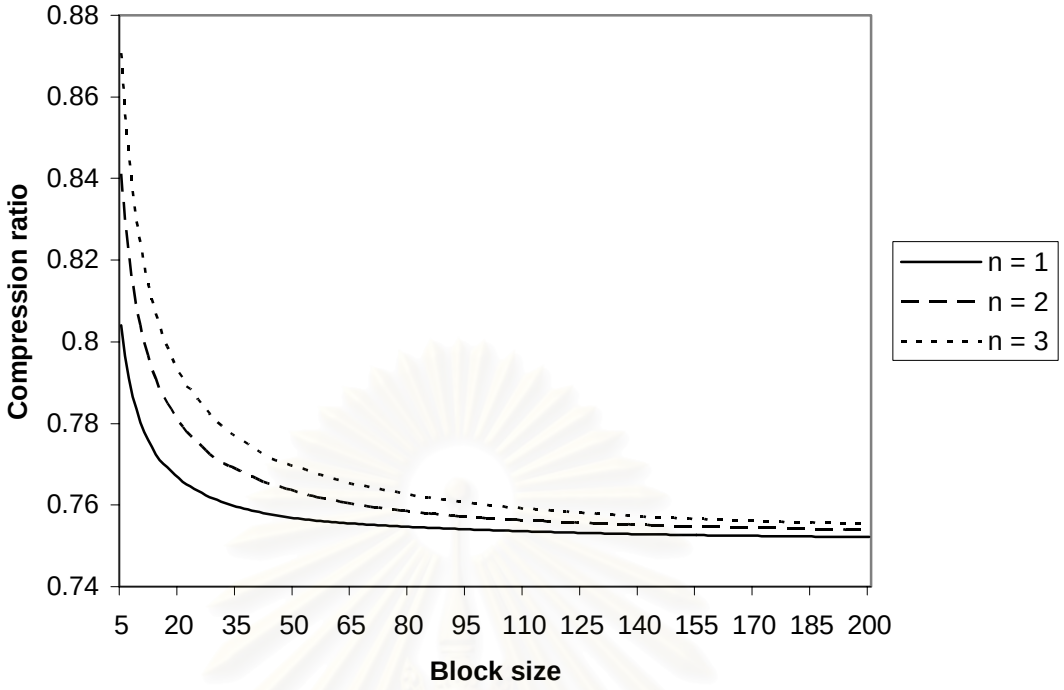
รูปที่ 4.4 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม geo



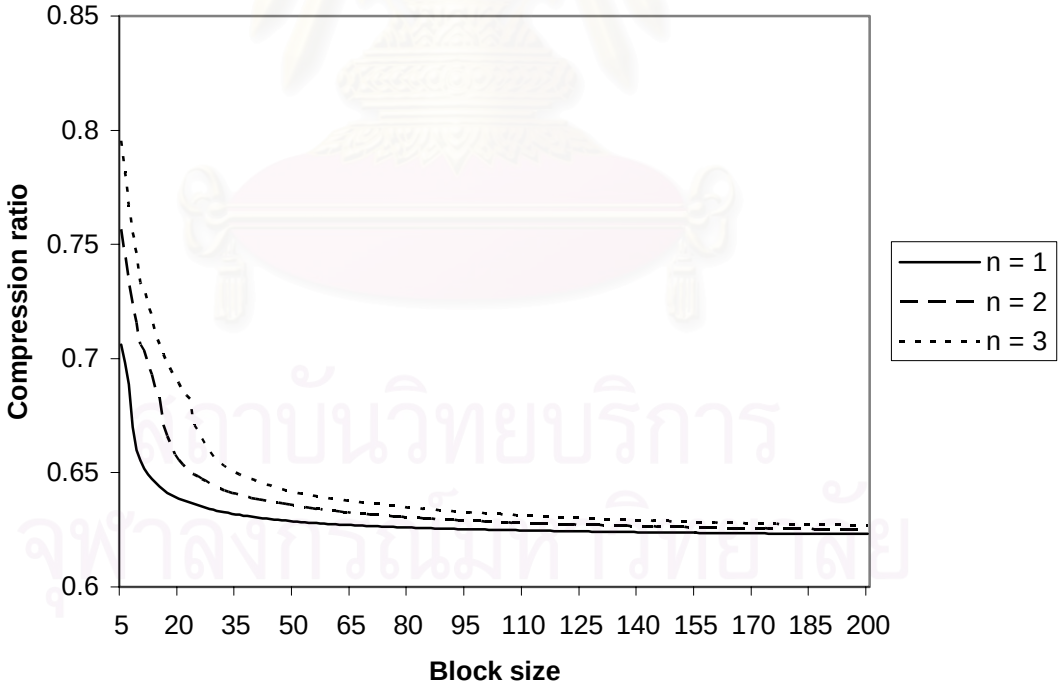
รูปที่ 4.5 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม news



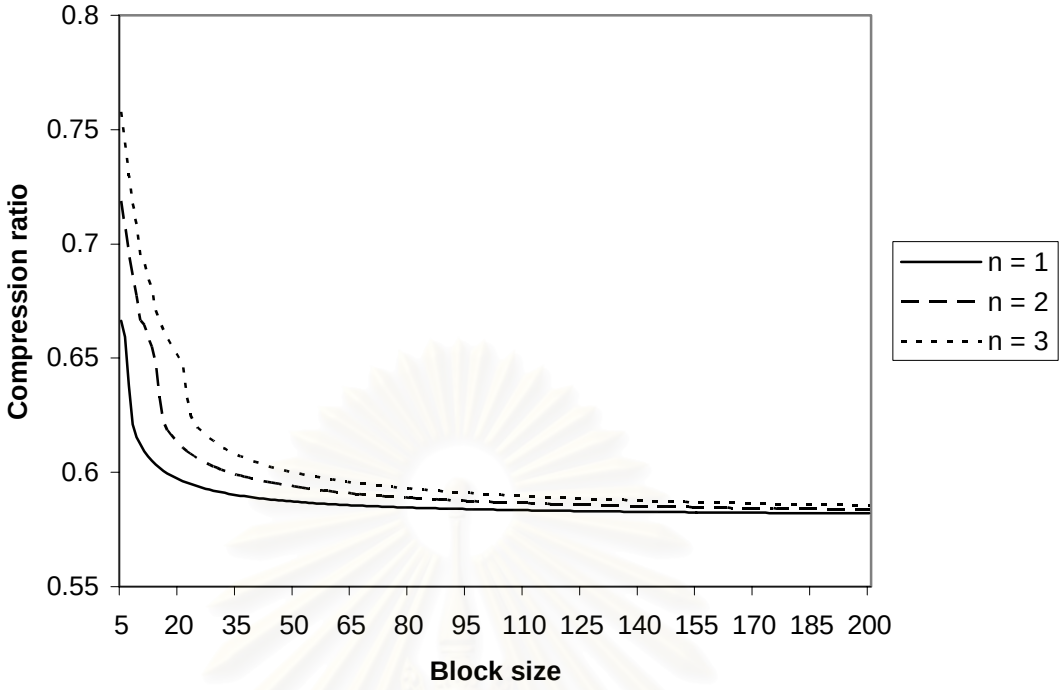
รูปที่ 4.6 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม obj1



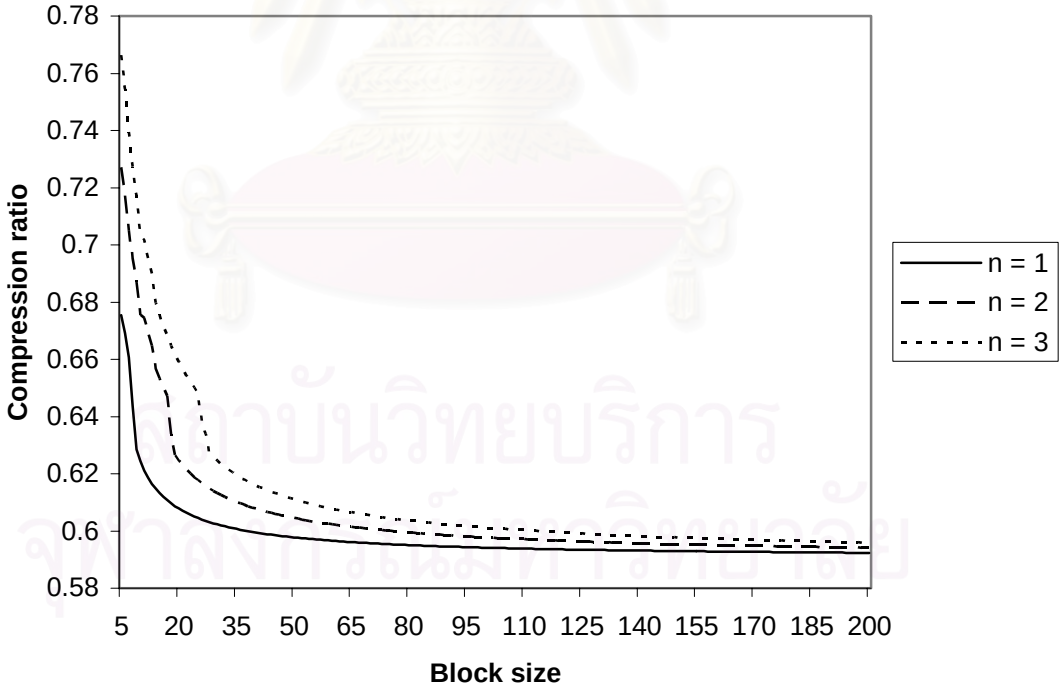
รูปที่ 4.7 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม obj2



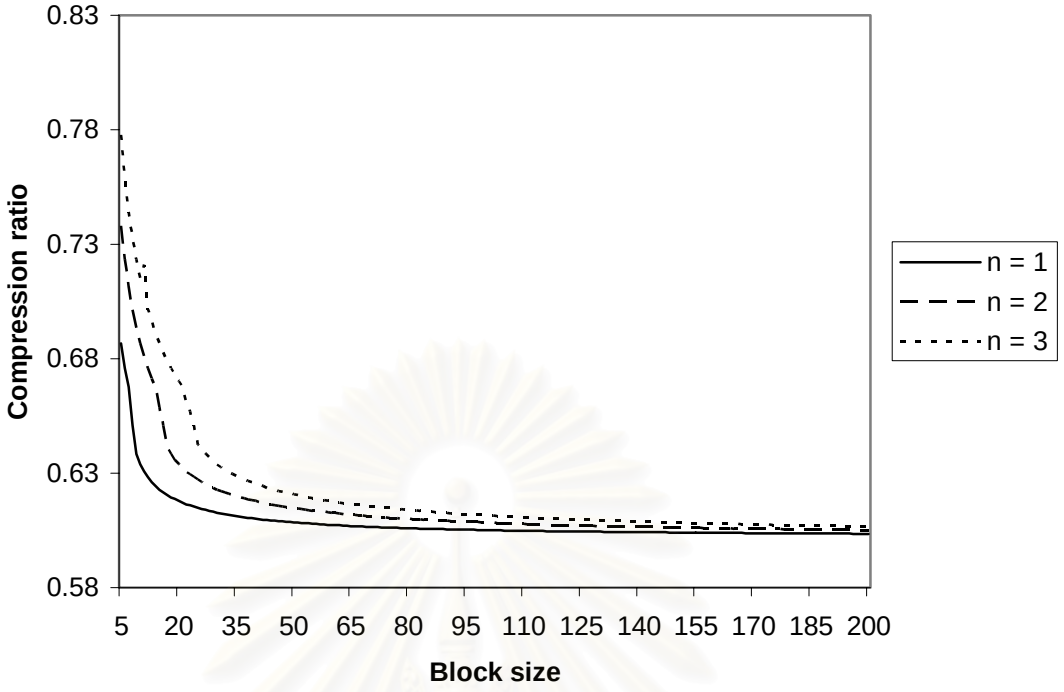
รูปที่ 4.8 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม paper1



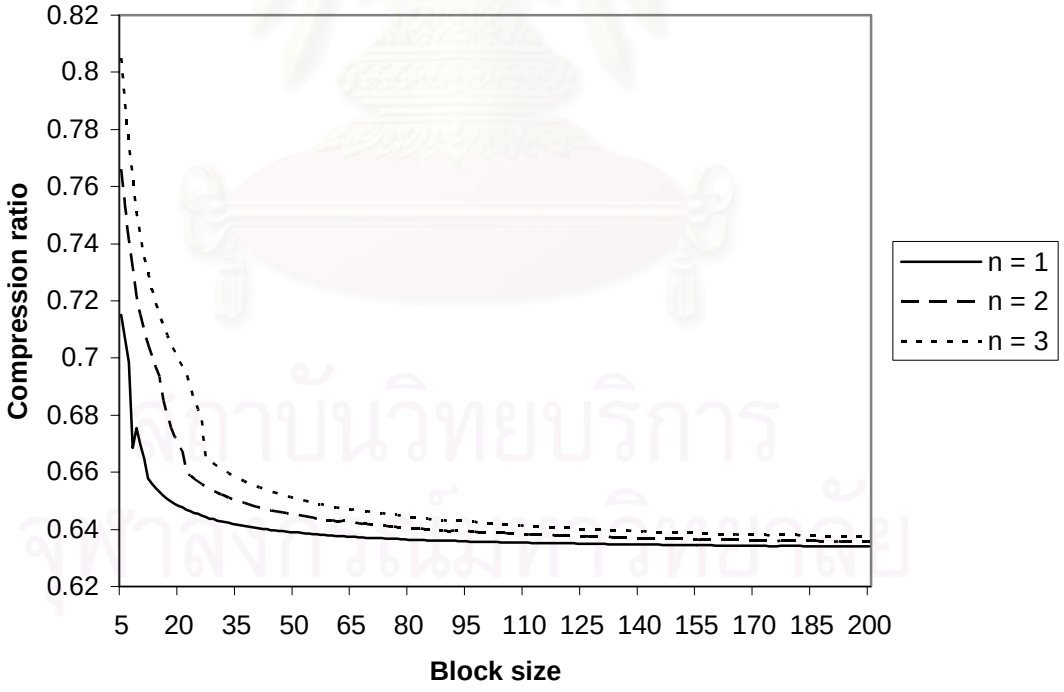
รูปที่ 4.9 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม paper2



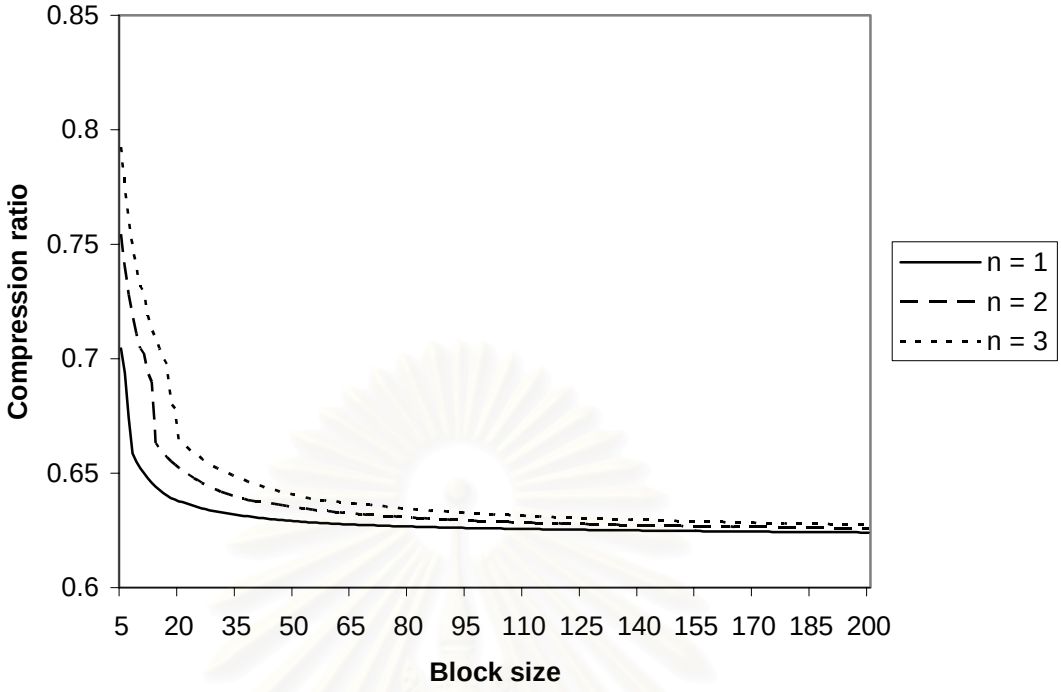
รูปที่ 4.10 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม paper3



