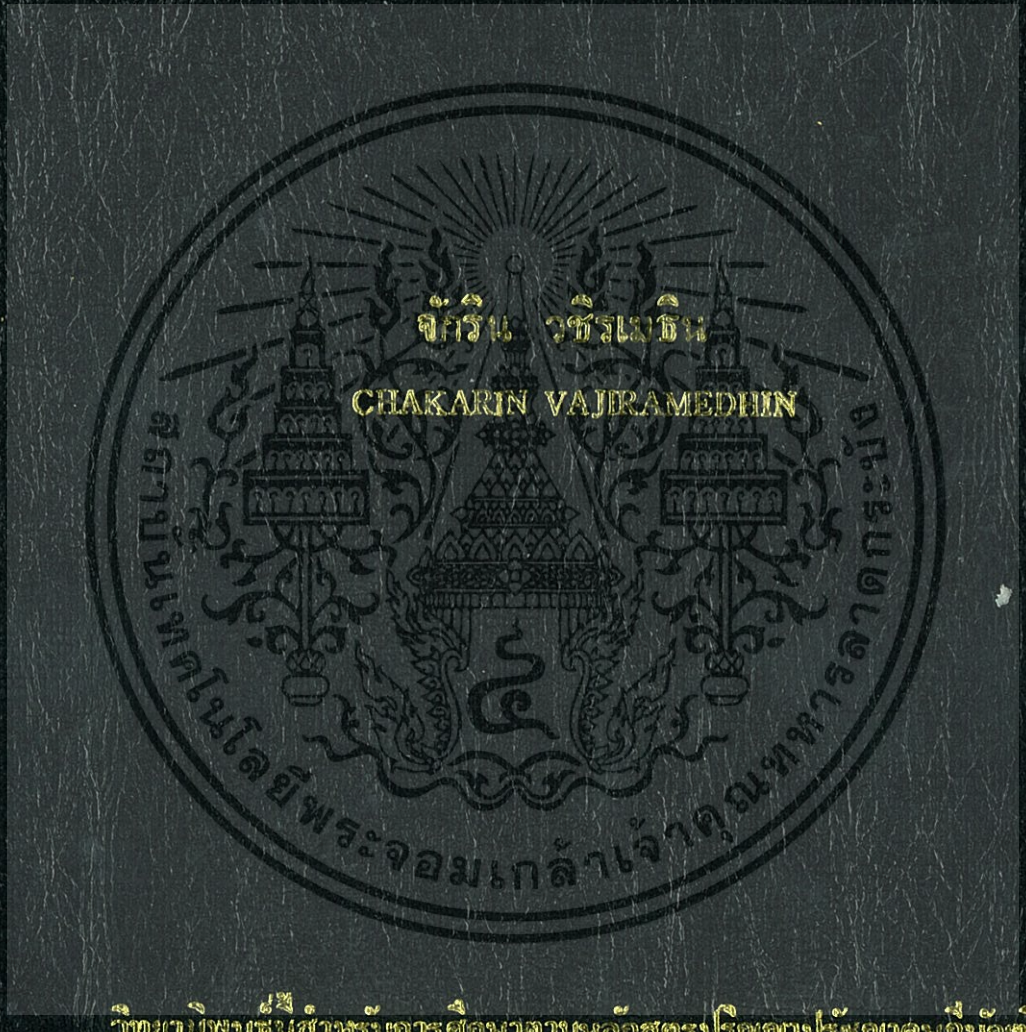
การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดโดยใช้วิธีการอีบีพีเจ

AN EBPA APPROACH TO FREQUENT CLOSED ITEMSET MINING



จักริน วชิรเมธิน

CHAKARIN VAJRAMEDHIN

วิทยานิพนธ์นี้สำหรับการศึกษาตามหลักสูตรปริญญาปรัชญาดุษฎีบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2556

KMITL-2013-SC-D-002-007

การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดโดยใช้วิธีการอีบีพีเอ

AN EBPA APPROACH TO FREQUENT CLOSED ITEMSET MINING

จักริน วชิรเมธิน

CHAKARIN VAJIRAMEDHIN

วิทยานิพนธ์นี้สำหรับการศึกษาตามหลักสูตรปริญญาปรัชญาดุษฎีบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2556

KMITL – 2013 – SC – D – 002 - 007

AN EBPA APPROACH TO FREQUENT CLOSED ITEMSET MINING

CHAKARIN VAJIRAMEDHIN

A THESIS SUBMITTED IN FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2013

KMITL-2013-SC-D-002-007

คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์	การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดโดยใช้วิธีการอีบีพีเอ An Ebpa Approach to Frequent Closed Itemset Mining
นักศึกษา	นายจักริน วชิรเมธิน
รหัสประจำตัว	49062951
ปริญญา	ปรัชญาดุษฎีบัณฑิต
สาขาวิชา	วิทยาการคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.ดร.จีรพร วีระพันธุ์
อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม	-

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ผศ.ดร.นันทิกา	เบญจเทพานันท์	
รศ.ดร.วีระ	บุญจริง	
ผศ.ดร.ศรัณย์	อินทโกสุม	
ดร.อนันตพร	หรรษคุณาฒัย	
ดร.เฉลิมศักดิ์	เลิศวงศ์เสถียร	
รศ.ดร.จีรพร	วีระพันธุ์	

วัน / เดือน/ ปี ที่สอบ 12 มีนาคม พ.ศ. 2556 เวลา 13.00 – 16.00 น.
สถานที่สอบ ณ ห้อง 216 ชั้น 2 อาคารจุฬาภรณวลัยลักษณ์ 1

คณะวิทยาศาสตร์รับรองแล้ว

(รองศาสตราจารย์ ดร.ดุษณี ธนะบริพัฒน์)
คณบดีคณะวิทยาศาสตร์

วันที่ ๒ **เดือน** มีนาคม **พ.ศ.** ๒๕๕๖

หัวข้อวิทยานิพนธ์	การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดโดยใช้วิธีการอีบีพีเอ
นักศึกษา	นายจักริน วชิรเมธิน
รหัสประจำตัว	49062951
ปริญญา	ปรัชญาดุษฎีบัณฑิต
สาขาวิชา	วิทยาการคอมพิวเตอร์
พ.ศ.	2556
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.ดร.จีรพร วีระพันธุ์

บทคัดย่อ

การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดจากฐานข้อมูลขนาดใหญ่โดยตรงนั้นเป็นการทำงานที่ไม่มีประสิทธิภาพทั้งในเรื่องของการใช้หน่วยความจำและเวลาในการประมวลผล ในช่วงหลายปีที่ผ่านมานี้จึงมีการนำเสนอโครงสร้างข้อมูลออกมาหลากหลายวิธี ซึ่งแต่ละโครงสร้างข้อมูลมีทั้งข้อดีและข้อด้อยแตกต่างกันออกไป โดยล่าสุดนั้นโครงสร้างข้อมูลแบบรวมที่เกิดจากการรวมเอาข้อดีของโครงสร้างข้อมูลอาร์เรย์ลิตส์ โครงสร้างข้อมูลบิทแมบ และโครงสร้างข้อมูลพรีฟิคซ์ทรีได้ถูกนำเสนอขึ้นเพื่อช่วยลดเวลาในการประมวลผลและพื้นที่จัดเก็บข้อมูล แต่อย่างไรก็ตามโครงสร้างข้อมูลนี้มีขั้นตอนในการจัดเรียงและการรวมแถวข้อมูลที่ซ้ำซ้อนกันที่ยังใช้เวลาในการทำงานนาน ดังนั้นในงานวิจัยชิ้นนี้จึงได้นำเสนอโครงสร้างข้อมูลอีบีพีเอสำหรับการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด โดยใช้โครงสร้างข้อมูลแบบรวมเป็นต้นแบบในการปรับปรุงประสิทธิภาพ โดยปรับปรุงในส่วนของการเข้าถึงข้อมูลระหว่างโหนดแม่และโหนดลูกด้วยความเร็วเท่ากับ $O(1)$ และตัดขั้นตอนการจัดเรียงและการรวมแถวข้อมูลที่ซ้ำซ้อนออก ซึ่งช่วยเพิ่มความเร็วในการประมวลผลได้ดีขึ้น จากผลการทดลองกับฐานข้อมูลสามชุด (*connect*, *chess* และ *pumsb**) พบว่าวิธีการในงานวิจัยฉบับนี้มีประสิทธิภาพการทำงานที่ดีกว่าวิธีการต้นแบบที่ใช้โครงสร้างข้อมูลแบบรวมในทุกฐานข้อมูล