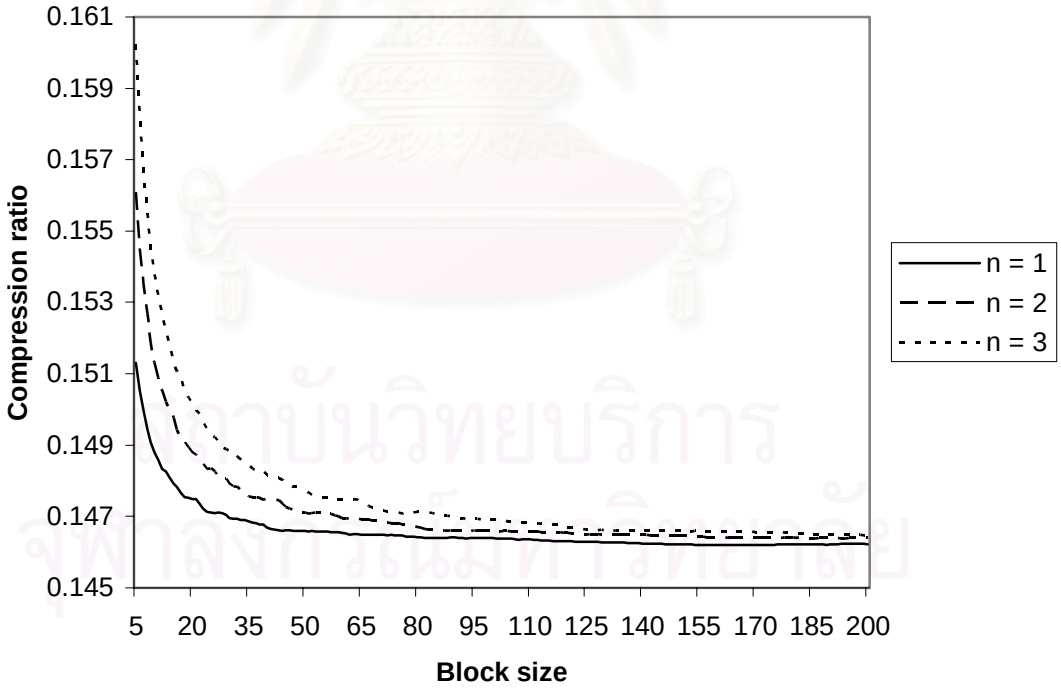
รูปที่ 4.11 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม paper4



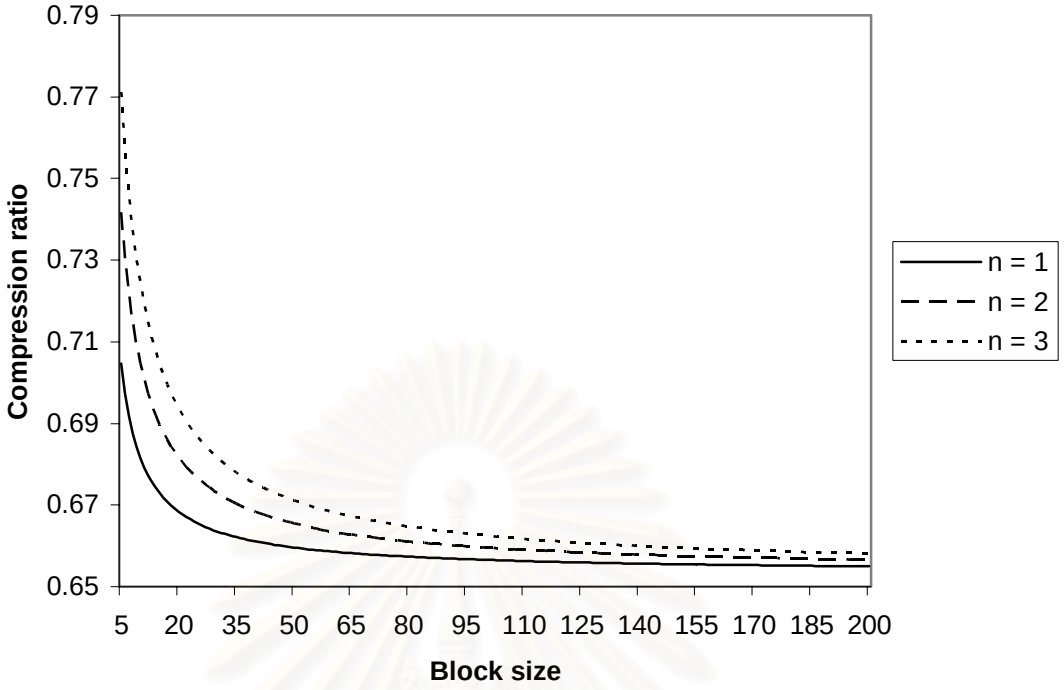
รูปที่ 4.12 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม paper5



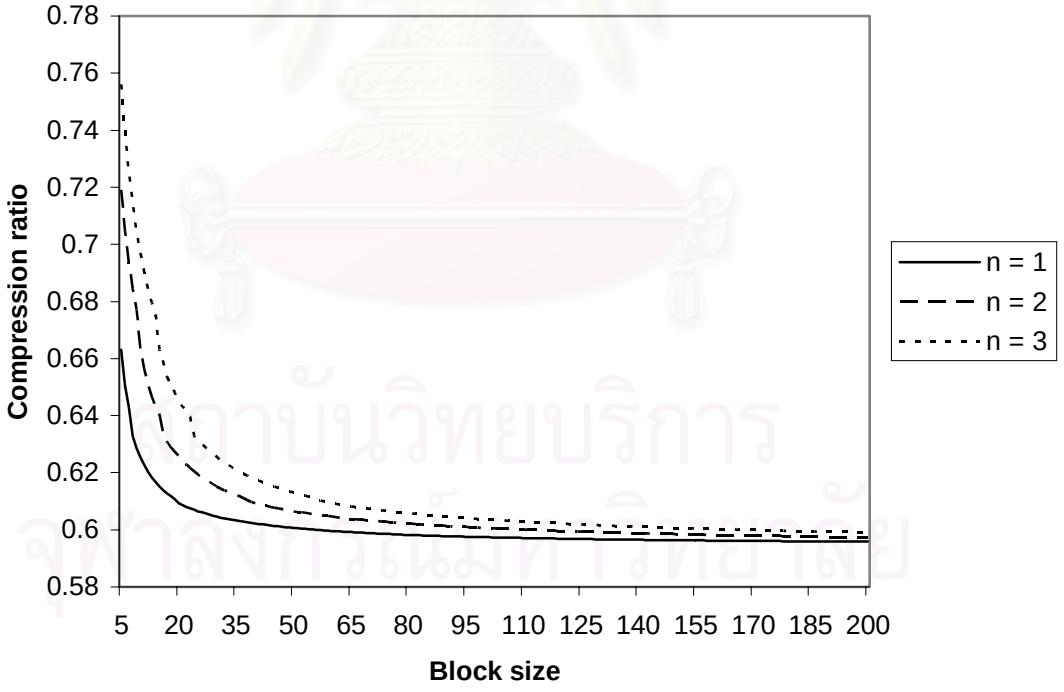
รูปที่ 4.13 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม paper6



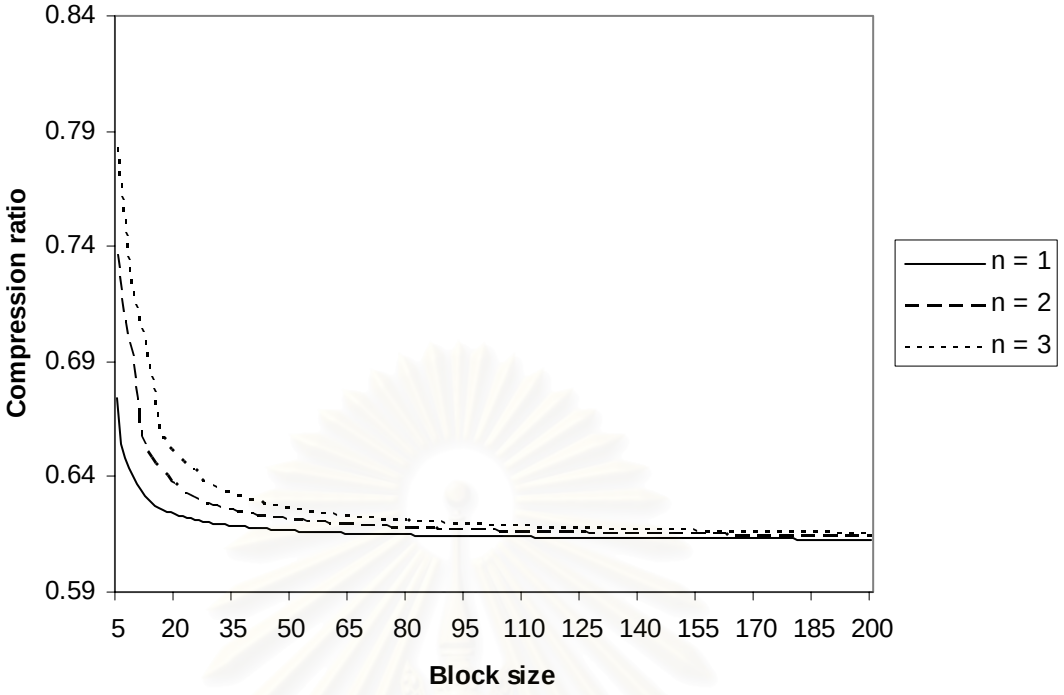
รูปที่ 4.14 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม pic



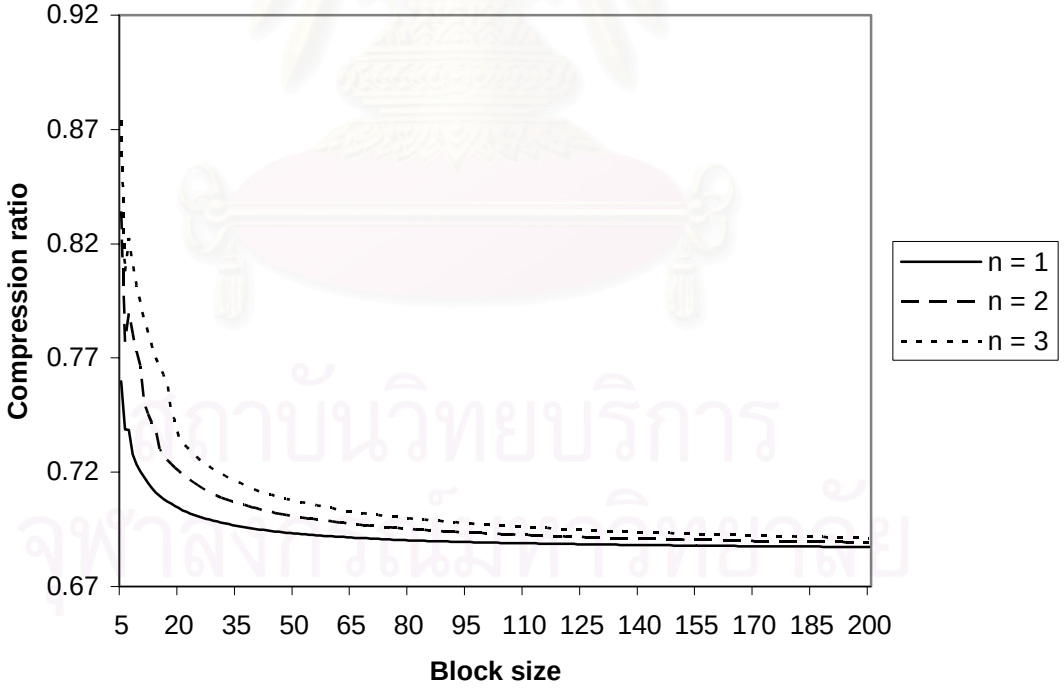
รูปที่ 4.15 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม progc



รูปที่ 4.16 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม progl



รูปที่ 4.17 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม progp

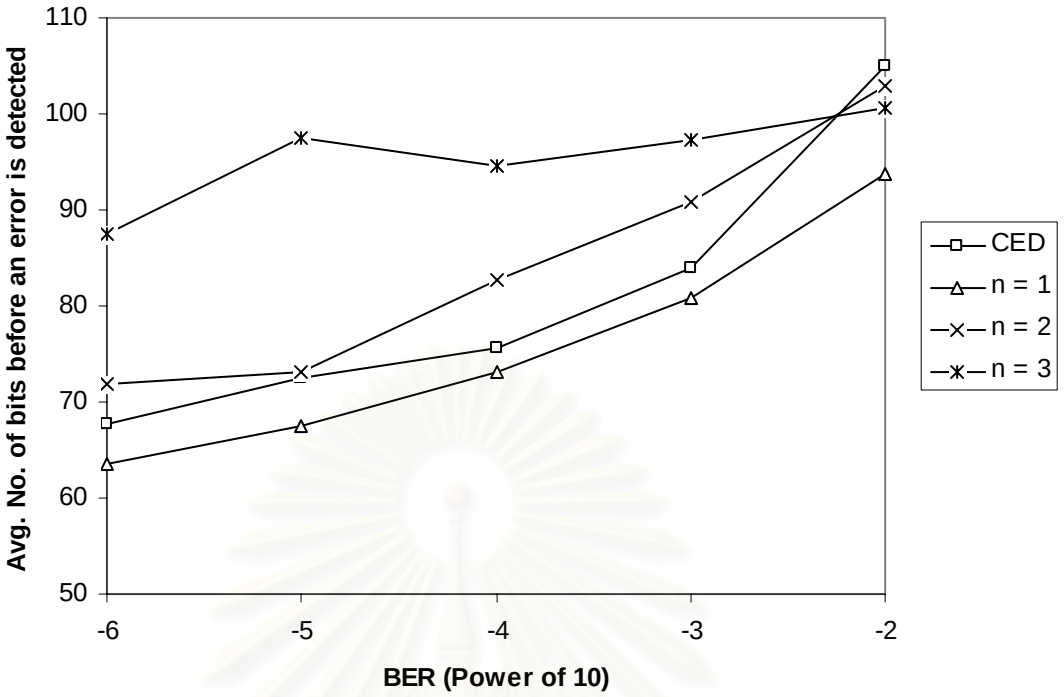


รูปที่ 4.18 ผลการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดของแฟ้ม trans

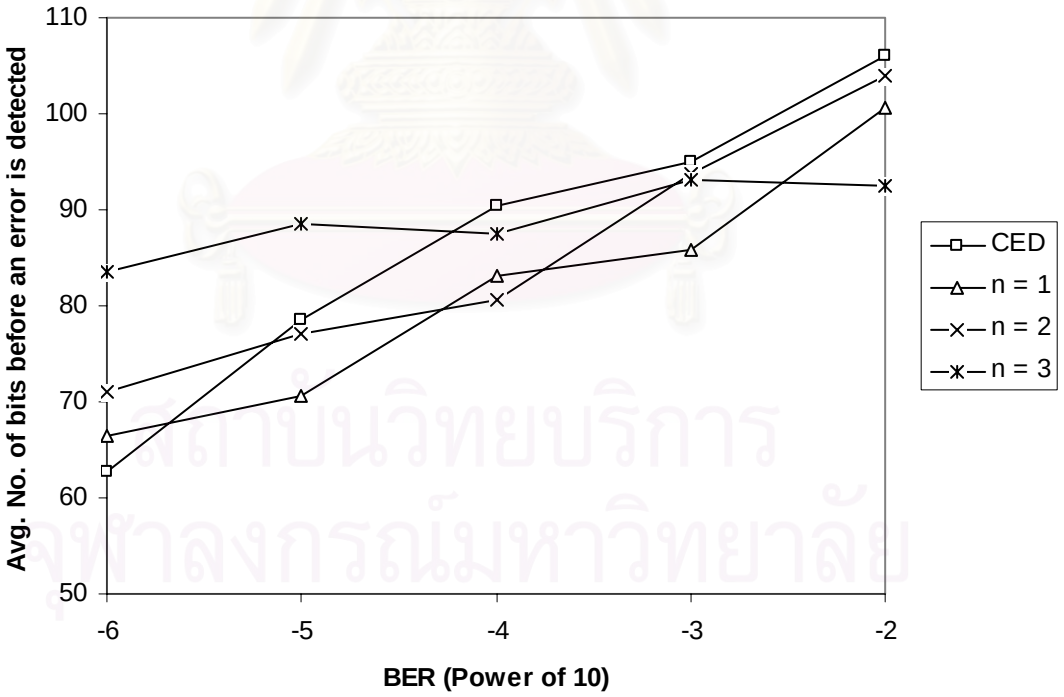
จากรูปที่ 4.1 ถึงรูปที่ 4.18 จะเห็นได้ว่าผลการเข้ารหัสโดยการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดจะเป็นไปตามผลการวิเคราะห์จากทฤษฎี ส่วนที่เห็นว่าในการทดสอบจะมีการเปลี่ยนแปลงอย่างรวดเร็วในช่วงต้นของการทดสอบกับบางแฟ้มนั้นเกิดขึ้นเนื่องจากในช่วงแรกของการเข้ารหัสแบบจำลองที่ใช้จะมีความน่าจะเป็นของสัญลักษณ์แต่ละตัวใกล้เคียงกันเมื่อมีการเปลี่ยนแปลงสัญลักษณ์ที่มีความถี่ในการเกิดสูงที่สุดจะทำให้เกิดการเปลี่ยนแปลงเกิดขึ้นอย่างรวดเร็วในบางจุดซึ่งปรากฏให้เห็นชัดเจนในแฟ้ม bib และ book1

4.4.2 ผลการทดสอบตรวจจับข้อผิดพลาด

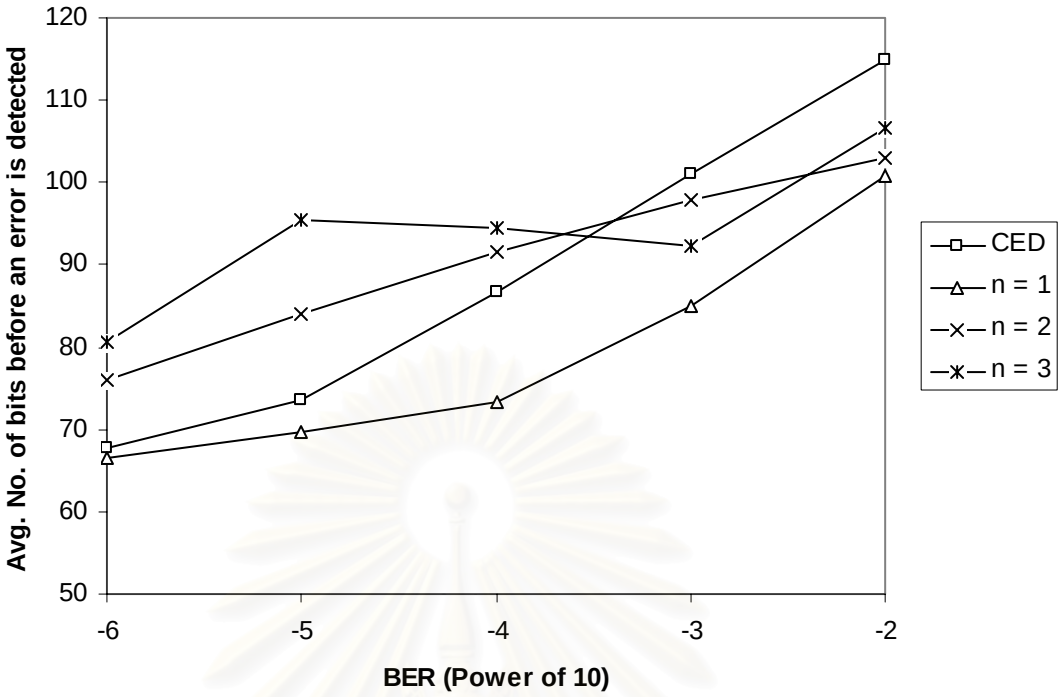
ในการทดสอบนี้จะเป็นการเลือกเอาผลจากการเข้ารหัสที่ทำไว้แล้วในการทดลองที่ผ่านมาเพื่อทดสอบการตรวจจับข้อผิดพลาดกับข้อผิดพลาดที่เกิดขึ้นด้วยอัตราข้อผิดพลาดบิตต่างๆ กัน โดยผลที่แสดงนี้เป็นจำนวนของบิตโดยเฉลี่ยที่ถูกถอดรหัสก่อนที่จะตรวจจับข้อผิดพลาดได้สำเร็จโดยจะเริ่มนับจากบิตแรกที่เกิดข้อผิดพลาดขึ้น เพื่อทำการเปรียบกับการตรวจจับข้อผิดพลาดต่อเนื่อง ดังนั้นจะเลือกใช้ข้อมูลที่ผ่านการเข้ารหัสมาแล้วมีขนาดของข้อมูลใกล้เคียงกัน โดยจะมีการเลือกตัวอย่างเพื่อใช้ในการทดสอบดังนี้ แฟ้ม bib เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.690 แฟ้ม book1 เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.611 แฟ้ม book2 เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.629 แฟ้ม geo เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.721 แฟ้ม news เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.681 แฟ้ม obj1 เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.759 แฟ้ม obj2 เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.777 แฟ้ม paper1 เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.658 แฟ้ม paper2 เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.617 แฟ้ม paper3 เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.623 แฟ้ม paper4 เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.632 แฟ้ม paper5 เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.652 แฟ้ม paper6 เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.653 แฟ้ม pic เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.149 แฟ้ม progc เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.683 แฟ้ม progl เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.621 แฟ้ม progp เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.633 และแฟ้ม trans เลือกข้อมูลที่มีอัตราส่วนการบีบอัดเท่ากับ 0.725 ผลการทดสอบการตรวจจับข้อผิดพลาดได้รับผลลัพธ์ดังต่อไปนี้



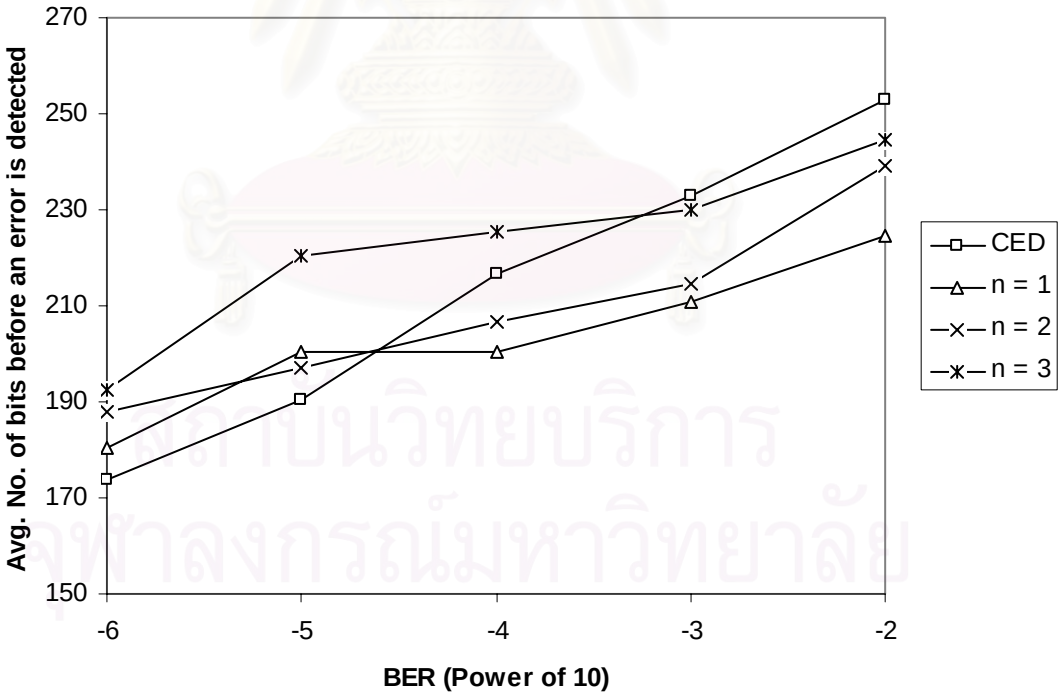
รูปที่ 4.19 ผลการตรวจจับข้อผิดพลาดของแฟ้ม bib



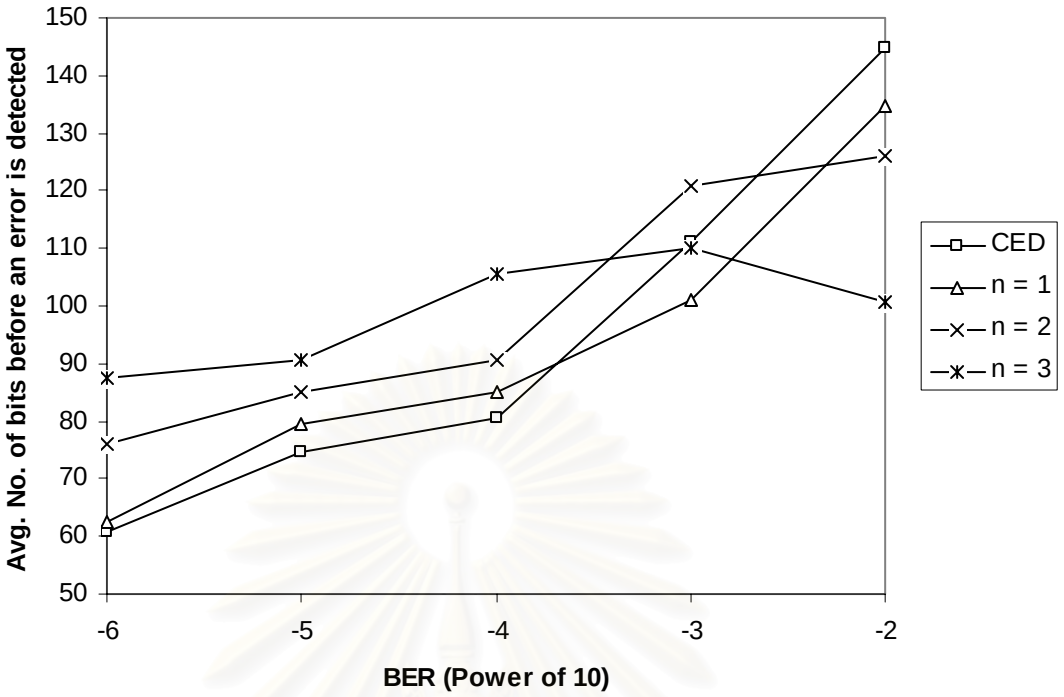
รูปที่ 4.20 ผลการตรวจจับข้อผิดพลาดของแฟ้ม book1



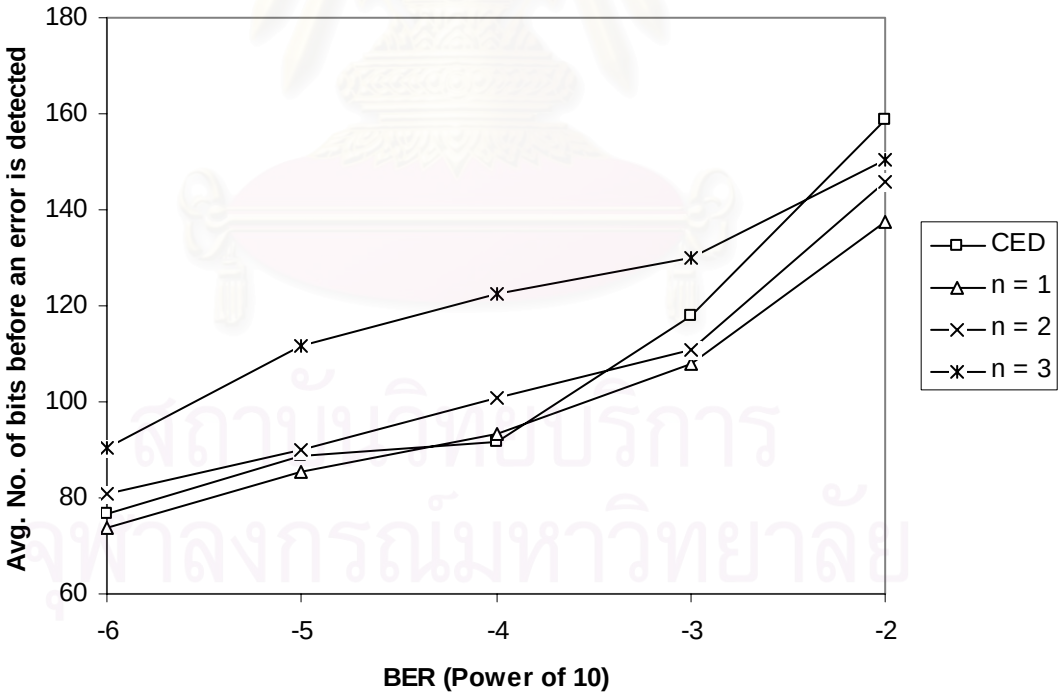
รูปที่ 4.21 ผลการตรวจจับข้อผิดพลาดของแฟ้ม book2



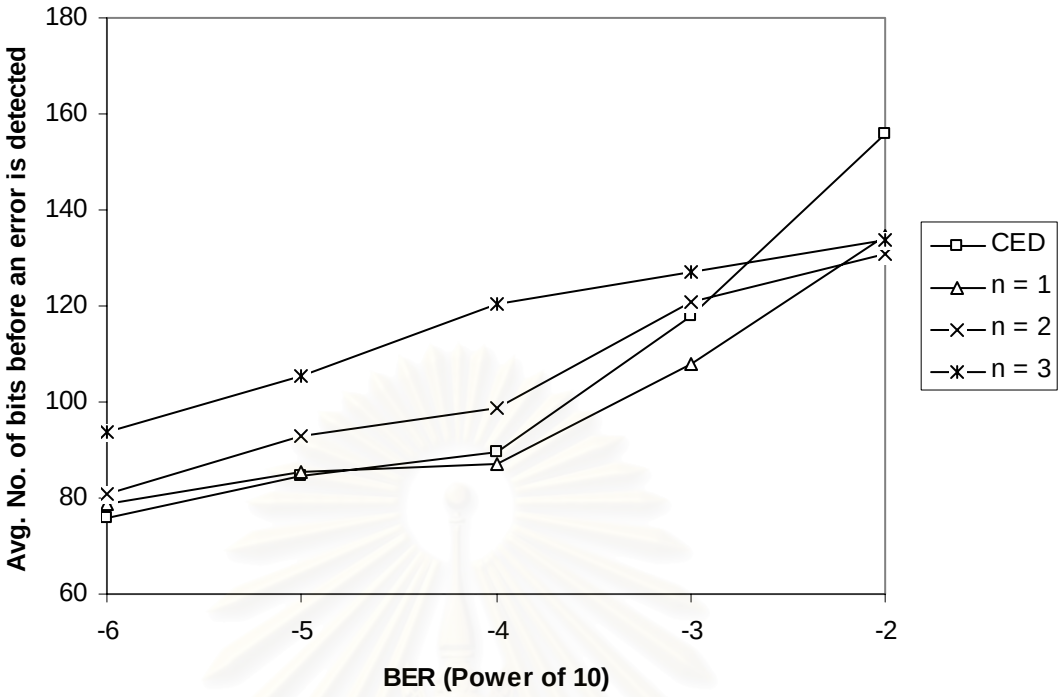
รูปที่ 4.22 ผลการตรวจจับข้อผิดพลาดของแฟ้ม geo



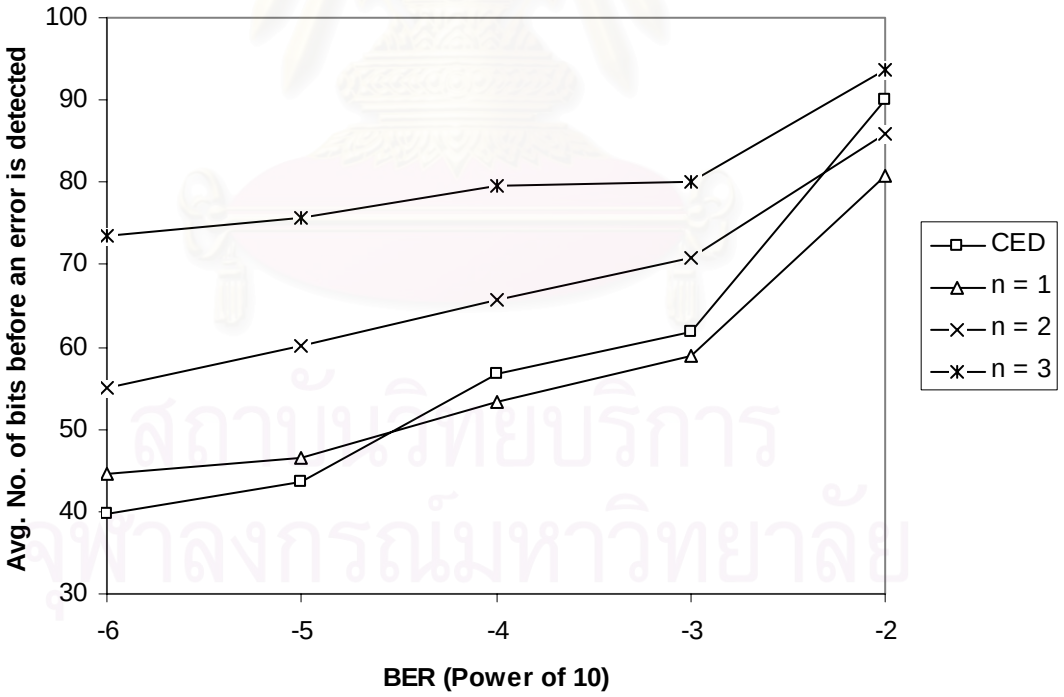
รูปที่ 4.23 ผลการตรวจจับข้อผิดพลาดของแฟ้ม news