Thesis Title	An EBPA Approach to Frequent Closed Itemset Mining
Student	Chakarin Vajiramedhin
Student ID.	49062951
Degree	Doctor of Philosophy
Program	Computer Science
Year	2013
Thesis Advisor	Assoc. Prof. Dr. Jeeraporn Werapun

ABSTRACT

The process of closed itemset mining from a large transaction database directly often leads to inefficient cost (time and space). In the past ten years, many data structures were proposed to process the frequent closed itemset mining in efficient time, while each data structure has its own advantages and disadvantages. Lately, a collaboration of array, bitmap, and prefix tree was proposed to gain the advantages of those basic data structures. In such collaboration, the prefix tree and array list are joined to reduce the computing time of the FCIM and the bitmap array is used to represent data in each node of that structure for saving space over the node of the (original) prefix tree, which requires extra space for parent-child pointers and its hashing. However, extra sorting and merging repeated transactions are required before constructing the prefix tree (from leaf nodes to the root), corresponding to the transaction order. In this paper, we propose the improved collaboration data structure, called EBPA (the Efficient Bitmap-Prefix tree-Array), especially the construction of the prefix-tree array with the efficient (parent-child) access in *O(1)* without extra space (for pointers and hashing) and no extra sorting transactions before constructing the prefix tree. In system performance evaluation, experimental results from three available datasets (*connect*, *chess*, and *pumsb**) showed that the response time of our efficient BPA-based FCIM mining outperforms over that of the existing collaboration-based FCIM approach.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดีจากคำชี้แนะและความอนุเคราะห์ของอาจารย์ที่ปรึกษา รศ .ดร .จีรพร วีระพันธุ์ ซึ่งคอยให้คำแนะนำอย่างสม่ำเสมอจนกระทั่งวิทยานิพนธ์ฉบับนี้มีความสมบูรณ์มากยิ่งขึ้น อีกทั้งยังคอยดูแลเอาใจใส่พร้อมทั้งอบรมสั่งสอนจนกระทั้งข้าพเจ้าสำเร็จการศึกษา จึงขอขอบคุณเป็นอย่างสูงไว้ ณ โอกาสนี้

ขอขอบคุณสำนักงานคณะกรรมการการอุดมศึกษา และคณะบริหารศาสตร์ มหาวิทยาลัยอุบลราชธานี ที่สนับสนุนและจัดสรรทุนการศึกษาให้กับข้าพเจ้าจนกระทั่งสำเร็จการศึกษา

ขอขอบคุณเพื่อน ๆ นักศึกษาปริญญาเอกทุกท่านที่คอยให้คำปรึกษา และคอยให้กำลังใจด้วยดีเสมอมา

และสุดท้ายขอขอบคุณครอบครัว ภรรยาและลูก ๆ ที่เข้าใจและให้โอกาสในการศึกษาครั้งนี้ อีกทั้งยังคอยดูแลเอาใจใส่ด้วยดีเสมอมา ซึ่งเป็นกำลังใจให้ข้าพเจ้าสามารถฝ่าฟันอุปสรรคต่าง ๆ มาได้จนกระทั้งสำเร็จการศึกษา

จักริน วชิรเมธิน

สารบัญ

หน้า

สารบัญ

หน้า

สารบัญตาราง

ตาราง **หน้า**

สารบัญรูปภาพ

รูปภาพ **หน้า**

รูปภาพ **หน้า**

บทที่ 1
บทนำ

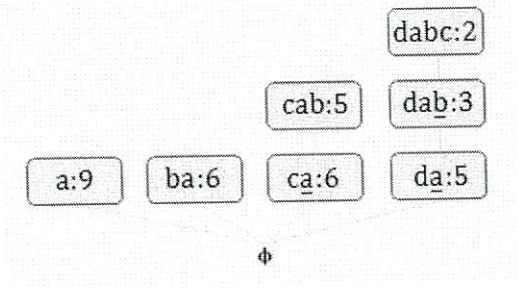
1.1 ความสำคัญและความเป็นมาของปัญหา

การสืบค้นกลุ่มรายการที่เกิดบ่อย (Frequent Itemsets Mining) เป็นเทคนิคสำคัญที่งานพื้นฐานของการทำเหมืองข้อมูลนำมาประยุกต์ใช้ เช่น การจำแนกประเภทข้อมูล (Classifiers) การหากฎความสัมพันธ์ของข้อมูล (Association Rules) เป็นต้น โดยวิธีการดังกล่าวนี้จะค้นหากลุ่มรายการ (Pattern หรือ Itemset) ที่ซ่อนอยู่ในฐานข้อมูลพร้อมทั้งพิจารณาค่าความถี่ (Frequency) ในการเกิดกลุ่มรายการนั้น ๆ จากนั้นจึงนำเอากลุ่มรายการที่ได้ไปจำแนกประเภทหรือหาความสัมพันธ์หรือนำไปประยุกต์ใช้ในงานด้านต่าง ๆ ต่อไป ซึ่งมีนักวิจัยหลาย ๆ ท่านนำเสนองานวิจัยที่เกี่ยวข้องออกมาหลากหลายวิธีการ [1]-[13] โดยแต่ละวิธีก็มีประสิทธิภาพแตกต่างกันไป ปัญหาหลักที่มีผลกระทบต่อประสิทธิภาพของวิธีการเหล่านี้ก็คือ ปริมาณข้อมูลดิบที่นำมาประมวลผล ซึ่งนับวันจะเพิ่มขึ้นเรื่อย ๆ โดยเฉพาะอย่างยิ่งในปัจจุบันนี้เป็นยุคสารสนเทศที่ข้อมูลมีความสำคัญเป็นอย่างมาก หน่วยงานต่าง ๆ ทั้งภาครัฐและภาคธุรกิจมักจะเก็บสะสมข้อมูลเอาไว้เพื่อที่จะนำเอาไปใช้งาน เช่น ใช้ประกอบการวิเคราะห์ การวางแผน และการบริหารจัดการ ยิ่งมีข้อมูลดิบมากเพียงใดก็ยิ่งมีโอกาสที่จะได้สารสนเทศที่มีประโยชน์มากขึ้นเท่านั้น ด้วยเหตุนี้เองจึงมีการเก็บสะสมข้อมูลดิบเพิ่มมากขึ้นเรื่อย ๆ นอกจากนี้โดยธรรมชาติของการจัดเก็บข้อมูลแล้วก็มักจะมีการจัดเก็บข้อมูลที่เกี่ยวข้องเอาไว้ด้วยเพื่อให้มีข้อมูลครบถ้วนมากพอสำหรับการวิเคราะห์ เช่น ข้อมูลลูกค้า นอกจากชื่อและนามสกุลแล้ว ยังจัดเก็บข้อมูลอื่น ๆ เช่น อายุ เพศ การศึกษาหรือถ้าเป็นข้อมูลสินค้านอกจากรหัสและชื่อสินค้าแล้วอาจจะเก็บข้อมูลประเภทสินค้า ยี่ห้อสินค้า รุ่นสินค้าเพิ่มเติม ซึ่งส่งผลกระทบต่อประสิทธิภาพการทำงานโดยตรง โดยเฉพาะอย่างยิ่งในกรณีที่ทำการประมวลผลกับข้อมูลดิบจำนวนมาก ๆ และกำหนดค่าสนับสนุนขั้นต่ำเอาไว้ต่ำ ๆ อาจจะได้กลุ่มรายการที่เป็นผลลัพธ์จำนวนมหาศาล ซึ่งใช้เวลาในการประมวลผลที่ยาวนานและใช้พื้นที่ในการจัดเก็บข้อมูลในหน่วยความจำในขณะประมวลผลจำนวนมากจนทำให้หน่วยความจำหลักไม่สามารถรองรับได้

ดังนั้น จึงมีนักวิจัยหลาย ๆ ท่านได้หาวิธีการเพื่อแก้ไขปัญหาข้างต้นนี้ โดยเรียกวิธีการนี้ว่าการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด (Frequent Closed Itemset Mining) [14]-[27] (ดังแสดงในรูปที่ 1.1 (ข)) โดยวิธีการนี้จะค้นหาและจัดเก็บเฉพาะกลุ่มรายการที่เกิดบ่อยแบบปิด (Closed Itemset) แทนการค้นหาข้อมูลทั้งหมด กลุ่มรายการที่เกิดบ่อยแบบปิดที่ได้นั้นจะเป็นกลุ่มรายการสูงสุด (Maximum Itemset) ที่อยู่ในคลาสสมมูล (Equivalence Class) เดียวกันและสามารถใช้อธิบายแทนกลุ่มรายการทั้งหมดได้ โดยไม่ทำให้สูญเสียความหมายเดิมของข้อมูลทั้งหมดไป

tid	Item			
	a	b	c	d
0	a	b	c	d
1	a	b	c	d
2	a	b		d
3	a			d
4	a	b	c	
5	a	b	c	
6	a		c	
7	a	b	c	
8	a			d