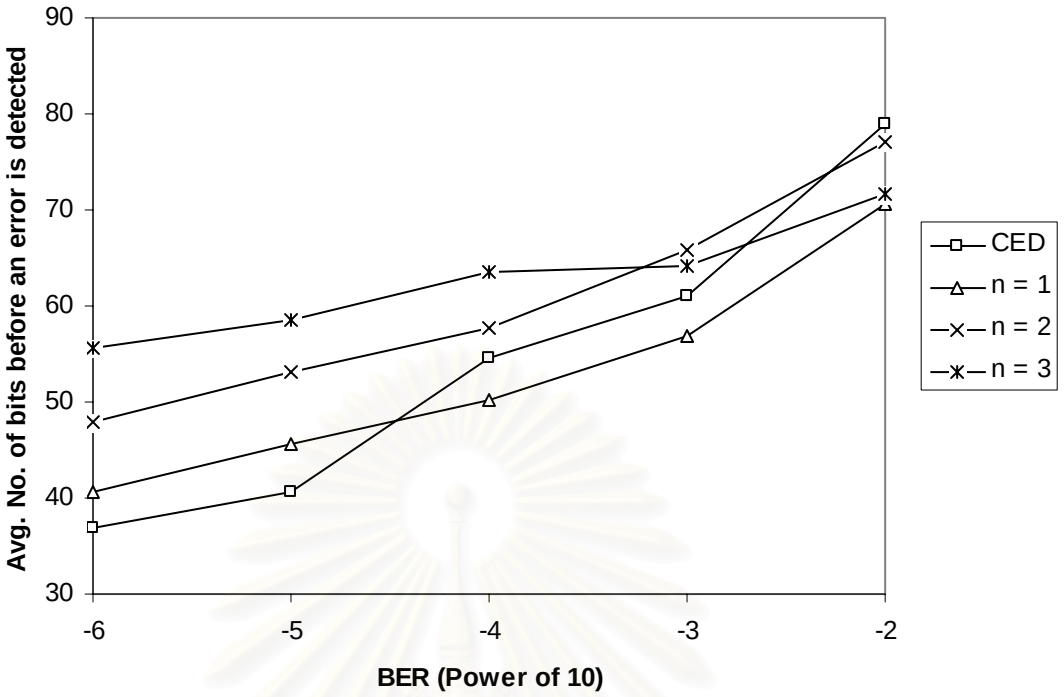
รูปที่ 4.24 ผลการตรวจจับข้อผิดพลาดของแฟ้ม obj1



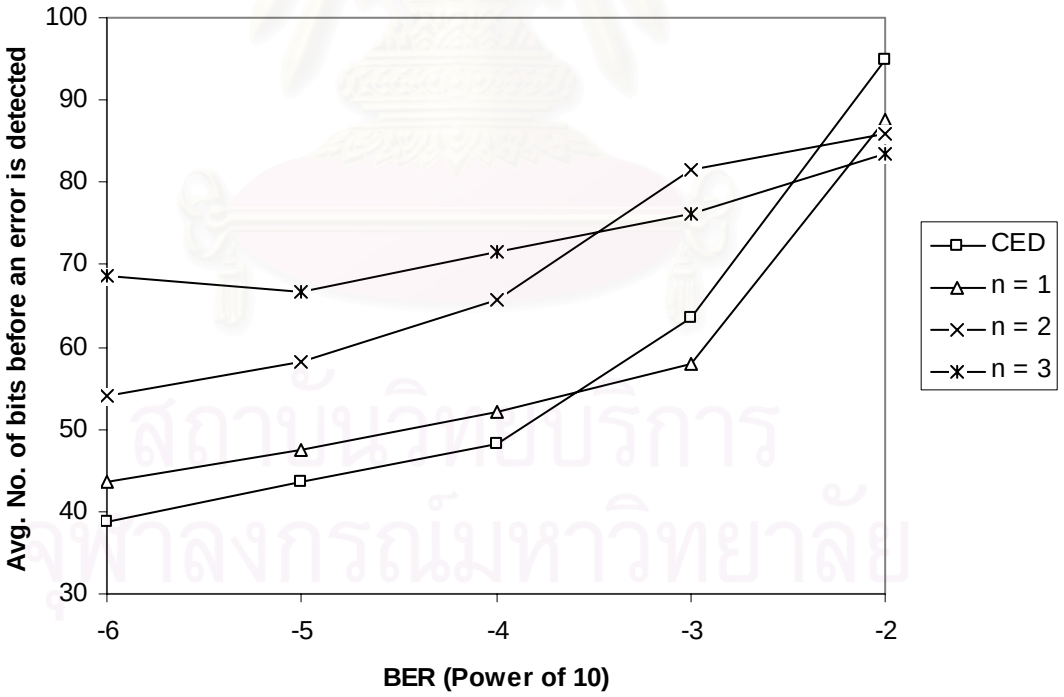
รูปที่ 4.25 ผลการตรวจจับข้อผิดพลาดของแฟ้ม obj2



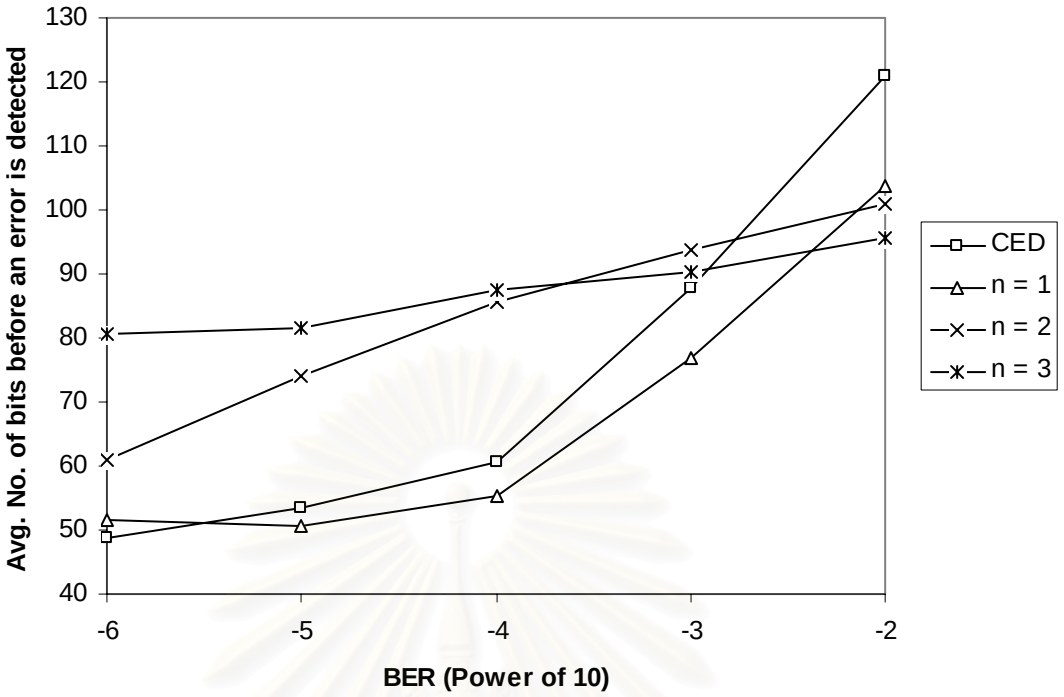
รูปที่ 4.26 ผลการตรวจจับข้อผิดพลาดของแฟ้ม paper1



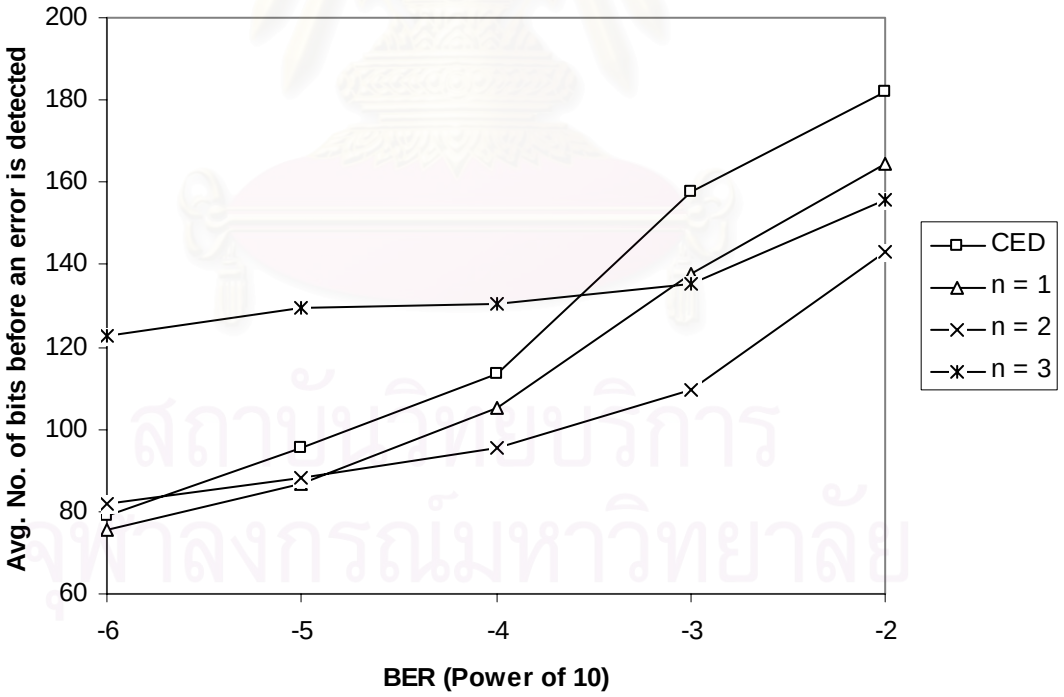
รูปที่ 4.27 ผลการตรวจจับข้อผิดพลาดของแฟ้ม paper2



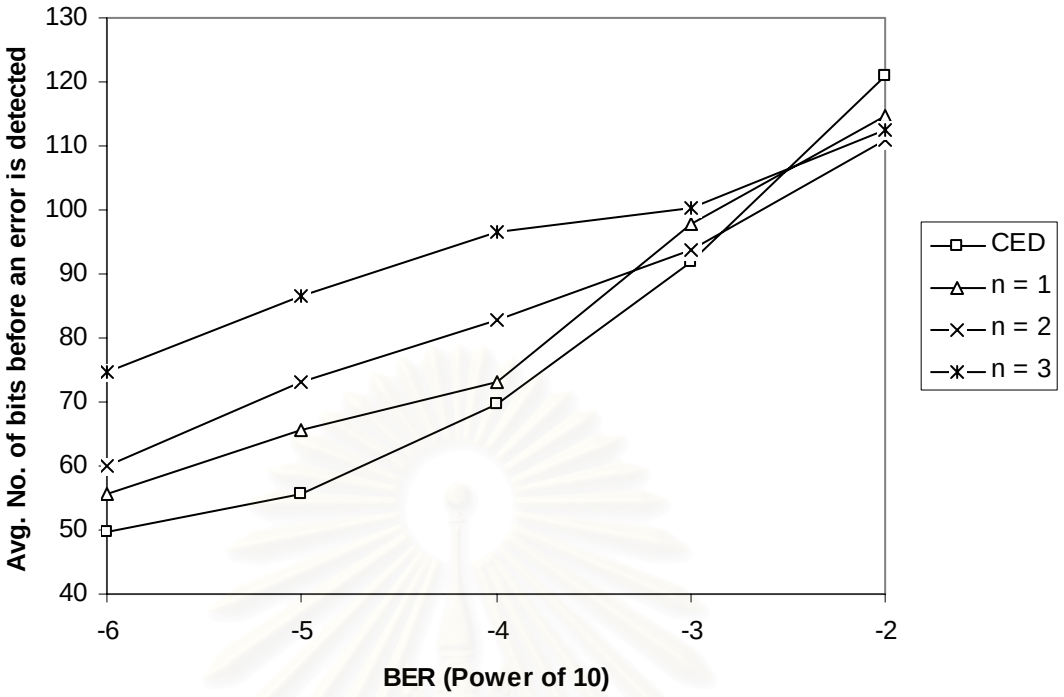
รูปที่ 4.28 ผลการตรวจจับข้อผิดพลาดของแฟ้ม paper3



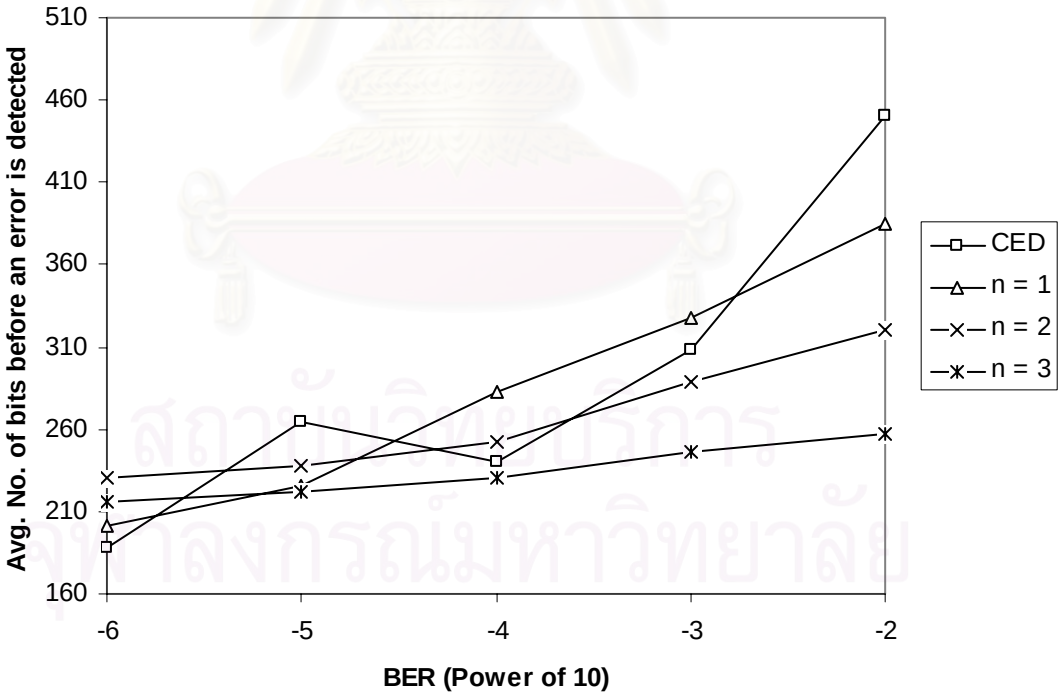
รูปที่ 4.29 ผลการตรวจจับข้อผิดพลาดของแฟ้ม paper4



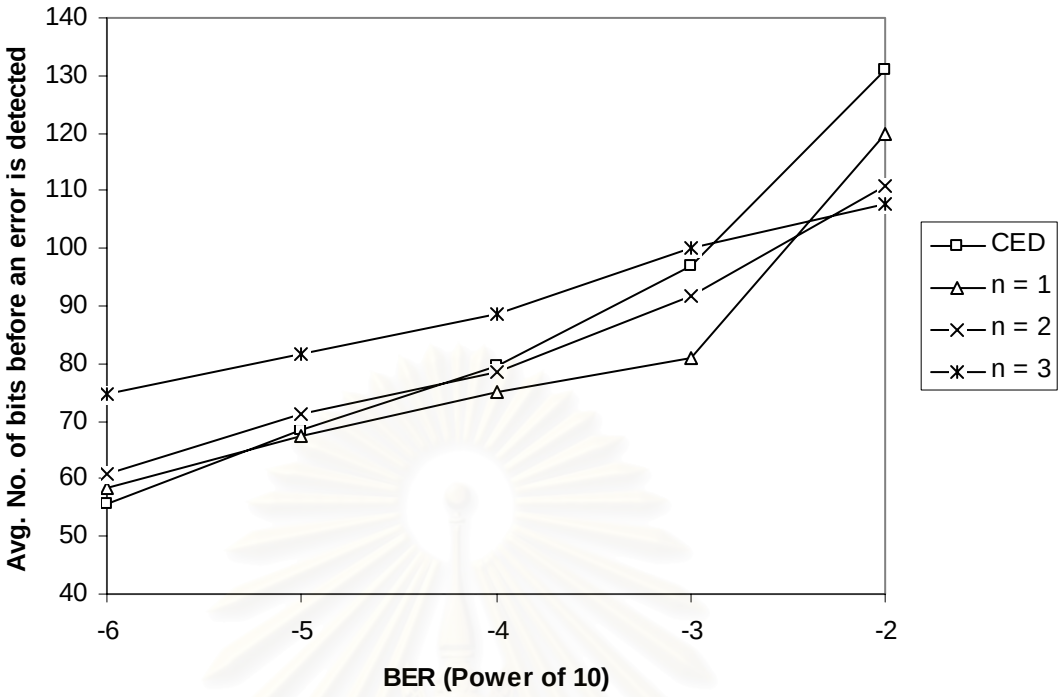
รูปที่ 4.30 ผลการตรวจจับข้อผิดพลาดของแฟ้ม paper5



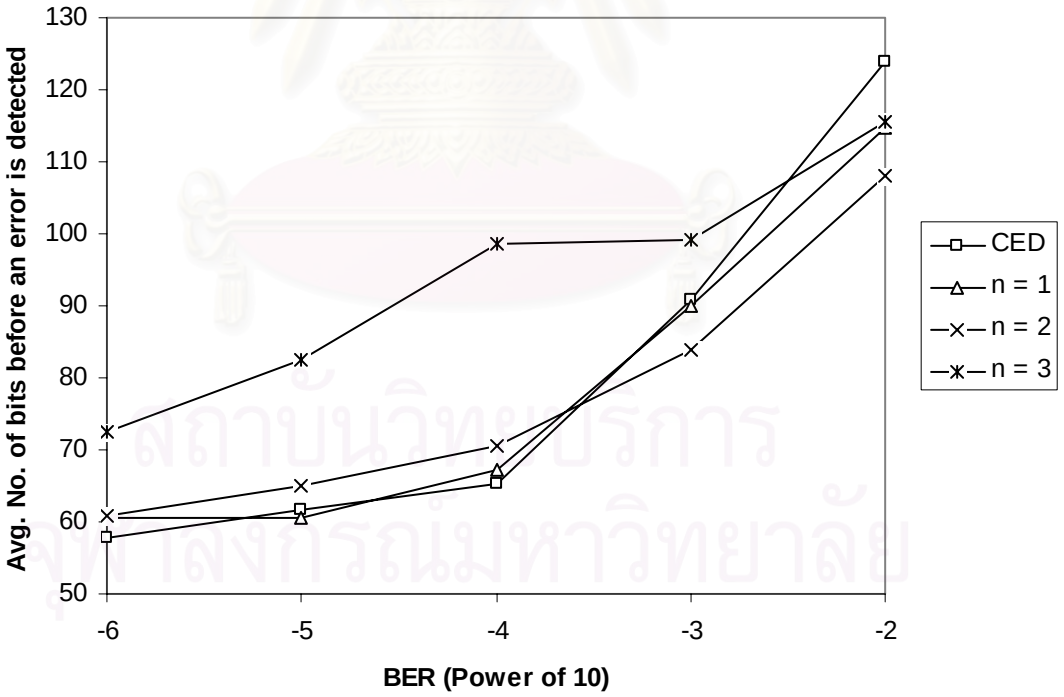
รูปที่ 4.31 ผลการตรวจจับข้อผิดพลาดของแฟ้ม paper6



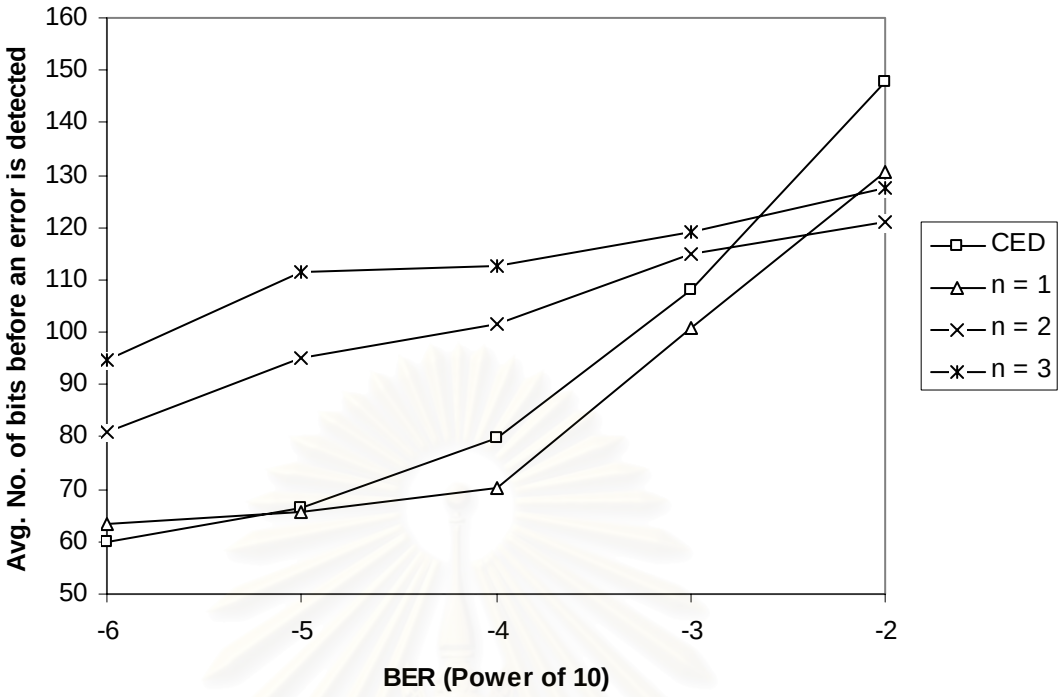
รูปที่ 4.32 ผลการตรวจจับข้อผิดพลาดของแฟ้ม pic



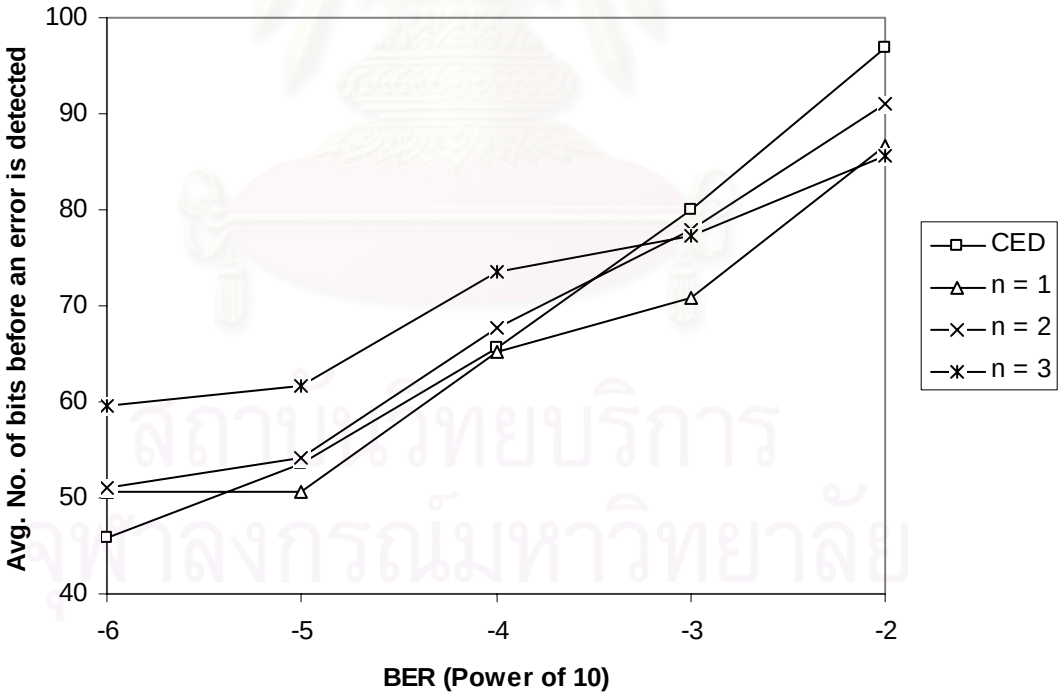
รูปที่ 4.33 ผลการตรวจจับข้อผิดพลาดของแฟ้ม progc



รูปที่ 4.34 ผลการตรวจจับข้อผิดพลาดของแฟ้ม progl



รูปที่ 4.35 ผลการตรวจจับข้อผิดพลาดของแฟ้ม progp



รูปที่ 4.36 ผลการตรวจจับข้อผิดพลาดของแฟ้ม trans

4.4.3 การทดสอบความซับซ้อนในการคำนวณ

ในการทดสอบนี้จะทำการเข้ารหัสข้อมูลด้วยวิธีการเข้ารหัสที่ใช้พารามิเตอร์ต่างๆ กันเพื่อวัดและเปรียบเทียบความเร็วในการเข้ารหัส โดยการเข้ารหัสจะทำบนเครื่องคอมพิวเตอร์ที่มีหน่วยประมวลกลางเป็น Intel Pentium III 666 MHz ซึ่งมีหน่วยความจำ 256 MB และใช้ระบบปฏิบัติการวินโดว์ 2000 โดยจะทำการเข้ารหัสข้อมูลทั้ง 18 แฟ้มและทำการเปลี่ยนแปลงพารามิเตอร์ของการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดเพื่อดูการเปลี่ยนแปลงของความเร็วในการคำนวณที่เกิดขึ้น ผลการทดสอบที่ได้เป็นดังต่อไปนี้

ตารางที่ 4.2 เวลาที่ใช้ในการประมวลผล

วิธีการเข้ารหัส	เวลาที่ใช้ในการประมวลผล(วินาที)
การเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่อง	7.37
การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุด $k=5$ และ $n=1$	6.79
การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุด $k=5$ และ $n=2$	7.20
การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุด $k=5$ และ $n=3$	7.50
การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุด $k=50$ และ $n=1$	6.30
การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุด $k=50$ และ $n=2$	6.41
การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุด $k=50$ และ $n=3$	6.53

จากข้อมูลในตารางที่ 4.2 จะเห็นได้ว่าการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดจะทำงานได้เร็วกว่าการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่อง แต่ยิ่งเพิ่มปริมาณของเครื่องหมายที่ใส่เพิ่มมากขึ้นแค่ไหนก็จะใช้ความซับซ้อนในการคำนวณมากขึ้นเท่านั้น ซึ่งจะเห็นได้จากการที่จะมีการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดบางการทดสอบที่มีความซับซ้อนในการคำนวณสูงกว่าการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่อง

บทที่ 5

สรุปผลการวิจัย และข้อเสนอแนะ

5.1 สรุปผลการวิจัย

วิทยานิพนธ์นี้นำเสนอวิธีการเข้ารหัสตรวจจับข้อผิดพลาดที่มีการนำเอากระบวนการคำนวณและคุณสมบัติการแพร่กระจายของข้อผิดพลาดของการเข้ารหัสเชิงเลขคณิตมาใช้ เพื่อช่วยให้การเข้ารหัสเชิงเลขคณิตสามารถที่จะทำการตรวจจับข้อผิดพลาดได้โดยไม่จำเป็นต้องมีกระบวนการอื่นๆ เพื่อทำการตรวจจับข้อผิดพลาด วิธีการที่นำเสนอนั้นจะเลือกสัญลักษณ์ที่มีความถี่สูงสุดมาใช้เป็นเครื่องหมายเพื่อตรวจจับข้อผิดพลาดที่เกิดขึ้น เนื่องจากหากเลือกเครื่องหมายที่มีความถี่ในการเกิดสูงสุดมาใช้จะเป็นการรับประกันว่าความซ้ำซ้อนที่เกิดขึ้นจากเครื่องหมายแต่ละตัวจะมีค่าน้อยที่สุด ซึ่งจะทำให้การเข้ารหัสเชิงเลขคณิตลดประสิทธิภาพลงน้อยที่สุดในการใส่เครื่องหมายแต่ละครั้ง และทำให้สามารถทำการตรวจจับข้อผิดพลาดได้บ่อยครั้งยิ่งขึ้นหากใช้ความซ้ำซ้อนเท่ากัน วิธีการที่นำเสนอสามารถควบคุมความซ้ำซ้อนที่ถูกใช้ได้ดีกว่าวิธีการเลือกเครื่องหมายอื่นเนื่องจากนอกจากจะสามารถควบคุมขนาดของบล็อกได้แล้วยังสามารถควบคุมจำนวนของเครื่องหมายที่ใช้ในแต่ละบล็อกได้อีกด้วย

จากผลการทดสอบในบทที่ 4 จะเห็นได้ว่าการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดจะให้ผลอัตราส่วนการบีบอัดข้อมูลตรงตามที่ใช้ทฤษฎีวิเคราะห์ไว้ โดยเมื่อนำเอาผลลัพธ์ที่ได้จากการเข้ารหัสไปผ่านการใส่ข้อผิดพลาดเข้าไปและนำไปทดสอบการตรวจจับข้อผิดพลาด การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดจะสามารถตรวจจับข้อผิดพลาดได้อย่างถูกต้องและมีจำนวนบิตที่ผ่านการถอดรหัสก่อนที่จะตรวจพบข้อผิดพลาดใกล้เคียงหรือน้อยกว่าการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่อง เนื่องจากการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่องอาจมีการใช้ความน่าจะเป็นสำหรับสัญลักษณ์ต้องห้ามน้อยเกินไปซึ่งทำให้การตรวจจับข้อผิดพลาดเป็นไปได้ยาก ในขณะที่การเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดนั้นจะใช้สัญลักษณ์ที่มีความน่าจะเป็นในการเกิดสูงที่สุดเป็นเครื่องหมายดังนั้นความน่าจะเป็นที่จะตรวจจับข้อผิดพลาดได้แต่ละครั้งจะมีค่ามากกว่า จึงทำให้สามารถจะทำการตรวจจับข้อผิดพลาดในแต่ละครั้งได้แม่นยำกว่า แต่จะทำการตรวจจับข้อผิดพลาดได้น้อยครั้งกว่า

จากการทดสอบความซับซ้อนในการเข้ารหัสนั้นจะเห็นได้ว่าการเข้ารหัสโดยใช้สัญลักษณ์ความถี่สูงสุดนั้นจะใช้ความซับซ้อนน้อยกว่าการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่อง เมื่อการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดไม่ใช้เครื่องหมายบ่อยครั้งจนเกินไปเนื่องจากการเข้ารหัสเครื่องหมายแต่ละครั้งจะต้องมีการหาสัญลักษณ์ที่มีความถี่ในการเกิดสูงที่สุดก่อนจากนั้นจึงเลือกสัญลักษณ์นั้นเป็นเครื่องหมายและทำการเข้ารหัสเครื่องหมายนั้น

ตัวอย่างของการใช้เครื่องหมายบ่อยครั้งเกินไปจะเห็นได้จากการทดสอบที่ใช้ขนาดของบล็อกเท่ากับ 5 และใส่เครื่องหมาย 3 เครื่องหมายในแต่ละบล็อกซึ่งจะเห็นได้ว่าความซับซ้อนในการคำนวณจะสูงกว่าความซับซ้อนในการคำนวณของการเข้ารหัสตรวจจับข้อผิดพลาดต่อเนื่อง

5.2 ข้อเสนอแนะ

ปัญหาที่เกิดขึ้นในการทำงานวิจัยในวิทยานิพนธ์คือ งานวิจัยที่เกี่ยวข้องยังมีไม่มากนักทำให้การค้นคว้าข้อมูลสามารถทำได้ยาก อีกทั้งงานวิจัยที่เกี่ยวข้องบางงานวิจัย ไม่มีผลการทดสอบที่สมบูรณ์ทำให้การทดสอบเปรียบเทียบกับงานวิจัยเหล่านั้นต้องมีการทำซ้ำงานวิจัยนั้น ซึ่งทำให้ผลการทดสอบเปรียบเทียบอาจมีความคลาดเคลื่อนเกิดขึ้นได้

งานที่ควรได้รับการศึกษาหรือพัฒนาต่อไปในอนาคต คือ

1. ควรมีการปรับให้พารามิเตอร์ของการเข้ารหัสตรวจจับข้อผิดพลาดโดยใช้สัญลักษณ์ความถี่สูงสุดเป็นแบบปรับตัวได้เพื่อให้การตรวจจับข้อผิดพลาดมีความปรับตัวให้เข้ากับความสำคัญของข้อมูลเป็นส่วนๆ ได้

2. ศึกษาหาค่าเหมาะที่สุดของความน่าจะเป็นของสัญลักษณ์ที่ใช้เป็นเครื่องหมายเพื่อให้ได้การตรวจจับข้อผิดพลาดที่มีประสิทธิภาพสูงขึ้น

3. ศึกษาการเข้ารหัสแก้ข้อผิดพลาดโดยใช้การกระจายของข้อผิดพลาดของการเข้ารหัสเชิงเลขคณิต