(ก)



(ข)

รูปที่ 1.1 (ก) ฐานข้อมูลตัวอย่าง (ข) โครงข่ายกลุ่มรายการที่เกิดบ่อยแบบปิดจากฐานข้อมูลตัวอย่างในรูปที่ 1.1 (ก)

ดังนั้น การค้นหาและจัดเก็บเฉพาะกลุ่มรายการที่เกิดบ่อยแบบปิดจึงทำให้ลดปริมาณการใช้พื้นที่หน่วยความจำและลดเวลาในการประมวลผลลงได้เป็นอย่างมาก

ปัจจัยที่ส่งผลกระทบต่อประสิทธิภาพของการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดนั้น นอกจากมีขั้นตอนวิธีการทำงานที่ดีแล้ว โครงสร้างข้อมูล (Data Structure) ก็เป็นอีกหนึ่งปัจจัยสำคัญที่มีผลกระทบต่อประสิทธิภาพการทำงานโดยตรง การออกแบบโครงสร้างข้อมูลที่ดีนั้นจะต้องคำนึงถึงการจัดเก็บข้อมูลในหน่วยความจำหลักในขณะประมวลผลเป็นสำคัญและจะต้องสนับสนุนการทำงานของวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดให้สามารถทำงานได้ง่ายและสะดวกรวดเร็ว โดยเฉพาะอย่างยิ่งในเรื่องของการคำนวณค่าความถี่ ซึ่งเป็นส่วนที่ทำงานหนักที่สุดของการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด [24] หากมีโครงสร้างข้อมูลที่เอื้อต่อการค้นหาความถี่ของกลุ่มรายการก็จะทำให้วิธีการหลักมีประสิทธิภาพมากยิ่งขึ้น ด้วยเหตุนี้นักวิจัยหลาย ๆ ท่านได้นำเสนอโครงสร้างข้อมูลและวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดออกมาหลาย ๆ วิธีการ ([17], [20], [24], [26], [28]) ซึ่งแต่ละโครงสร้างข้อมูลก็มีข้อดีและข้อด้อยแตกต่างกันไป โดยล่าสุดนี้ได้มีการนำเอาข้อดีของแต่ละวิธีการมาใช้ในการออกแบบโครงสร้างข้อมูล โดยเรียกวิธีการนี้ว่าโครงสร้างข้อมูลแบบรวม (Collaboration of array list-bitmap-prefix tree) ซึ่งเกิดจากการนำเอาข้อดีของ 3 โครงสร้างข้อมูลคือ โครงสร้างข้อมูลอาร์เรย์ลิตส์ (Array List) โครงสร้างข้อมูลบิทแมบ (Vertical Bitmap) และโครงสร้างข้อมูลพรีฟิคซ์ทรี (Prefix Tree) มารวมกันเพื่อให้ได้ประสิทธิภาพในการทำงานที่ดีขึ้นทั้งในเรื่องของความเร็วในการทำงานและเรื่องของการจัดเก็บข้อมูลในหน่วยความจำให้ประหยัดพื้นที่ให้ได้มากที่สุด แต่วิธีการดังกล่าวยังมีจุดที่สามารถปรับปรุง เพื่อให้ได้ประสิทธิภาพในการทำงานที่สูงขึ้น โดยเฉพาะอย่างยิ่ง ในขั้นตอนการสร้างโครงสร้างข้อมูลที่ต้องจัดเรียงและรวมแถวข้อมูลที่ซ้ำกันที่ใช้เวลาในการทำงานนานนั้น สามารถนำมาปรับปรุงให้มีประสิทธิภาพมากยิ่งขึ้น

ดังนั้นในงานวิจัยชิ้นนี้จึงได้นำเอาวิธีการดังกล่าวมาเป็นต้นแบบในการปรับปรุงในส่วนที่ยังทำงานได้ไม่เต็มประสิทธิภาพ โดยเน้นในเรื่องของการออกแบบในส่วนของการเข้าถึงโหนดในโครงสร้างข้อมูล ซึ่งใช้การคำนวณค่าดัชนีจากตำแหน่งในโครงสร้างข้อมูลพรีฟิคซ์ทรีแบบสมบูรณ์ (Complete Prefix Tree) โดยไม่จำเป็นต้องสร้างโครงสร้างข้อมูลพรีฟิคซ์ทรีแบบสมบูรณ์และออกแบบในส่วนของการเข้าถึงข้อมูลระหว่างโหนดแม่และโหนดลูกโดยประยุกต์ใช้พรีฟิคซ์อาร์เรย์ ที่ใช้เวลาในการเข้าถึงของแต่ละโหนดเท่ากับ $O(1)$ โดยได้ตัดขั้นตอนการจัดเรียงข้อมูลและการรวมแถวข้อมูลที่ซ้ำซ้อนออก ซึ่งช่วยเพิ่มความเร็วในการประมวลผลและลดพื้นที่ในการจัดเก็บข้อมูลในหน่วยความจำลงเพื่อให้สามารถรองรับการทำงานกับข้อมูลขนาดใหญ่ได้

นอกจากนี้ยังได้ปรับปรุงประสิทธิภาพการทำงานของวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด โดยลดปริมาณการเปรียบเทียบข้อมูลด้วยเทคนิคที่เรียกว่าพรีเทส (pre-test) ซึ่งเป็นขั้นตอนการคัดกรองเอาเฉพาะข้อมูลที่เกี่ยวข้องกับกลุ่มรายการที่เป็นตัวสร้าง (Generator Itemset) ที่จำเป็นจะต้องใช้ในการเปรียบเทียบมาเก็บไว้ในข้อมูลเริ่มต้นเท่านั้น ซึ่งตรงจุดนี้ช่วยลดเวลาการทำงานได้มากยิ่งขึ้น

1.2 วัตถุประสงค์ของงานวิจัย

1. เพื่อพัฒนาโครงสร้างข้อมูลสำหรับการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดที่สนับสนุนให้การคำนวณหาความถี่ของกลุ่มรายการสามารถทำได้อย่างมีประสิทธิภาพ
2. เพื่อพัฒนาวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดร่วมกับโครงสร้างข้อมูลได้อย่างมีประสิทธิภาพ

1.3 ขอบเขตของงานวิจัย

1. ศึกษาโครงสร้างข้อมูลที่เกี่ยวข้องสำหรับการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดและวิเคราะห์โครงสร้างข้อมูลในงานวิจัยที่เกี่ยวข้องเพื่อให้ทราบถึงข้อดีและข้อด้อยของแต่ละวิธี
2. ออกแบบโครงสร้างข้อมูลใหม่เพื่อสนับสนุนวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดให้สามารถคำนวณค่าความถี่ได้สะดวกรวดเร็วยิ่งขึ้น
3. ข้อมูลที่ใช้ในการทดลองคือข้อมูลที่ได้รับการยอมรับและใช้ในการทดลองของงานวิจัยที่เกี่ยวข้องได้แก่ Pumsb*, Chess และ Connect
4. ทำการทดลองโดยประมวลผลเปรียบเทียบเวลาการทำงานกับงานวิจัยที่เกี่ยวข้องในสภาวะแวดล้อมเดียวกัน

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. โครงสร้างข้อมูลอีบีพีเอ (EBPA) ที่พัฒนาขึ้นสนับสนุนให้การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดทำงานได้อย่างมีประสิทธิภาพ

2. โครงสร้างข้อมูลอีบีพีเอที่พัฒนาขึ้นสนับสนุนให้การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดสามารถคำนวณหาความถี่ของกลุ่มรายการได้สะดวกรวดเร็วขึ้น และรองรับข้อมูลดิบที่จะนำมาประมวลผลได้มากขึ้น

3. วิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดที่พัฒนาขึ้นสามารถทำงานร่วมกับโครงสร้างข้อมูลอีบีพีเอได้อย่างมีประสิทธิภาพ โดยช่วยลดเวลาในการประมวลผลและลดพื้นที่หน่วยความจำได้เป็นอย่างดี

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้นำเสนอข้อมูลเบื้องต้นเกี่ยวกับแนวคิด ทฤษฎีและงานวิจัยที่เกี่ยวข้อง โดยเฉพาะอย่างยิ่งข้อมูลเกี่ยวกับการสืบค้นกลุ่มรายการที่เกิดบ่อย การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด และโครงสร้างข้อมูลสำหรับการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด

2.1 แนวคิดและทฤษฎีที่เกี่ยวข้อง

ในยุคสารสนเทศอย่างเช่นปัจจุบันนี้ ข้อมูลถือว่าเป็นสิ่งที่มีความสำคัญเป็นอย่างมาก หน่วยงานต่าง ๆ ทั้งภาครัฐและเอกชนนิยมนำเอาข้อมูลมาใช้ประกอบการวิเคราะห์ วางแผนและบริหารจัดการ โดยเฉพาะอย่างยิ่งในภาคธุรกิจนั้นยิ่งใครมีข้อมูลที่มีประโยชน์อยู่ในมือมากเท่าไหร่ก็ยิ่งได้เปรียบคู่แข่งมากขึ้นเท่านั้น การจะได้ข้อมูลที่มีประโยชน์มาใช้งานนั้นจะต้องนำเอาข้อมูลดิบที่เก็บสะสมเอาไว้ไปผ่านการประมวลผลในหลาย ๆ ขั้นตอน เพื่อให้เกิดเป็นสารสนเทศที่พร้อมสำหรับการนำไปใช้งาน ดังแสดงรูปที่ 2.1 ด้วยเหตุนี้เอง จึงมีวิธีในการกลั่นกรองข้อมูลดิบออกมาหลากหลายวิธีการ ซึ่งวิธีการเหล่านี้อยู่ในขั้นตอนการประมวลผลเพื่อให้ได้สารสนเทศที่สามารถนำไปใช้ประโยชน์ได้นั้นเอง

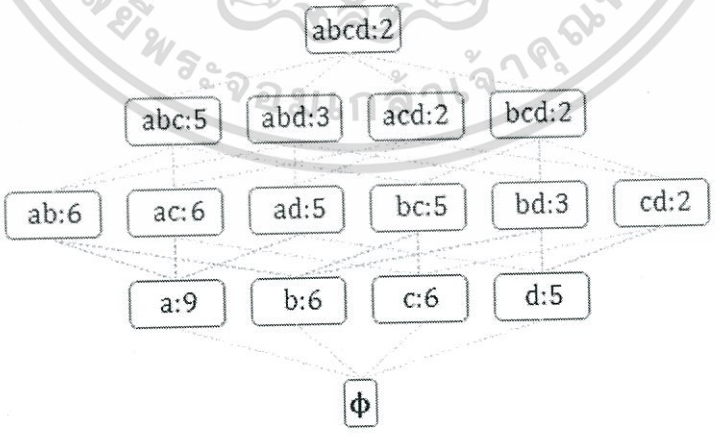
การสืบค้นกลุ่มรายการที่เกิดบ่อย และการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดเป็นเทคนิคสำคัญที่งานพื้นฐานของการทำเหมืองข้อมูลนิยมนำเอาไปประยุกต์ใช้ไม่ว่าจะเป็นการจำแนกประเภทข้อมูลหรือการหากฎความสัมพันธ์ของข้อมูล ซึ่งวิธีการดังกล่าวนี้เป็นขั้นตอนการกลั่นกรองข้อมูลประเภทหนึ่ง โดยทำการประมวลผลเพื่อให้ได้กลุ่มรายการที่เป็นประโยชน์สามารถนำเอาไปใช้งานต่อได้

รูปที่ 2.1 การประมวลผลข้อมูลเพื่อให้ได้สารสนเทศที่เป็นประโยชน์

2.1.1 การสืบค้นกลุ่มรายการที่เกิดบ่อย (Frequent Itemset Mining)

การสืบค้นกลุ่มรายการที่เกิดบ่อยเป็นเทคนิคพื้นฐานของงานด้านเหมืองข้อมูล โดยจะค้นหากลุ่มรายการที่ซ่อนอยู่ในฐานข้อมูลพร้อมทั้งพิจารณาความถี่ในการเกิดกลุ่มรายการนั้น ๆ ด้วย จากนั้นจึงนำเอากลุ่มรายการที่ได้ไปประยุกต์ใช้ในงานด้านต่าง ๆ ต่อไป ค่าความถี่ (Frequency) คือ จำนวนการเกิดขึ้นของข้อมูลในฐานข้อมูล ซึ่งความถี่ในการเกิดข้อมูลเป็นนัยสำคัญต่อการนำไปวิเคราะห์และใช้งานข้อมูล ด้วยเหตุนี้จึงมีนักวิจัยหลาย ๆ ท่านนำเสนองานวิจัยที่เกี่ยวข้องออกมาหลากหลายวิธีการ [1]-[13] โดยแต่ละวิธีการก็มีประสิทธิภาพที่แตกต่างกันไป การสืบค้นกลุ่มรายการที่เกิดบ่อยสามารถค้นหาได้จากนิยามด้านล่างนี้

นิยามที่ 2.1 *กำหนดให้ $D = \{t_1, t_2, t_3, ..., t_j\}$ เป็นฐานข้อมูล โดย $t_1, t_2, t_3, ..., t_i$ เป็นทรานแซคชันของข้อมูล ซึ่ง t_i แทนแถวของทรานแซคชันข้อมูลในฐานข้อมูลที่ประกอบไปด้วยหมายเลขทรานแซคชัน (transaction identifier : tid) และรายการ (item) ข้อมูลที่เรียงลำดับจาก 1 จนถึง k ($i_1, i_2, i_3, ..., i_k$) กำหนดให้ $I = (i_1, i_2, i_3, ..., i_k)$ เป็นเซตของรายการในฐานข้อมูล โดยแต่ละซับเซต (subset) P ของ I เรียกว่า กลุ่มรายการ (itemset) ซึ่งกลุ่มรายการ P ที่ประกอบไปด้วยกลุ่มรายการที่มีจำนวนสมาชิก k รายการ เรียกว่า k-itemset (กลุ่มรายการขนาด k รายการ) โดยจำนวนทรานแซคชันใน D ที่มีกลุ่มรายการ P เรียกว่าค่าสนับสนุน (support) ของ P หรือค่าความถี่ในการเกิดของ P หรือ supp(P) และกำหนดให้ min_supp คือค่าสนับสนุนขั้นต่ำ (minimum support) โดยจะเรียกกลุ่มรายการ P ว่า กลุ่มรายการที่เกิดบ่อย (frequent itemset) ก็ต่อเมื่อกลุ่มรายการ P มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ $supp(P) \geq min\_supp$.*



รูปที่ 2.2 โครงข่ายกลุ่มรายการที่เกิดบ่อยจากฐานข้อมูลตัวอย่างในรูปที่ 1.1 (ก)