รายการอ้างอิง

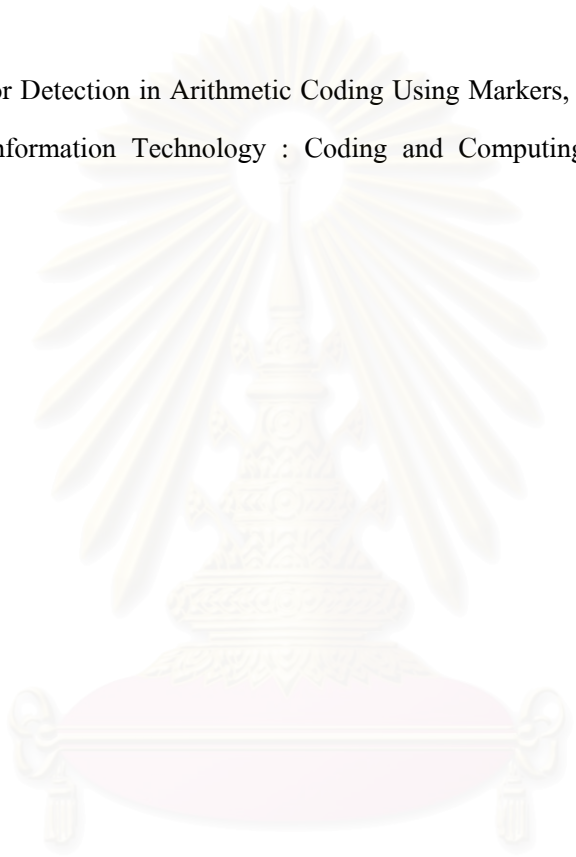
1. G. Langdon. An Introduction to Arithmetic Coding. IBM J. Res. Develop. vol.28 (March 1984) : 135-149.
2. I.H. Witten, J.G. Cleary and R. Neal. Arithmetic coding for data compression. Communication of ACM. no.6 (June 1987) : 520-540.
3. A. Moffat, R.M. Neal, I.H. Witten. Arithmetic Coding Revisited. ACM Trans. on Inform. Sys. vol.16, no.3 (July 1998) : 256-294.
4. D. A. Huffman. A method for construction of minimum redundancy codes. Proceedings of IRE. vol.40, no.10 (Sep 952) : 1098-1101.
5. C.E. Shannon. A Mathematical theory of Communication. The Bell System Technical Journal. vol.27 (July, Oct 1948) : 379-423, 623-656.
6. K.M.S. Soyjaudah and S. Ramsamy. A Comparative Study of Context Free Models of Arithmetic Coding. Proceedings of IEEE EUROCON'2001. vol.2 (July 2001) : 428-431.
7. C. Boyd, J.G. Cleary, S.A. Irvine, I. Rinsma-Melchert and I.H. Witten. Integrating Error Detection into Arithmetic Coding. IEEE Trans. on Comm. vol.45, no.1 (Jan 1997) : 1-3.
8. R. Anand, K. Ramchandran, and I. V. Kozintsev. Continuous Error Detection (CED) for Reliable Communication. IEEE trans. on Comm. vol.49, no.9 (Sep 2001) : 1540-1549.
9. J. Sayir. Arithmetic Coding for Noisy Channels. Proceedings of IEEE ITW 1999. vol.1 (June 1999) : 69-71.
10. G.F. Elmasry. Joint Lossless-Source and Channel Coding Using Automatic Repeat Request. IEEE Trans. On Comm. vol.47 (July 1999) : 953-955.
11. T.C. Bell, J.G. Cleary and I.H. Witten. Text Compression. Englewood Cliffs, NJ: Prentice Hall, 1990.

บทความทางวิชาการที่ได้รับการเผยแพร่

1. Joint Error Detection and Arithmetic Coding, Proceedings of Electrical Engineering Conference (EECON26), November 2003, Thailand.

2. Joint Error Detection using Arithmetic Coding, Proceedings of 2003 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS2003), December 2003, Japan.

3. Embedded Error Detection in Arithmetic Coding Using Markers, Proceedings of International Conference on Information Technology : Coding and Computing (ITCC2004), April 2004, Nevada, USA.



Joint Error Detection and Arithmetic Coding

Somphop Chokchaitam and Prasit Teekaput
Department of Electrical Engineering Chulalongkorn University
Phayathai Road, Bangkok 10330, Thailand
E-mail: Somphop.C@student.chula.ac.th , Prasit.T@chula.ac.th
Tel (662) 218-6904 Fax (662) 218-6912

Abstract

In this paper, joint error detection arithmetic coding is present along with related theories and simulations results. The proposed coding has its application on the system with feed back error correction with low Bit Error Rate (BER) such as Automatic Repeat Request (ARQ) system. This paper shows how to archived better compression ratio but still has high probability of detecting errors.

Keywords: Joint-Source-Channel Coding, Arithmetic Coding, Error Detection Capability

1. Introduction

Arithmetic Coding and another compression methods such as Huffman Coding are widely use in multimedia compression. Because the channel has band limit the large multimedia data need to be compressed before send it through the channel. The most significant drawback of using data compression is if a single bit error occurs, it will severely damage the decoded data. Then the Error Control Coding (ECC) needs to be applied if data compression is used. The conventional Error Detection Capability (EDC) such as Cyclic Redundancy Check (CRC) did not give enough information to determine which parts the errors occur.

There are some methods already suggested such as the method proposed by C. Boyd [1]. The method is to use the reduction factor to reduce the range use to contain the used symbol if the decoder decodes a symbol out of that range then an error has occurs. This method is similar to the method proposed by J. Sayir [2] which uses an artificial symbol to occupy the gap between the source probabilities interval by using rescaling factor. Another method proposed by G.F. Elmasry [3] done by putting the source symbol into block of source then insert a marker to check for errors.

The method proposed in this paper use lesser overhead redundancy to detect the errors. The idea of this method is to use the properties of arithmetic codes to determine the errors. The method use in this paper is very similar to G.F. Elmasry's method. The different is the proposed method use adaptive arithmetic coding with adaptive markers which we consider from current most frequent symbol. By using current most frequent symbol results in a better compression ratio.

The proposed coding can determine how many information bits needed to retransmit to correct the errors by adding a small amount of redundancy into the conventional arithmetic code. This makes the proposed code suitable to use with the system with feed back error correction capability.

This paper presents the following sections: section 2 provides the basic idea of arithmetic coding and how to do error detection. Section 3 presents details on the proposed code how to implement and how it works. Section 4 provides the analysis result on added overhead redundancy and how to determine the length of information bits needed to retransmit. Section 5 presents the simulations on selected sources. And finally conclusions on proposed code draw in section 6.

2. Arithmetic Coding and Error Detection

Arithmetic Coding is a way to do a progressive compression. Arithmetic Coding generates Variable Length Codes (VLC) by divide range of probabilities. Each time the is contained only in upper or lower half then we scale up the range and send 1 or 0 represent upper and lower half and so on. If the range that represents the symbol is wide then the information bits we need to send is smaller. In situation which source is distributed with skewed probability arithmetic coding can perform better than Huffman coding because arithmetic coding can use less than 1 bit to represent a symbol.

To do error detection on arithmetic code we can easily add a cyclic redundancy check to the end of the data by doing this very small amount of redundancy will added to the data. But when an error occurs there is no clue about the location that the error occurs so to correct the data we have to retransmit the whole data. Instead of using above method we can partition the coded data into block of code and add a CRC to each block. With this alternate method we can detect corrupted data on every block. From this reason data needed to retransmit is lesser but introduce a lot more redundancy to the data. The redundancy introduce in this method came from the partition process which introduce some waste bit at the end of each block and also from CRC applied to each block.

3. Proposed Coding

In this proposed method instead of using one of two method discussed above we do the error detection similar to the method in [3].The method is perform by block the source sequence to block code length k symbols then code the k symbol using arithmetic coding then simply add markers to the end of the block. Continue doing the process until the last block with less than k symbols instead of using marker now we use end-

of-file (EOF) symbol. On the decoder side we can decoder and check the marker if we did not get the markers at their locations then we know that error has occurs.

The strategy used in this paper is use the most frequent symbol as a marker. The reason to use the most frequent symbol as a marker is because the most appear symbol use lesser bit to represent then use this symbol will give the best compression ratio than use another symbol. According to [3] using particular character strategy the probability of misdetection of a marker is defined by.

$$P_{(mis)} = P(s_n). \quad (1)$$

where s_n is a symbol use as marker. From equation (1) we can see that the higher probability of the symbol use as marker raise the probability of misdetection of a marker. Using the most frequent symbol will raise the probability of misdetection of a marker to maximum. And that is the reason not to use the most frequent symbol. But if the most frequent symbol is use in the skewed probability condition we can add more markers into the source sequence at the same amount of redundancy add to the code words. We can do the coding by using 2 of the following strategies

1) Lower the block length k will help reduce the probability of misdetection of a marker when decoded l block to

$$P_{(mis)} = P^l(s_n). \quad (2)$$

Where l is the number of locations checked. From equation (2) we can see that the more markers checked the lower probability of misdetection of a marker. But more bits will have to retransmit to correct the data.

2) Use n markers at each location where n > 1 then we have to check for the marker n time after each k symbol decoded. And we now reduce the probability of misdetection of a marker to

$$P_{(mis)} = P^n(s_n). \quad (3)$$

From equation (2) and (3) we can see that they result on the same probability of misdetection of a marker if l and n are equal and use the same marker.

Both of these strategies have their weakness. The first strategy uses lesser redundancy per block but may take longer time to detect errors due to high probability of misdetection of a marker. The second strategy will lower the probability of misdetection of a maker but add more redundancy to each block of k symbols. When the first strategy is used we need to determine the number of block decoded before we can detect errors then we can predict the number of information bits need to retransmission.

The other problem of using this coding scheme is the last block which has no marker adds. If there is an error occurs in the last codeword we can not detect the error in the codeword. We can prevent this situation by code a marker after EOF symbol.

The processing steps of the above proposed encoder are listed below.

3.1 Encoder

1. Encode *k* symbols using arithmetic coding.
2. Find the most frequent symbol appeared from source.
3. Encode the most frequent symbol. (Encode n times if using the second strategy).
4. Repeat steps 1-3 until source symbols are depleted.
5. Encode the End of File (EOF) artificial symbol to tell the arithmetic decoder to stop decode symbol.

3.2 Decoder

1. Decode *k* symbols.
2. Find the most frequent symbol appeared on the decoder.
3. Decode the most frequent symbol from compressed data.
4. Compare the most frequent symbol found on the decoder with one from the compressed data. The retransmission is required If they are not the same symbol.
5. Repeat the step 1- 4 until End of File symbol is found.

The encoder and decoder describes above is the proposed coding using adaptive arithmetic coding. Using adaptive model instead of using static model we need to find the most frequent symbol after code every k symbols. This will cost more complexity than use static model but give better compression ratio as stated in [4]. The most frequent symbol use as adaptive marker adapt every k symbols coded also help archived the better compression ratio.

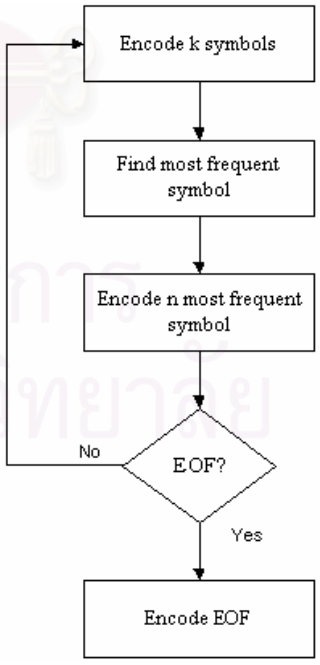


Figure1. Encoder Diagram

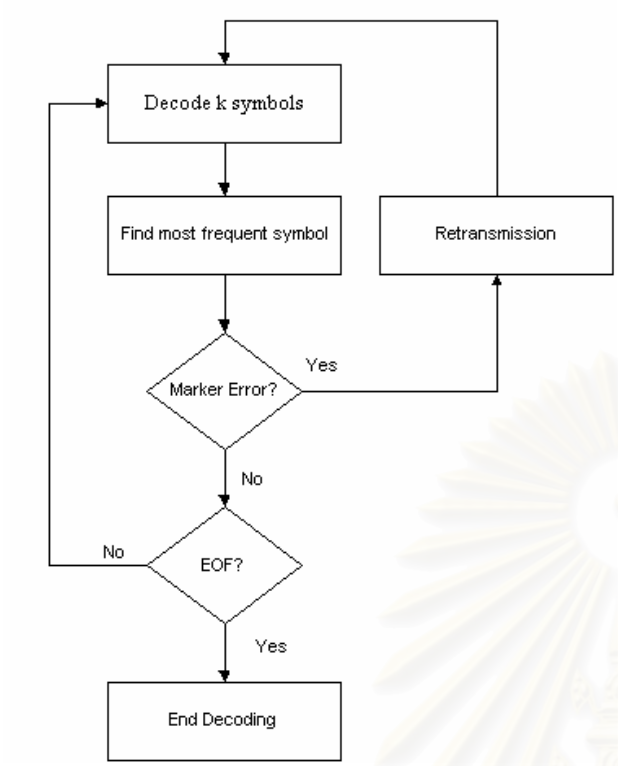


Figure2. Decoder Diagram

4. Theoretical Analysis

4.1 Compressed File Expansion

Consider an i.i.d. (independent identically distributed) source with the source symbols $S = \{s_1, s_2, ..., s_n\}$ and their correspondence probabilities $p(s_i) = \{p_1, p_2, ..., p_n\}$. We can consider the entropy of this source as

$$H(p) = -\sum_{i=1}^{l} p_i \log p_i . \quad (4)$$

where l is the size of the source. The probability p_i of symbol s_i is $N(s_i)/L$, where $N(s_i)$ is the total number of time symbol s_i appears in the file, and L is the size of the file. After code the source symbol using proposed coding. The new probability of occurrence of each symbol in the symbol not use as a marker is

$$p_i' = \frac{N(s_i)}{L\left(1+\frac{n}{k}\right)} = \frac{p_i}{1+\frac{n}{k}}. \quad (5)$$

And if the first strategy is use the equation become

$$p_i' = \frac{N(s_i)}{L\left(1+\frac{1}{k}\right)} = \frac{p_i}{1+\frac{1}{k}}. \quad (6)$$

For the marker s_n , the new probability is

$$p_n' = \frac{N(s_n)+\frac{nL}{k}}{L\left(1+\frac{n}{k}\right)} = \frac{p_n+\frac{n}{k}}{1+\frac{n}{k}}. \quad (8)$$

And for the marker s_n using first strategy, the new probability is

$$p_n' = \frac{N(s_n)+\frac{L}{k}}{L\left(1+\frac{1}{k}\right)} = \frac{p_n+\frac{1}{k}}{1+\frac{1}{k}}. \quad (7)$$

where n is the number of total time the markers code into codeword. The change in probabilities of the symbols changes the average information. The new average information, dynamic entropy $H'(p)$ can be calculated from equation (5), (6), (7) and (8) above.

Follow the instructions in [3] we now get the percentage of file expansion, Δ_f . For the first strategy the percentage of file expansion is

$$\Delta_f = \frac{1}{H(p)}\left[\left(1+\frac{1}{k}\right)\log\left(1+\frac{1}{k}\right) + p_n \log p_n - \left(p_n+\frac{1}{k}\right)\log\left(p_n+\frac{1}{k}\right)\right]. \quad (9)$$

And for the second strategy the percentage of file expansion is

$$\Delta_f = \frac{1}{H(p)}\left[\left(1+\frac{n}{k}\right)\log\left(1+\frac{n}{k}\right) + p_n \log p_n - \left(p_n+\frac{n}{k}\right)\log\left(p_n+\frac{n}{k}\right)\right]. \quad (10)$$

4.2 Information bits retransmission

When the errors have been detected in this coding scheme to correct the error the new information bits needed to retransmit. Assume that the errors have been correctly detected. The amount of information bits need to retransmit can predict from.

$$I_{retransmit} = -(n+k) \times \sum_{i=1}^{l} p'(s_i) \log p'(s_i) \quad (11)$$

which is the dynamic entropy of the source multiply by the number of symbols. Most errors occur within the corrupted block can be correct by retransmit this amount of data again.

5. Simulation

In this simulation we use the text source named book1, paper1 and progc which available from [5] and using adaptive arithmetic coding as described in [6] to code the source symbols. The sources named book1,

paper1 and progc has 768,771 bytes, 53,161bytes and 39,611 bytes respectively. All of them were code using both first and second strategy.

The simulation results show the relative between the length of block and compression ratio which we can easily see that more redundancy has been introduce when we use more markers. The second simulation also confirms the results from first simulation.

The compression ratio shown in the simulations results is defined by

$$\text{Compression ratio} = \frac{L'}{L} \qquad (12)$$

where L' is the size of file after encoded by the encoder and L is the size of the original file.