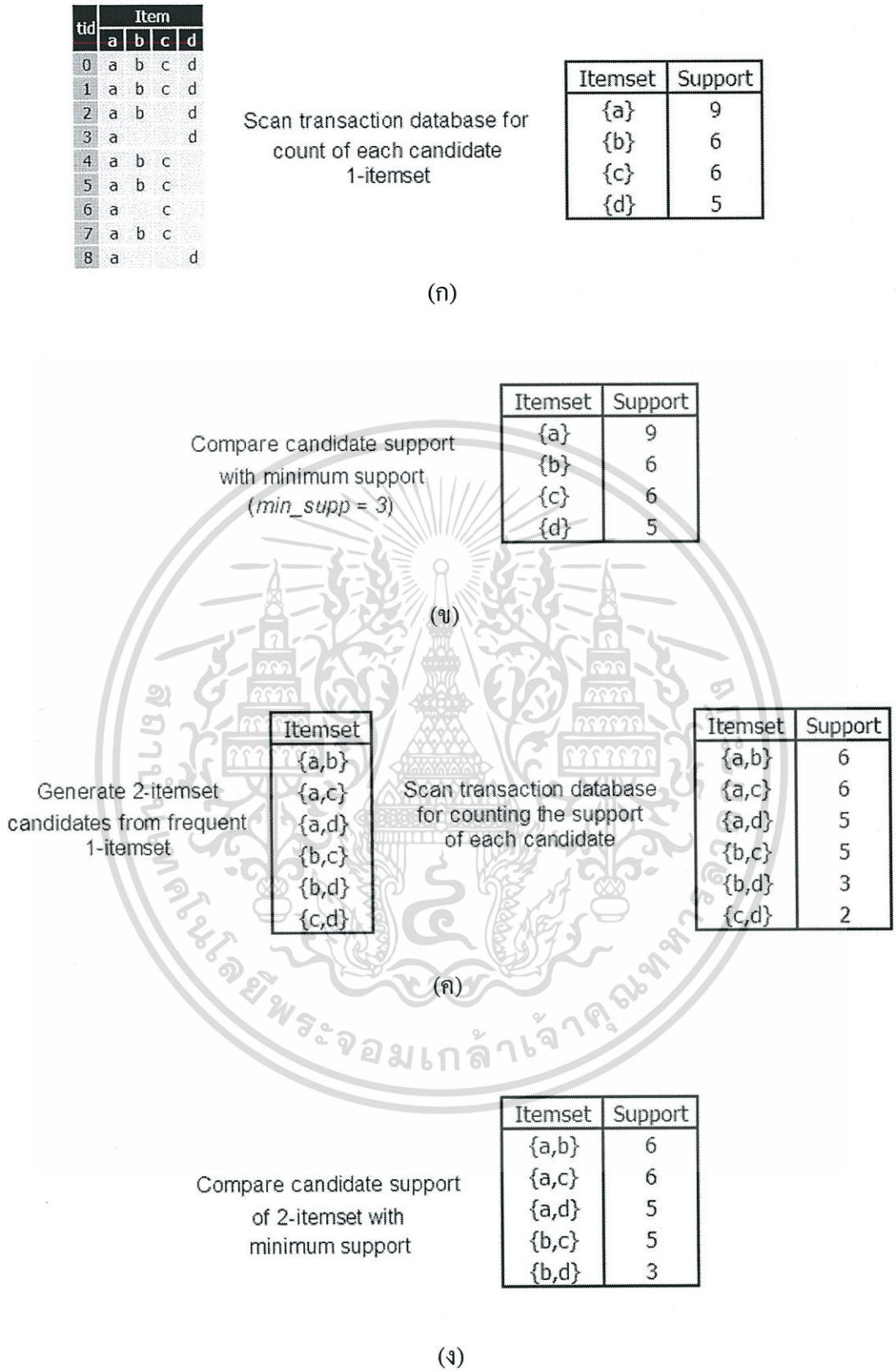
กลุ่มรายการที่ได้จากวิธีการนี้สามารถนำเอาไปประยุกต์ใช้งานได้ในหลากหลายสาขาด้วยกัน เช่น ในร้านสะดวกซื้อสามารถนำเอาข้อมูลไปใช้ในการวิเคราะห์การซื้อสินค้าและการจัดวางสินค้าได้ ยกตัวอย่างเช่น จากโครงข่ายของกลุ่มรายการที่เกิดบ่อยในรูปที่ 2.2 ที่แสดงโครงข่าย (Lattice) ของกลุ่มรายการที่เกิดบ่อยจากข้อมูลตัวอย่างในรูปที่ 1.1 (ก) โดยกำหนดให้ค่าสนับสนุนขั้นต่ำเท่ากับ 1 พบว่า สินค้า *a, b* และ *c* ถูกซื้อรวมกันถึง 5 ครั้งจากรายการซื้อทั้งหมด 9 ครั้ง ดังนั้นหากทางร้านค้าวางแผนการจัดวางสินค้าโดยนำเอาสินค้าทั้ง 3 รายการจัดวางไว้ใกล้กันน่าจะช่วยเพิ่มยอดขายได้มากขึ้นเพราะช่วยเพิ่มความสะดวกให้กับลูกค้ามากขึ้น จากตัวอย่างข้างต้นแสดงให้เห็นถึงประโยชน์ของวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อย ซึ่งสามารถนำเอากลุ่มรายการที่ได้ไปประยุกต์ใช้ประโยชน์ต่อได้ ด้วยเหตุนี้เอง นักวิจัยหลายท่านจึงได้นำเสนอวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยออกมาหลากหลายวิธีการ [1]-[13] ซึ่งจากการศึกษาวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยที่มีประสิทธิภาพได้แก่ วิธีการ Apriori และวิธีการ FP-growth

2.1.1.1 วิธีการ Apriori

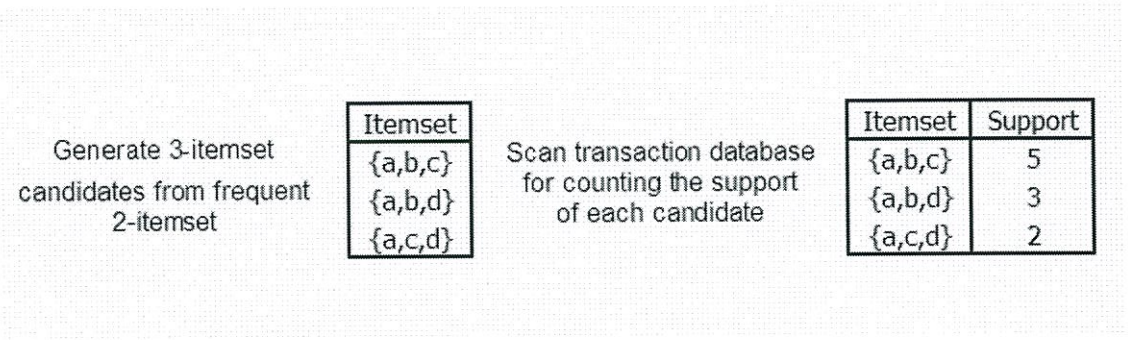
วิธีการ Apriori นำเสนอโดย Rakesh Agrawal และ Ramakrishnan Srikant จาก IBM Almaden Research Center ในปี 1994 [1] ซึ่งเป็นอัลกอริทึมพื้นฐานของการสืบค้นกลุ่มรายการที่เกิดบ่อยจากฐานข้อมูลที่ใช้กันอย่างแพร่หลาย ซึ่งอัลกอริทึม Apriori ทำงานโดยสร้างกลุ่มรายการแคนดิเดต (Candidate Itemsets) ทั้งหมดและทำการค้นหากลุ่มรายการที่มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำที่กำหนดไว้ โดยมีรูปแบบการทำงานแบบล่างขึ้นบน (Bottom up) การทำงานของวิธีการนี้มีขั้นตอนการทำงานหลัก ๆ อยู่สองขั้นตอนคือ (แสดงตัวอย่างในรูปที่ 2.3)

ขั้นตอนที่ 1 สร้างกลุ่มรายการแคนดิเดตทั้งหมดขึ้นมา โดยในรอบแรกจะเริ่มจากการค้นหากลุ่มรายการแคนดิเดตทั้งหมดจากฐานข้อมูลที่เป็นกลุ่มรายการขนาด 1 รายการ ส่วนในรอบถัด ๆ ไปก็จะสร้างกลุ่มรายการแคนดิเดตจากกลุ่มรายการที่ได้จากผลลัพธ์ของการทำงานในรอบก่อนหน้านั้น (*k* รายการ คือกลุ่มรายการที่มีจำนวนสมาชิก *k* รายการ) เมื่อได้กลุ่มรายการมาเรียบร้อยก็จะเข้าสู่การทำงานในขั้นตอนที่ 2 ต่อไป

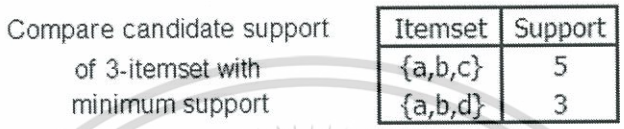
ขั้นตอนที่ 2 สแกนหาความถี่ของกลุ่มรายการแคนดิเดตทั้งหมดในฐานข้อมูลและทำการเปรียบเทียบกลุ่มรายการแคนดิเดตกับค่าสนับสนุนขั้นต่ำที่กำหนดไว้ หากกลุ่มรายการตัวใดมีค่าสนับสนุนขั้นต่ำผ่านเกณฑ์ที่ตั้งเอาไว้ก็จะเรียกว่า กลุ่มรายการที่เกิดบ่อย (Frequent Itemset) จากนั้นก็จะนำผลลัพธ์ที่ได้ไปเป็นกลุ่มรายการตั้งต้นของการทำงานในรอบถัด ๆ ไป โดยจะทำตามวิธีการดังกล่าวไปเรื่อย ๆ จนกว่ากลุ่มรายการแคนดิเดตจะหมด (กลุ่มรายการที่มีจำนวนสมาชิกสูงสุดจำนวน *k* รายการ)



รูปที่ 2.3 ขั้นตอนการสืบค้นกลุ่มรายการที่เกิดบ่อยโดยอัลกอลิทึม Apriori



(จ)



(ฉ)

(ช)