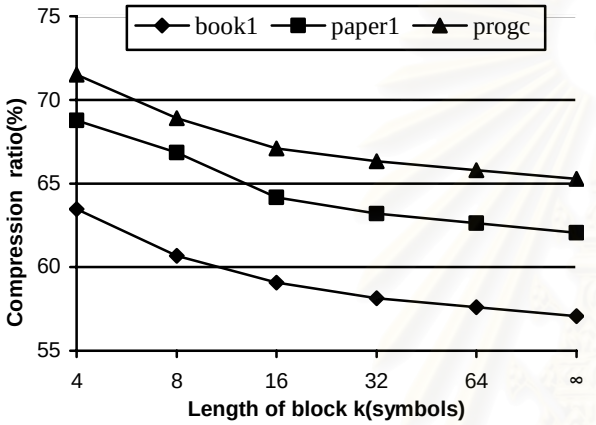


Figure3. Compression ratio vs. length of block using first strategy

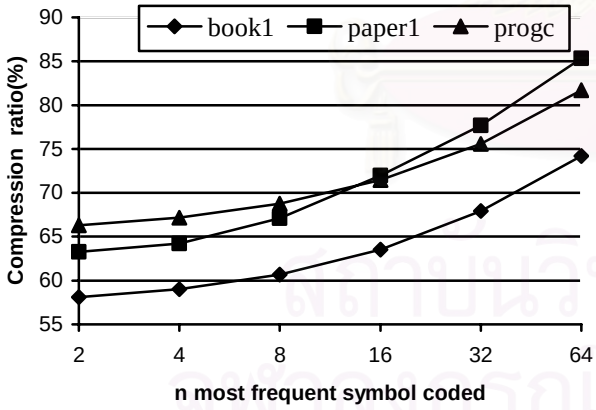


Figure4. Compression ratio vs. n most frequent symbol coded using second strategy and k = 64

From figure3. and figure4. we can see that the strategies not only affect how fast we can detect errors but also affect the compression ratio as seen on result of paper1 and progc. The curve of progc raise higher than paper1 because it depends on symbols distribution of source. This effect can eliminate by using static model instead of adaptive model but will lower the compression ratio effectiveness.

Another source not shown in figure3. and figure4. have the similar trend and lead to the same conclusions.

The overhead complexity charges when encode and decode proposed code is depending on the number of marker use. In our implementation for 4, 8, 16, 32, and 64 symbols before add a marker is about 5%, 3%, 2%, 1%, and 0.6% respectively.

6. Conclusions

The coding scheme we proposed use marker to determine error. The markers introduce a small amount of redundancy and also use the same complexity as use to code one symbol. The codes can detect error faster if more marker is used which results in shorter retransmission length of corrupted data in the cost of more overhead complexity and redundancy added to proposed code.

References

[1] C. Boyd, J.G. Cleary, S.A. Irvine, I. Rinsma-Melchert and I.H. Witten, “Integrating Error Detection into Arithmetic Coding”, IEEE Trans. On Comm., vol.45, no.1, pp.1-3, Jan 1997

[2] J. Sayir, “Arithmetic Coding for Noisy Channels”, IEEE ITW 1999, pp.69-71, June 1999

[3] G.F. Elmasry, “Joint Lossless-Source and Channel Coding Using Automatic Repeat Request”, IEEE Trans. On Comm., vol. 47, pp.953-955, July 1999

[4] K.M.S. Soyjaudah and S. Ramsamy, “A comparative study of context free models of arithmetic coding” EUROCON'2001, vol. 2, pp.428-431, July 2001

[5] T.C. Bell, J.G. Cleary and I.H Witten, Text Compression, Prentice-Hall, Inc., 1990

[6] I.H Witten, J.G. Cleary and R. Neal, “Arithmetic coding for data compression”, Comm. ACM, no.6, pp.520-540, June 1987

Joint Error Detection using Arithmetic Coding

Somphop Chokchaitam and Prasit Teekaput
Department of Electrical Engineering Chulalongkorn University
Phayathai Road, Bangkok 10330, Thailand
E-mail: Somphop.C@student.chula.ac.th , Prasit.T@chula.ac.th
Tel (662) 218-6904 Fax (662) 218-6912

Abstract: A technique to do error detection using small amount of extra redundancy and arithmetic code's properties is described .The proposed technique is shown by related theorem and practical implementation to prove that it can use as described.

1. Introduction

Arithmetic coding is one of the most popular approaches to generate variable length codes. Arithmetic coding is especially useful when deals with small alphabets sources such as binary sources and alphabets with highly skewed probabilities. It is also a very useful approach, for various reasons, when the modeling and coding aspects of lossless compression are to be kept separated. Arithmetic coding also gains better compression ratio than Huffman coding in this case because variable length code generated by Huffman coding can't go lower than 1 bit to represent a symbol while that is not the problem for arithmetic coding .

Arithmetic coding is widely used on multimedia transmission (mostly use as the entropy coder of image and video codec). But arithmetic codes are weak against error as many source coders are. Even a single error occurs on the bit stream but the later decode symbols are not correctly decoded because the error will spread through the rest of the bit stream and result on generated corrupted data. This is the reason why if source data are compressed the error control coding is needed to be applied

The point of using error control coding is how to correct any of these possible bits to make them usable or at least detect that an error is happened. So we can request for retransmission. One method is to employ any of available error control encoding but that is not a good idea if there's another way that uses lesser overhead redundancy or complexity in some applications.

There are several researchers studying for new methods which combined the properties of source coding with some techniques to make them possible to do error detection. The method suggested by C. Boyd is to use the reduction factor and use dummy symbol if a dummy symbol is decompressed. Then the compressed data are regarded as erroneousness which can use for detection of any single bit error [1].After that S.H. Cho and R. Kohno combined that method with VF Arithmetic Coding (VFAC) [2]. And the other approach to do error detection using internal state of VF Arithmetic Coding is introduced by H. Chen which can do some burst error correction. [3].

The new proposed code reduces the complexities use to error detection by using arithmetic code's properties to determine error. The time before an error is detected is also better than Boyd's suggestion [1] at the same compression ratio.

2. Basic Idea

The idea of arithmetic coding is to represent the source sequence in the form of sub interval that use the finite precision to represent the interval in order to be able to do progressive transmission. On the decoder side, the same procedure will process to mimics the encoder in order to decode the sequence. If a sub interval is corrupted by an error, the rest of compressed data can not decode correctly.

To integrate the error detection into arithmetic coding, we need some overhead redundancy to determine that an error is occurred. In this paper the method is to check a symbol (next time this symbol called marked symbol) every time n symbols have been decoded. So any symbol can be use as marked symbol because we know that after n symbols decoded it followed by a marked symbol. If the symbol decode after n symbols are not marked symbol we can said that the last $n+1$ symbols are corrupted. Now we get a new question. Which symbol should we use? The answer is clear to be the current largest cumulative frequency symbol because we need the smallest amount of overhead redundancy. Using the current largest cumulative frequency symbol makes more and more skewed probabilities each marked symbol added. This results on better compression ratio than using another symbol.

How accurate is it that errors will be detected? And how long before they are? The error detection capability is depending on what type of errors are and the model used. In this paper we are experimental with a stochastic account of expected error's behavior.

3. Implementation

The method using to add the redundancy here is to add a marked symbol after each n symbols coded into compressed data, the symbol with the current largest cumulative frequency is used as marked symbol. On decoder side we easily compare the largest cumulative frequency symbol appear on the decoder with the inserted one. If they are not the same symbol so an error occurred on the last b bits used for decode last $n+1$ symbols.

The implementation steps of the proposed encoder are listed below

3.1 Encoder

1. Encode n symbols using arithmetic coding.
2. Find the largest cumulative frequency symbol appeared from source.
3. Encode the largest cumulative frequency symbol.
4. Repeat step 1-3 until source symbols are depleted.
5. Encode the End of File (EOF) symbol to tell the arithmetic decoder to stop decode symbol.

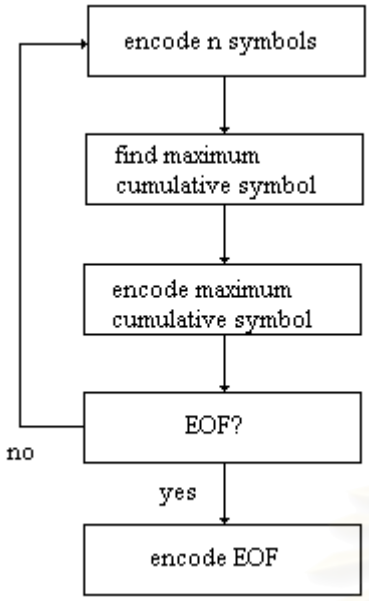


Figure1. Encoder Diagram

3.2 Decoder

1. Decode n symbols
2. Find the largest cumulative frequency symbol appeared on the decoder.
3. Decode the largest cumulative frequency symbol from compressed data.
4. Compare the largest cumulative frequency symbol found on the decoder with one from the compressed data. The retransmission is required If they are not the same symbol.
5. Repeat the step 1-4 until End of File symbol is found.

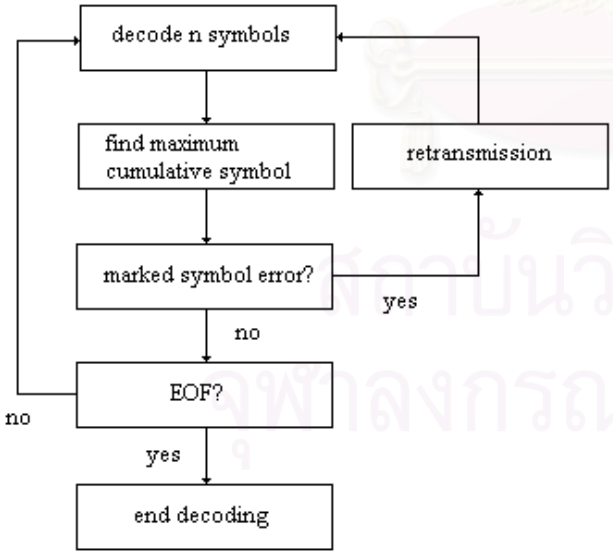


Figure2. Decoder Diagram

4. Implementation Issues

There are several questions to be asked when implementation the proposed encoder is happened. Here are the answers to some questions.

4.1 Choosing the number of symbol before adds a marked symbol

The number of symbol before adds a marked symbol can determine as how frequency we use marked symbol or how much overhead redundancy added to the compressed data. More marked symbol yields faster to detect an error because we check the decoded data every time after completed n symbols decoding. This value is depending on what applications used.

4.2 How many bits should retransmit if an error occurred?

Let we determine the case that single error occurs. In this case, the maximum number of corrupted decoded symbols is $n+1$ symbols. So the maximum bit that can always correct this part is the length of variable length code using to represent the (lowest cumulative frequency symbol $\times n$) + length of variable length code using to represent the largest cumulative frequency symbol bits which can determined from the current probability model.

4.3 How accurate the detection of single errors?

In many applications which the most common kind of error is an isolated single-bit inversion This type of error will be detected because the interval use to decode marked symbol will changed and wrong symbol should decoded in most case.

4.4 How to protect the End of File symbol?

It is necessary to wait for some bits to be processed before an error is detected. This result the last few bits of any compressed data are particularly vulnerable to undetected errors. A special End of File symbol is the last symbol processed during encoding, and many errors can be detected.

Sometime the errors will cause the compressed data to decode as End of File symbol, this result in truncates the data without apparent error. We can protect this by encoding the End of File symbol twice or using another symbol with low probability which will prevent this from unwanted happening.

5. Proposed Code's Properties

Proposed code consists of the message symbol and for each n symbol the largest cumulative frequency symbol is additive to use as a marked symbol. As we see in figure3.

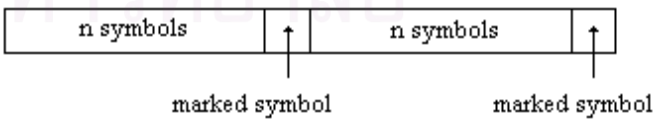


Figure3. Organization of code

The proposed code gives the following properties.

1. Marked symbol gives lesser redundancy each time it added to the message symbol.
2. Using larger n symbols before add a marked symbol reduces overhead redundancy but increases time before an error is detected.

3. If an error is occurred, the length of bit the error lies within and use to request for retransmission can predict from the length of variable length code using to represent lowest cumulative frequency symbol ($l_{\max}$) and the largest cumulative frequency symbol ($l_{\min}$). If assume that errors occur lie within the length of marked symbol to the next marked symbol which is $n+1$ symbols, and the n transmitted symbol has length $l_{\max}$ and a marked symbol with length $l_{\min}$. Now we can predict the length of bit should request for retransmission is $(l_{\max} \times n) + l_{\min}$ bits.

6. Simulation Results

In this simulation we use the text source named book1, paper1 and progc which available from [4] and using adaptive arithmetic coding as described in [5] to code the source symbols.

The simulation is shown that the relative between numbers of used marked symbol and compression ratio which we can easily see that more redundancy has been introduce when we use more marked symbol.

Another source which not shown in figure4. and table 1. have the similar trend and lead to the same conclusion.

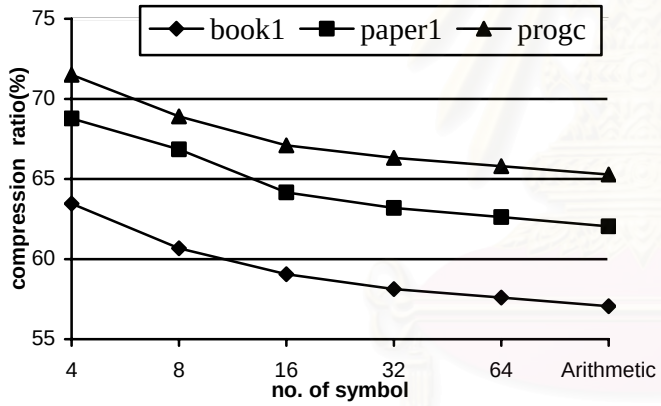


Figure4. Compression ratio vs. no. of symbol before adds a marked symbol

The simulation is also shown to confirm that when increase the use of marked symbol the error can be detected faster as expected. This is shown in Table1.

The predicted number in Table1.is the same as we calculate using its property. The real number is count from the correct compressed data from the first error occur to the marked symbol. We got the results in Table1. from adding random generated 30,000 errors to the compressed data, and use the proposed code to determine the number of bit that should retransmit.

Some strange results in Table1. we get due to the different distribution of symbol on each source. Because some symbols are more occur in some parts than the rest parts of the source.

Table1.
Average number of bits should retransmit

No. of symbol	book1	paper1	progc
4 Predicted Real	56.3 36	41.8 29	36.4 32
8 Predicted Real	62.3 54	50.6 46	40.6 36
16 Predicted Real	78.6 59	70.4 60	52.3 47
32 Predicted Real	145.3 100	126.5 98	90.7 78
64 Predicted Real	240.7 120	170.5 108	165.5 98

From Table1. we can assume that the predicted number will always enough to correct the corrupted part if the errors did not spread more than one of the block source i.e. $n+1$ symbols. Because the number of bit we use as required retransmission bits is the upper bound of possible retransmission for one block of source.

The overhead complexity charge when encode and decode proposed code is depending on the number of marked symbol use. In our implementation for 4, 8, 16, 32, and 64 symbols before add a marked symbol is about 5%, 3%, 2%, 1%, and 0.6% respectively.

7. Conclusions

The coding scheme we proposed use marked symbol to determine error. The marked symbol introduces a small amount of redundancy and also uses the same complexity as use to code one symbol. The code can detect error faster if more marked symbol is used which results in shorter retransmission length of corrupted data in the cost of more overhead complexity and redundancy added to proposed code. The proposed code is better than the conventional standard checksum because it contains the information enough to predict the length of bit that should request for retransmission.

References

[1] C. Boyd, J.G. Cleary, S.A. Irvine, I. Rinsma-Melchert and I.H. Witten, "Integrating Error Detection into Arithmetic Coding", IEEE Trans. on Comm., vol.45, no.1, pp.1-3, Jan 1997.

[2] S.H Cho and R. Kohno, "VF Arithmetic Coding with Error Detecting Capability", Proc. Symp. On Inform. Theory & its Application, pp.278-281, Oct. 1998.

[3] H. Chen, "Joint Error Detection and VF Arithmetic Coding", IEEE Int. Conf. on Comm., vol.9, pp. 2763-2767, 2001.

[4] T.C. Bell, J.G. Cleary and I.H Witten, *Text Compression*, Prentice-Hall, Inc., 1990

[5] I.H Witten, J.G. Cleary and R. Neal, "Arithmetic coding for data compression", Comm. ACM, no.6, pp.520-540, June 1987.

Embedded Error Detection in Arithmetic Coding using Markers

Somphop Chokchaitam and Prasit Teekaput
Department of Electrical Engineering Chulalongkorn University
Phayathai Road, Bangkok 10330, Thailand
E-mail: Somphop.C@student.chula.ac.th, Prasit.T@chula.ac.th

Abstract

This paper provides a joint source channel coding method based on arithmetic coding. The proposed method gives arithmetic coding error detection capability by using its error propagation property and markers. This paper provides theoretical analysis and an experiment with text files.

1. Introduction

Many today's applications are based on multimedia information and transmission. The multimedia data is usually too large to transmit in its original form so the multimedia coding is applied to the data to achieve smaller data. The multimedia coding methods consist of many coding tools including source coding. The source coding has its role to generate the variable length code which can consider as lossless data compression. Arithmetic coding is employed at this stage in the international standards such as JBIG, JPEG2000 and MPEG-4. Arithmetic coding and most of source coding methods have the same major draw back, the error propagation property. The error propagation can damage the whole data after an error has occurred to the compressed data. This is reasonable for adding an error control coding to the system after the source is encoded with source encoder. If the system uses the conventional way to add error detection capability then it use more complexity and redundancy than a joint source channel coding scheme.

There are some methods already suggested such as the method proposed by C. Boyd [1]. The method is use the reduction factor to reduce the range use to represent the symbol if the decoder decodes a symbol out of that range then at least an error has occurred. This method is similar to the method described by J. Sayir [2] which uses artificial symbols to occupy the gap between the source probabilities interval by using rescaling factor if the artificial symbols has been decoded that an error occurred. Another method proposed by G.F. Elmasry [3] performs by placing the source symbol into blocks of source then insert a marker to check for errors.

The method proposed in this paper use smaller overhead redundancy to detect the errors. The idea of this method is to use the error propagation property of arithmetic codes by adding markers to each block of source to determine the errors. The method use in this paper is very similar to Elmasry's method. The different is the proposed method uses adaptive arithmetic coding with adaptive markers which we consider from current most frequent symbol. By using current most frequent symbol as marker results in a better compression ratio. The proposed coding can also determine how many information bits have to retransmit in order to correct the errors this makes the proposed coding suitable to use with the systems with feed back error correction capability.

2. Coding Method

The proposed method in this paper is very similar to the method in [3] but instead of using one of the strategies presented in that paper we now dynamically choose the marker from the appearing frequency from the source. In this paper the most frequent symbol appear in the source is used because if we choose the marker which adds lowest redundancy to the compressed data then we can employ more of them to do a faster error detection or more accurate error detection each time we run through the marked point. The method performs by encode the block of k symbols using arithmetic coding then search for the most frequent symbol to use as marker and simply add markers to the end of each block. Continue doing the same process for each block until the last block with less than k symbols instead of using markers now we use end-of-file (EOF) symbol followed by markers in order to tell the arithmetic decoder to stop and verify that the EOF symbol is not an error decoded symbol. On the decoder side we can decode the same way as we did in the encoder, decode k symbols then check for each marker. If we did not get the markers at their locations then we know that one or more error has occurred before the location that the markers have been checked then the system requests for retransmission to correct the data. Continue the decoding process like this until the EOF symbol is decoded then stop the decoder.

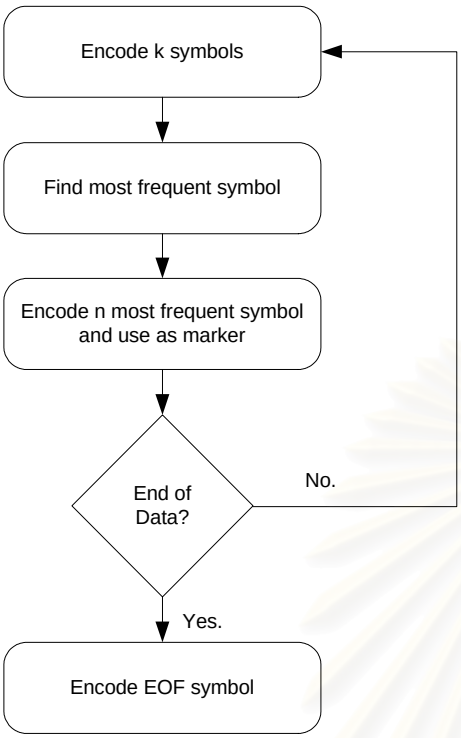


Figure 1. Encoder Diagram

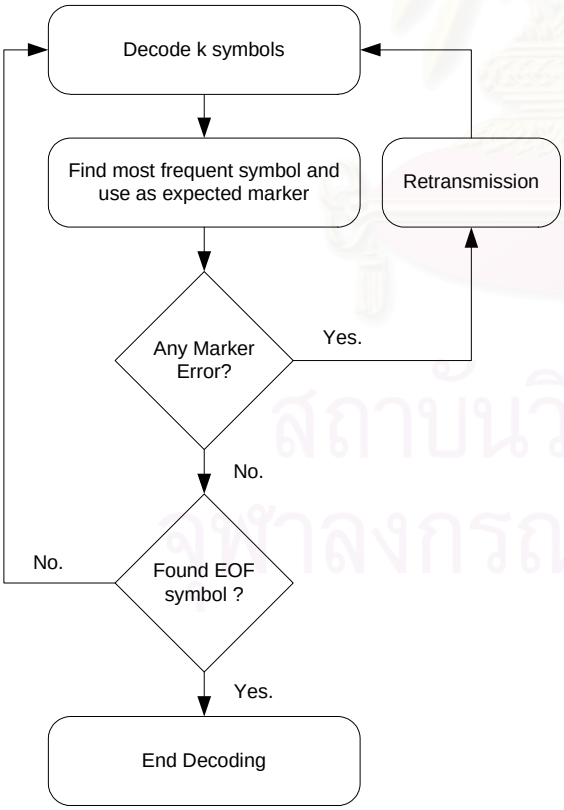


Figure 2. Decoder Diagram

The encoder and decoder described in Figure 1. and Figure 2. are the proposed method using adaptive arithmetic coding and adaptive marker. By using adaptive model instead of using static model we need to find the most frequent symbol to use as marker after encoded each block k symbols. This will cost more complexity than using static model but give better compression ratio as stated in [4]. The most frequent symbol uses as adaptive marker adapts to the new most frequent symbol for each block also helps archived the better compression ratio.

3. Theoretical Analysis

This section provides theoretical analysis results on misdetection probability, how the proposed coding affects the compression ratio and how many information bits need to retransmit if the data need to be correct. This provides simple way to determine how many redundancies added for the desired misdetection probability and easy to estimate the retransmission bits when the proposed coding is use with automatic repeat request system.

3.1. Misdetection Probability

According to [3] using particular character strategy the misdetection probability of a marker is defined by

$$P_{(mis)} = P(s_n). \quad (1)$$

Where s_n is the symbol use as marker. From equation (1) we can see that the higher probability of the symbol use as marker raise the misdetection probability of a marker. Using the most frequent symbol will raise the misdetection probability of a marker to maximum. And that is the reason not to use the most frequent symbol as marker. But if the most frequent symbol is used we can add more markers to the compressed data.

Assume that an error has occurred to the compressed data each marker has the misdetection probability as equation (1). After decode m blocks each block contain n markers then we can rewrite the misdetection probability equation to

$$P_{(mis)} = P^{m \times n}(s_n). \quad (2)$$

From equation (2) we can see that the more times that the markers have been checked yield the lower the misdetection probability of a marker. To reduce the misdetection probability we can do the coding by using the following strategies.

1) Use only one marker and lower the block length k. This will reduce the misdetection probability of a marker after decoded m blocks to

$$P_{(mis)} = P^{m}(s_n). \quad (3)$$

2) Use n markers at each marker location where $n >$ 1 then we have to check for the marker n times after decoded each block of k symbols. And now we reduce the misdetection probability to

$$P_{(mis)} = P^n(s_n). \quad (4)$$

Both strategies have their strength and weakness. The first strategy uses lesser redundancy per block but may not accurate in detecting errors due to high misdetection probability at each block. The second strategy has better misdetection probability but add more redundancy to each block of k symbols. Both strategies can combine to make the proposed coding more suitable for many kinds of applications.

3.2. Compressed Size Expansion

Considers an independent identically distributed source with the finite source symbols $S = \{s_1, s_2, ..., s_n\}$ and their probabilities $p(s_i) = \{p_1, p_2, ..., p_n\}$. We can consider the entropy of this source as

$$H(p) = -\sum_{i=1}^{l} p_i \log p_i. \quad (5)$$

Where l is the total number of the symbol use by the source. The occurrence probability p_i of symbol s_i is equal to $N(s_i)/L$, where $N(s_i)$ is the total time symbol s_i appears and L is the total symbols appear in the source. After encode the source symbol using proposed coding, the new occurrence probability of each symbol in the symbol not use as a marker is

$$p'_i = \frac{N(s_i)}{L\left(1+\frac{n}{k}\right)} = \frac{p_i}{1+\frac{n}{k}}. \quad (6)$$

For the symbol use as marker s_j the new probability is

$$p'_j = \frac{N(s_j)+\frac{nL}{k}}{L\left(1+\frac{n}{k}\right)} = \frac{p_j+\frac{n}{k}}{1+\frac{n}{k}}. \quad (7)$$

Where n is the number of times the markers encoded into each block and k is the number of symbols encoded in each block. The changes in probabilities of the symbols change the average information. The new average information, dynamic entropy $H'(p)$ can be calculated by using p'_i and p'_j from equation (6) and (7) above. Follow the instructions in [3] we now get the percentage of size expansion Δ_f

$$\Delta_f = \frac{1}{H(p)}\left[\left(1+\frac{n}{k}\right)\log\left(1+\frac{n}{k}\right) + p_j \log p_j - \left(p_j+\frac{n}{k}\right)\log\left(p_j+\frac{n}{k}\right)\right]. \quad (8)$$

From above equation we can see that if we use less block length k, more number of markers per block n or less probability of marker p_j. This results on more size expansion of the compressed data. In the other hand this offers us more accurate to detect the errors.

3.3. Information Bits Retransmit

When the errors have been detected in this coding scheme to correct the errors then the information bits have to retransmit. Assume that the errors have been correctly detected within one block. The amount of information bits have to retransmit can be estimate from

$$I_{retransmit} = -(n+k) \times \sum_{i=1}^{l} p'_i \log p'_i \quad (9)$$

which is the dynamic entropy of the source multiply by the number of symbols. Most errors occur within the corrupted block can be correct by retransmit this amount of data.

4. Experiments

In the following experiments we use the Calgary Corpus data set which available from [5] and perform proposed coding based on adaptive arithmetic coding as described in [6] to encode the source symbols. To see difference between vary n and k then we perform two experiment as followed.

In the first experiment all of them were encoded using proposed coding with the number of markers per block n and vary the block length k. The results from the first experiment are shown in table 1. And the second experiment performs by fixed the block length k and vary the number of markers per block n. The results from the second experiment are shown in table 2.

Table 1. First experiment results

Name	Block length k at n = 1					
	4	8	16	32	64	∞
bib	6.073	5.748	5.430	5.346	5.301	5.255
book1	5.082	4.856	4.722	4.648	4.609	4.568
book2	5.569	5.089	4.947	4.869	4.828	4.783
geo	6.029	5.860	5.762	5.711	5.682	5.654
news	5.726	5.492	5.353	5.275	5.234	5.192
obj1	6.246	6.081	5.986	5.927	5.901	5.868

Name	Block length k at n = 1					
	4	8	16	32	64	∞
obj2	6.512	6.292	6.160	6.084	6.044	6.001
paper1	5.511	5.360	5.139	5.061	5.017	4.971
paper2	5.177	4.968	4.805	4.727	4.686	4.642
paper3	5.280	5.147	4.895	4.812	4.769	4.724
paper4	5.613	5.205	4.973	4.896	4.856	4.813
paper5	5.571	5.348	5.213	5.140	5.100	5.058
paper6	5.749	5.269	5.131	5.061	5.021	4.978
pic	1.221	1.195	1.182	1.175	1.172	1.168
progc	5.719	5.501	5.374	5.303	5.266	5.226
progl	5.474	5.063	4.911	4.832	4.794	4.753
progp	5.589	5.152	5.013	4.955	4.924	4.893
trans	6.352	5.823	5.667	5.581	5.532	5.481

Table 2. Second experiment results

Name	Number of markers per block n at k = 256					
	2	4	8	16	32	64
bib	5.278	5.301	5.346	5.429	5.578	5.826
book1	4.589	4.609	4.648	4.722	4.856	5.082
book2	4.806	4.828	4.868	4.947	5.088	5.324
geo	5.668	5.682	5.711	5.762	5.860	6.028
news	5.213	5.234	5.275	5.353	5.492	5.726
obj1	5.885	5.900	5.927	5.985	6.079	6.243
obj2	6.025	6.045	6.084	6.160	6.296	6.528
paper1	4.993	5.016	5.061	5.136	5.280	5.511
paper2	4.664	4.685	4.727	4.804	4.965	5.448
paper3	4.747	4.769	4.812	4.893	5.200	5.517
paper4	4.835	4.854	4.893	4.964	5.103	5.338
paper5	5.079	5.100	5.140	5.217	5.427	5.913
paper6	5.001	5.021	5.060	5.130	5.267	5.492
pic	1.170	1.172	1.175	1.183	1.195	1.221
progc	5.246	5.266	5.303	5.374	5.500	5.717
progl	4.773	4.794	4.832	4.910	5.059	5.473
progp	4.908	4.924	4.954	5.012	5.127	5.316
trans	5.508	5.532	5.581	5.666	5.820	6.074

The experiments results shown in Table 1. and Table 2. are the compression ratio in bits per byte. In Table 1. we vary the block length k to 4, 8, 16, 32, 64 and ∞ and fixed the number of markers per block n to 1. The results at $k = \infty$ is the results from arithmetic coding without any markers. In Table 2. we fixed the block length k to 256 then vary the number of markers n to 2, 4, 8, 16, 32 and 64. From the results of both experiments we can easily see that more redundancy has been introduced when we add more markers to the compressed data. These results confirm the compress size expansion analyzed in previous section. From Table 1. and Table 2. we can see that the strategies not only affect how we detect errors but also affect how the data is compressed as we can see from the results. The compression ratio of the same source and same amount of markers are sometimes different when use different set of parameters. For example if we set $k = 4$ and $n = 1$ this must the compression ratio equal to when we set $k = 256$ and $n = 64$ but we can see that some results are different because we are using adaptive model then the order of symbol occurrence take effects. This effect can eliminates by using static model instead of adaptive model but will reduce the effective of overall compression ratio.

5. Conclusion

The proposed coding uses markers to determine errors. The marker added to each block can use as error detection because arithmetic coding has error propagation property if the data is damage then the data after the corrupted part will also incorrectly decode. Each marker introduces a small amount of redundancy and also uses the same complexity as use to encode one symbol. The codes can detect errors more accurate if more marker is used which results in shorter retransmission length of corrupted data in the cost of more overhead complexity and redundancy added to proposed code.

6. Acknowledgement

The authors wish to express their gratitude to The Cooperative Project for Research and Development between Electrical Engineering Department and Private Sector for financial support on our research.

7. References

[1] C. Boyd, J.G. Cleary, S.A. Irvine, I. Rinsma-Melchert and I.H. Witten, “Integrating Error Detection into Arithmetic Coding”, IEEE Trans. On Comm., vol.45, no.1, pp.1-3, Jan 1997.

[2] J. Sayir, “Arithmetic Coding for Noisy Channels”, IEEE ITW 1999, pp. 69-71, June 1999.

[3] G.F. Elmasry, “Joint Lossless-Source and Channel Coding Using Automatic Repeat Request”, IEEE Trans. On Comm., vol. 47, pp. 953-955, July 1999.

[4] K.M.S. Soyjaudah and S. Ramsamy, “A comparative study of context free models of arithmetic coding”, EUROCON'2001, vol. 2, pp. 428-431, July 2001.

[5] T.C. Bell, J.G. Cleary and I.H Witten, *Text Compression*, Prentice-Hall Inc., 1990.

[6] I.H Witten, J.G. Cleary and R. Neal, “Arithmetic coding for data compression”, Comm. ACM, no.6, pp. 520-540, June 1987.

ประวัติผู้เขียนวิทยานิพนธ์

นายสมภพ โชคชัยธรรม เกิดวันที่ 7 พฤษภาคม พ.ศ. 2524 ที่จังหวัดชลบุรี เข้าศึกษาในหลักสูตรวิศวกรรมศาสตรบัณฑิต คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์ ในปีการศึกษา 2541 สำเร็จการศึกษาปริญญาวิศวกรรมศาสตรบัณฑิต ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์ ในปีการศึกษา 2544 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต ที่ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2545