รูปที่ 2.3 ขั้นตอนการสืบค้นกลุ่มรายการที่เกิดบ่อยโดยอัลกอลิทึม Apriori (ต่อ)

จากรูปที่ 2.3 (ก) เริ่มการทำงานจากการสแกนฐานข้อมูลเพื่อหากลุ่มรายการแคนดิเดตโดยเริ่มจากกลุ่มรายการขนาด 1 รายการ ({*a*:9}, {*b*:6}, {*c*:6} และ {*d*:5}) จากฐานข้อมูลในรูปด้านซ้ายมือ จากนั้นก็นำกลุ่มรายการที่ได้ในรูปด้านขวามือไปเปรียบเทียบกับค่าสนับสนุนขั้นต่ำ (สมมุติว่ากำหนดค่าสนับสนุนขั้นต่ำเท่ากับ 3) แสดงในรูปที่ 2.3 (ข) หากกลุ่มรายการใดผ่านเกณฑ์จะถูกเรียกว่า กลุ่มรายการที่เกิดบ่อย และจะเป็นกลุ่มรายการตั้งต้นของรอบต่อไป จากกลุ่มรายการที่ได้ในตัวอย่างนั้นกลุ่มรายการทั้งหมดมีค่าความถี่มากกว่าค่าสนับสนุนขั้นต่ำจึงผ่านเกณฑ์ทั้งหมด จากนั้นวิธีการนี้จะสร้างกลุ่มรายการแคนดิเดตทั้งหมดขนาด 2 รายการขึ้นจากกลุ่มรายการขนาด 1 รายการที่เป็นผลลัพธ์ในรอบแรก และทำการสแกนหาความถี่ในฐานข้อมูล กลุ่มรายการแคนดิเดตในรอบนี้คือ {*ab*:6}, {*ac*:6}, {*ad*:5}, {*bc*:5}, {*bd*:3} และ {*cd*:2} ดังแสดงในรูปที่ 2.3 (ค) จากนั้นก็ทำการเปรียบเทียบกลุ่มรายการแคนดิเดตกับค่าสนับสนุนขั้นต่ำที่กำหนดไว้ ดังแสดงในรูปที่ 2.3 (ง) กลุ่มรายการ {*cd*:2} มีค่าความถี่ต่ำกว่าเกณฑ์ที่ตั้งไว้ จึงไม่ถือว่าเป็นกลุ่มรายการที่เกิดบ่อย

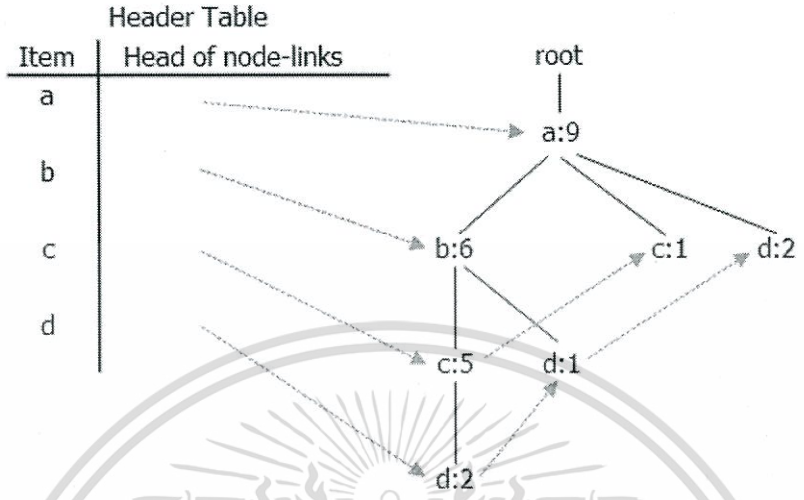
ในรอบที่ 3 ทำการสร้างกลุ่มรายการแคนดิเดตขนาด 3 รายการขึ้นจากกลุ่มรายการขนาด 2 รายการ ซึ่งมีกลุ่มรายการแคนดิเดตทั้งหมด 3 รายการ ดังแสดงในรูปที่ 2.3 (จ) จากนั้นทำการสแกนหาค่าความถี่ของกลุ่มรายการแคนดิเดตในฐานข้อมูลและเปรียบเทียบกับค่าสนับสนุนขั้นต่ำ ซึ่งมีกลุ่มรายการที่ผ่านเกณฑ์เพียง 2 รายการ คือ {*abc*:5} และ {*abd*:3} ดังแสดงในรูปที่ 2.3 (ฉ) ในรอบสุดท้ายทำการสร้างกลุ่มรายการแคนดิเดตขนาด 4 รายการ จากผลลัพธ์ของรอบที่ 3 ซึ่งมีกลุ่มรายการแคนดิเดตเพียง 1 รายการคือ "*abcd*" จากนั้นก็ทำการสแกนหาค่าสนับสนุนและเปรียบเทียบกับค่าสนับสนุนขั้นต่ำ ซึ่งกลุ่มรายการ {*abcd*:2} มีค่าความถี่ขั้นต่ำเพียง 2 เท่านั้นจึงไม่ผ่านเกณฑ์การพิจารณา

วิธีการ Apriori ถือว่าเป็นการทำงานที่ดีกว่าวิธีการหากลุ่มรายการที่เกิดบ่อยก่อนหน้านี้เพียงแต่ว่าหากนำเอาวิธีการนี้ไปทำงานกับฐานข้อมูลขนาดใหญ่ที่มีกลุ่มรายการในปริมาณมากจะต้องใช้เวลาในการทำงานนานและใช้หน่วยความจำในการจัดเก็บข้อมูลจำนวนมาก โดยเฉพาะอย่างยิ่งวิธีการนี้ต้องสร้างกลุ่มรายการแคนดิเดตในทุกรอบการทำงาน ในกรณีที่มีกลุ่มรายการขนาด 1 รายการจำนวนมาก ๆ ก็ยิ่งใช้เวลาในการทำงานมากขึ้นด้วย เช่น ถ้ากลุ่มรายการขนาด 1 รายการเท่ากับ 10^4 วิธีการนี้จะสร้างกลุ่มรายการแคนดิเดตขนาด 2 รายการเท่ากับ 10^7 นอกจากนี้ วิธีการนี้ยังต้องสแกนฐานข้อมูลทั้งหมดเพื่อหาความถี่ของแต่ละตัวในทุกรอบการทำงาน ซึ่งทำให้ต้องใช้เวลาในการทำงานนานขึ้น ด้วยเหตุนี้เองจึงมีนักวิจัยหลาย ๆ คนได้นำเสนอวิธีการทำงานที่ไม่จำเป็นต้องสร้างกลุ่มรายการแคนดิเดตและลดจำนวนการสแกนฐานข้อมูลลง

2.1.1.1 วิธีการ FP-Growth

อัลกอลิทึม Frequent Pattern Growth (FP-Growth) นำเสนอโดย Jiawei Han, Jian Pei และ Yiwen Yin ในปี 2000 [5] เพื่อแก้ปัญหาของวิธีการ Apriori ที่ต้องสร้างกลุ่มรายการแคนดิเดตทั้งหมดก่อนทำการเปรียบเทียบค่าความถี่ และต้องสแกนฐานข้อมูลในทุกรอบการทำงานเพื่อหาความถี่ของแต่ละกลุ่มรายการ วิธีการนี้ถูกออกแบบมาเพื่อหากลุ่มรายการที่เกิดบ่อยโดยไม่จำเป็นต้องสร้างกลุ่มรายการแคนดิเดต โครงสร้างข้อมูลหลักคือโครงสร้างข้อมูลเอฟพี-ทรี (Frequent Pattern Tree : FP-Tree) ซึ่งเป็นโครงสร้างข้อมูลพรีฟิคซ์ทรี โดยใช้ร่วมกับตารางเฮดเดอร์ (Header Table) ซึ่งเป็นตารางที่ทำหน้าที่เก็บลิงค์เชื่อมโยงไปยังกลุ่มรายการขนาด 1 รายการแต่ละตัว เพื่อให้สามารถเข้าถึงกลุ่มรายการแต่ละตัวได้โดยตรงโดยไม่จำเป็นต้องเริ่มจากจุดเริ่มต้นของโครงสร้างข้อมูลเอฟพี-ทรี ดังแสดงในรูปที่ 2.4 ในแต่ละโหนดของโครงสร้างข้อมูลเอฟพี-ทรีจะเก็บชื่อของโหนด (แค่ 1 กลุ่มรายการทำให้ประหยัดพื้นที่หน่วยความจำ) ค่าความถี่ ลิงค์เชื่อมโยงกับโหนดซัฟฟิคซ์ (Suffix) และลิงค์เชื่อมโยงกับโหนดที่เกี่ยวข้อง จึงทำให้วิธีการ FP-Growth สามารถทำงานได้รวดเร็วและประหยัดพื้นที่ในการจัดเก็บข้อมูลมากขึ้น

นอกจากนี้ วิธีการนี้ยังสแกนฐานข้อมูลแค่เพียง 2 รอบเท่านั้น โดยครั้งแรกสแกนเพื่อหากลุ่มรายการขนาด 1 รายการและครั้งที่สองสแกนเพื่อสร้างโครงสร้างข้อมูลเอฟพี-ทรี สำหรับการสืบค้นกลุ่มรายการที่เกิดบ่อยของวิธีการนี้จะใช้วิธี Partition-based และ Divide-and-Conquer



รูปที่ 2.4 โครงสร้างข้อมูลเอฟพี-ทรีแสดงข้อมูลจากฐานข้อมูลตัวอย่างในรูปที่ 1.1 (ก)

ขั้นตอนการสร้างโครงสร้างข้อมูลเอฟพี-ทรี เริ่มจากการสแกนข้อมูลจากฐานข้อมูลเพื่อหากลุ่มรายการที่เกิดบ่อยขนาด 1 รายการที่มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ จากฐานข้อมูลตัวอย่าง สมมุติว่ากำหนดให้ค่าสนับสนุนขั้นต่ำเท่ากับ 3 โดยกลุ่มรายการที่เกิดบ่อยคือ {*a*: 9}, {*b*:6}, {*c*:6}, {*d*:5} จากนั้นจึงสแกนฐานข้อมูลอีกรอบเพื่อจัดเรียงกลุ่มรายการที่เกิดบ่อยในแต่ละแถวตามค่าความถี่จากมากไปหาน้อยพร้อมคัดกรองเอากลุ่มรายการที่ไม่ผ่านค่าความถี่ขั้นต่ำออก ขั้นตอนต่อไปเป็นการสร้างตารางเฮดเดอร์พร้อมทั้งเก็บกลุ่มรายการที่เกิดบ่อยขนาด 1 รายการลงในตาราง ขั้นตอนสุดท้ายของการสร้างโครงสร้างข้อมูลเอฟพี-ทรีคือการอ่านค่าจากฐานข้อมูลที่จัดเรียงเอาไว้แล้วลงในโครงสร้างข้อมูลทีละแถวพร้อมทั้งเชื่อมโยงความสัมพันธ์เข้ากับกลุ่มรายการที่อยู่ในตารางเฮดเดอร์เพื่อความรวดเร็วในการเข้าถึงต่อไป

สำหรับขั้นตอนการสืบค้นกลุ่มรายการที่เกิดบ่อยของวิธีการนี้จะใช้ อัลกอลิทึม FP-Growth โดยประกอบด้วย 3 ขั้นตอนคือ ขั้นตอนที่ 1 สร้างพรีฟิคซ์พาสซัพทรีเพื่อเชื่อมโยงกลุ่มรายการเดียวกันที่อยู่ในแต่ละกิ่งของโครงสร้างเอฟพี-ทรีเพื่อความสะดวกในการเชื่อมโยงโดยเริ่มจากโหนดที่อยู่ด้านลางขึ้นบน จากข้อมูลตัวอย่างจะเริ่มเชื่อมโยงกลุ่มรายการ *d* ที่มีอยู่ตามกิ่งต่าง ๆ เข้าด้วยกัน จากนั้นก็เชื่อมโยงกลุ่มรายการ *c* ที่อยู่ตามกิ่งต่าง ๆ เข้าด้วยกัน ตามมาด้วยกลุ่มรายการ *b* และ *a* ที่มีแค่กลุ่มรายการละ 1 โหนดจึงไม่มีการเชื่อมโยง ผลลัพธ์ที่ได้จากการทำงานในขั้นตอนนี้ก็คือกลุ่มรายการ d = {*a*:9, *b*:6, *c*:5, *d*:2}, {*a*:9, *b*:6, *d*:1}, {*a*:9, *d*:2} กลุ่มรายการ c = {*a*:9, *b*:6, *c*:5}, {*a*:9, *c*:1} กลุ่มรายการ b = {*a*:9, *b*:6} และกลุ่มรายการ a = {*a*:9} ดังแสดงในตารางที่ 2.1

ตารางที่ 2.1 พรีฟิคซ์พาสซัพทรี, คอนดิชันนอลแพทเทินเบส และคอนดิชันนอลทรี

กลุ่มรายการ	พรีฟิคซ์พาสซัพทรี	คอนดิชันนอลแพทเทินเบส	คอนดิชันนอลเอฟพี-ทรี
d	{*a*:9, *b*:6, *c*:5, *d*:2}, {*a*:9, *b*:6, *d*:1}, {*a*:9, *d*:2}	{*abc*:2}, {*ab*:1}, {*a*:2}	{*a*:5}, {*ab*:3}
c	{*a*:9, *b*:6, *c*:5}, {*a*:9, *c*:1}	{*ab*:5}, {*a*:1}	{*a*:6}, {*ab*:5}
b	{*a*:9, *b*:6}	{*a*:6}	{*a*:6}
a	{*a*:9}	Null	Null

ขั้นตอนที่ 2 สร้างคอนดิชันนอลแพทเทินเบซและคอนดิชันนอลเอฟพี-ทรีของกลุ่มรายการจากผลลัพธ์พรีฟิคซ์พาสซัพทรีที่ได้ในขั้นตอนที่ 1 ดังแสดงในตารางที่ 2.1 ซึ่งสร้างขึ้นมาเพื่อใช้สำหรับการสืบค้นกลุ่มรายการที่เกิดบ่อยต่อไป ขั้นตอนที่ 3 เป็นการสร้างกลุ่มรายการที่เกิดบ่อยจากคอนดิชันนอลเอฟพี-ทรี พิจารณาจากข้อมูลที่ได้มาจากขั้นตอนที่ 2 เช่น โหนด *d* มีคอนดิชันนอลเอฟพี-ทรี 2 รายการ คือ {*a*:5} และ {*ab*:3} ซึ่งเป็นกลุ่มรายการที่ผ่านค่าความถี่ขั้นต่ำที่กำหนดไว้ หากนำไปสร้างกลุ่มรายการที่เกิดบ่อยโดยมีโหนด *d* เป็นฐานจะได้กลุ่มรายการที่เกิดบ่อย 4 รายการดังนี้ {*d*:5}, {*ad*:5}, {*bd*:3}, {*abd*:3} ต่อไปหากมีโหนด *c* เป็นฐานโดยมีคอนดิชันนอลเอฟพี-ทรีเป็น {*a*:6}และ {*ab*:6} จะได้กลุ่มรายการที่เกิดบ่อย 4 รายการคือ {*c*:6}, {*ac*:6}, {*bc*:5}, {a*bc*:5} โหนดถัดมา *b* มีคอนดิชันนอลเอฟพี-ทรีเพียงหนึ่งรายการคือ {*a*:6} จะได้กลุ่มรายการที่เกิดบ่อย {*b*:6}, {*ab*:6} และโหนดสุดท้าย *a* มีกลุ่มรายการที่เกิดบ่อยเพียงรายการเดียวคือ {*a*:9}

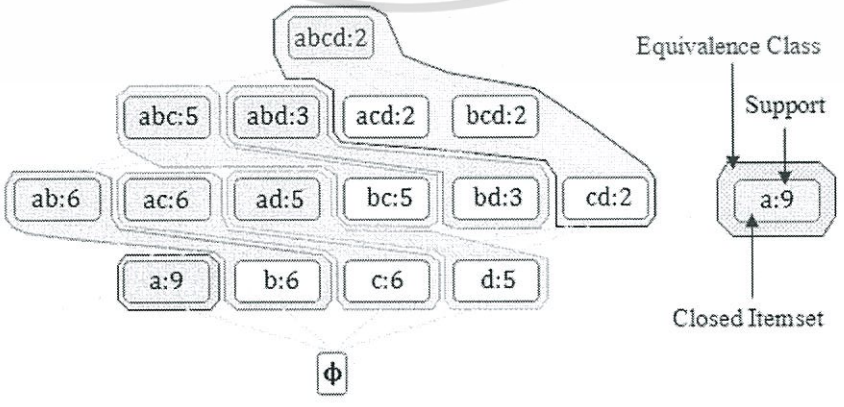
ถึงแม้ว่าการสืบค้นกลุ่มรายการที่เกิดบ่อยด้วยวิธีการต่าง ๆ จะมีประสิทธิภาพ แต่ปัญหาหลักที่มีผลกระทบต่อประสิทธิภาพของวิธีการเหล่านี้ก็คือ ปริมาณข้อมูลดิบที่นำมาประมวลผล ซึ่งนับวันจะเพิ่มขึ้นเรื่อย ๆ โดยปกติแล้วหน่วยงานต่าง ๆ มักจะสะสมข้อมูลไว้ให้มากที่สุด เพราะยิ่งมีข้อมูลดิบมากเพียงใดก็ยิ่งมีโอกาสที่จะได้สารสนเทศที่มีประโยชน์มากขึ้นเท่านั้น ยิ่งไปกว่านั้นโดยธรรมชาติของการจัดเก็บข้อมูลแล้วมักจะมีการจัดเก็บข้อมูลที่เกี่ยวข้องเอาไว้เพื่อใช้ประกอบการวิเคราะห์ให้ครบถ้วนมากยิ่งขึ้น เช่น ข้อมูลลูกค้า (อายุ เพศ การศึกษา) หรือข้อมูลสินค้า (ประเภทสินค้า ยี่ห้อสินค้า รุ่นสินค้า) จึงส่งผลกระทบต่อประสิทธิภาพการทำงานของแต่ละวิธีโดยเฉพาะอย่างยิ่งในกรณีที่ประมวลผลกับข้อมูลดิบจำนวนมาก ๆ และกำหนดค่าสนับสนุนขั้นต่ำเอาไว้ต่ำ ๆ อาจจะได้กลุ่มรายการที่เป็นผลลัพธ์จำนวนมหาศาล อีกทั้งยังใช้เวลาในการประมวลผลที่ยาวนานและใช้พื้นที่ในหน่วยความจำหลักในขณะประมวลผลจำนวนมากจนทำให้หน่วยความจำหลักไม่สามารถรองรับการทำงานดังกล่าวได้

ดังนั้นจึงมีนักวิจัยหลาย ๆ ท่านได้หาวิธีการแก้ปัญหาดังกล่าวนี้ โดยเรียกวิธีการแก้ปัญหานี้ว่า การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด [14]-[27] ซึ่งวิธีการนี้จะค้นหาเฉพาะกลุ่มรายการแบบปิดเพียงเท่านั้น กลุ่มรายการที่เกิดบ่อยแบบปิดที่ได้นั้นเป็นกลุ่มรายการสูงสุดที่อยู่ในคลาสสมมูลที่มีค่าความถี่เท่ากัน และสามารถใช้อธิบายแทนกลุ่มรายการทั้งหมดได้โดยไม่ทำให้สูญเสียความสำคัญของข้อมูลทั้งหมดไป การจัดเก็บเฉพาะกลุ่มรายการที่เกิดบ่อยแบบปิดทำให้สามารถลดปริมาณการใช้หน่วยความจำและเวลาในการประมวลผลลงได้เป็นอย่างมาก

2.1.2 การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด (Frequent Closed Itemset Mining)

การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดเป็นเทคนิคสำคัญที่นิยมนำไปใช้ในหลาย ๆ งานทางด้านเหมืองข้อมูล ซึ่งถูกนำเสนอครั้งแรกโดย Pasquier ในปี 1999 [14] วิธีการนี้เป็นวิธีที่มีประสิทธิภาพที่สกัดเอาเฉพาะกลุ่มรายการที่เกิดบ่อยแบบปิดจากฐานข้อมูลขนาดใหญ่แทนการสืบค้นกลุ่มรายการที่เกิดบ่อยทั้งหมด การค้นหาและจัดเก็บเฉพาะกลุ่มรายการที่เกิดบ่อยแบบปิดช่วยให้ลดปริมาณการจัดเก็บข้อมูลได้เป็นอย่างมาก กลุ่มรายการที่เกิดบ่อยแบบปิดที่ได้จากวิธีการนี้ยังสามารถนำไปใช้งานเพื่อสื่อความหมายแทนกลุ่มรายการที่เกิดบ่อยทั้งหมดที่อยู่ในคลาสสมมูลเดียวกันโดยไม่ทำให้สูญเสียความหมายของข้อมูลไป ซึ่งได้กำหนดนิยามของกลุ่มรายการที่เกิดบ่อยแบบปิดไว้ดังนี้

นิยามที่ 2.2 *กลุ่มรายการ P จะถูกเรียกว่าเป็นกลุ่มรายการแบบปิด (Closed Itemset) ก็ต่อเมื่อกลุ่มรายการ P ไม่มีซุปเปอร์เซต (Superset) และสามารถใช้อธิบายความหมายได้ครอบคลุมกลุ่มรายการทั้งหมดที่อยู่ในคลาสสมมูล (Equivalence class) ที่มีค่าสนับสนุนเดียวกันโดยไม่ทำให้สูญเสียความหมายสำคัญของข้อมูลทั้งหมดไป โดยกลุ่มรายการแบบปิด P จะเป็นกลุ่มรายการที่เกิดบ่อยแบบปิดก็ต่อเมื่อมีค่าความถี่มากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำที่กำหนดไว้*



รูปที่ 2.5 ตัวอย่างโครงข่ายของกลุ่มรายการที่เกิดบ่อยแบบปิดจากข้อมูลตัวอย่างในรูปที่ 1.1 (ก)

กลุ่มรายการที่เกิดบ่อยแบบปิดเป็นกลุ่มรายการที่อยู่ในระดับสูงสุดของคลาสสมมูล โดยที่ไม่มีซุปเปอร์เซตที่สูงกว่า ยกตัวอย่างเช่น {*abc*:5} เป็นกลุ่มรายการที่สามารถใช้อธิบายความหมายครอบคลุมกลุ่มรายการ {*ab*:5, *ac*:5, *bc*:5, *a*:5, *b*:5, *c*:5} ได้ครอบคลุมทั้งหมด โดยในคลาสสมมูลนี้ไม่มีเซตใด ๆ ที่สูงกว่ากลุ่มรายการนี้อีกแล้ว ดังนั้นจึงเรียกกลุ่มรายการ {*abc*:5} ว่าเป็นกลุ่มรายการที่เกิดบ่อยแบบปิด

จากรูปที่ 2.5 แสดงโครงข่ายของกลุ่มรายการที่เกิดบ่อยแบบปิดจากฐานข้อมูลตัวอย่างในรูปที่ 1.1 (ก) กลุ่มรายการที่เกิดบ่อยแบบปิดจากข้อมูลตัวอย่างคือ {*abcd*:2, *abc*:5, *abd*:3, *ab*:6, *ac*:6, *ad*:5, *a*:9} หากจัดเก็บเฉพาะกลุ่มรายการแบบปิดจะจัดเก็บกลุ่มรายการเพียง 7 รายการจากทั้งหมด 15 รายการ ทำให้ประหยัดพื้นที่ในการจัดเก็บข้อมูลได้เป็นอย่างมาก โดยเฉพาะอย่างยิ่งในกรณีที่ข้อมูลมีจำนวนมาก ๆ วิธีการนี้ทำให้ประหยัดทั้งพื้นที่หน่วยความจำและเวลาในการประมวลผล โดยแต่ละอัลกอลิทึมก็จะมีวิธีการทำงานและโครงสร้างข้อมูลของตัวเอง ซึ่งก็มีทั้งจุดดีและจุดด้อยแตกต่างกันไป โครงสร้างข้อมูลถือว่าเป็นจุดสำคัญอย่างมากต่อประสิทธิภาพการทำงานของแต่ละวิธีการ โครงสร้างข้อมูลที่ดีควรจะสนับสนุนการทำงานที่ใช้งานได้ง่ายและรวดเร็วทั้งในเรื่องการเข้าถึงและการคำนวณหาค่าความถี่ของแต่ละกลุ่มรายการ และที่สำคัญเพื่อให้รองรับการทำงานกับข้อมูลขนาดใหญ่ได้ และควรใช้พื้นที่จัดเก็บข้อมูลในหน่วยความจำหลักในขณะประมวลผลของแต่ละกลุ่มรายการน้อย ซึ่งในปัจจุบันนี้มีนักวิจัยหลายท่านได้นำเสนอโครงสร้างข้อมูลออกมาหลากหลายวิธีการ ([17], [20], [24], [26], [28]) ซึ่งในที่นี้ได้ศึกษาบางอัลกอลิทึมที่มีประสิทธิภาพที่นำเสนอพร้อมโครงสร้างข้อมูลเพื่อใช้ในการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด

2.2 งานวิจัยที่เกี่ยวข้อง

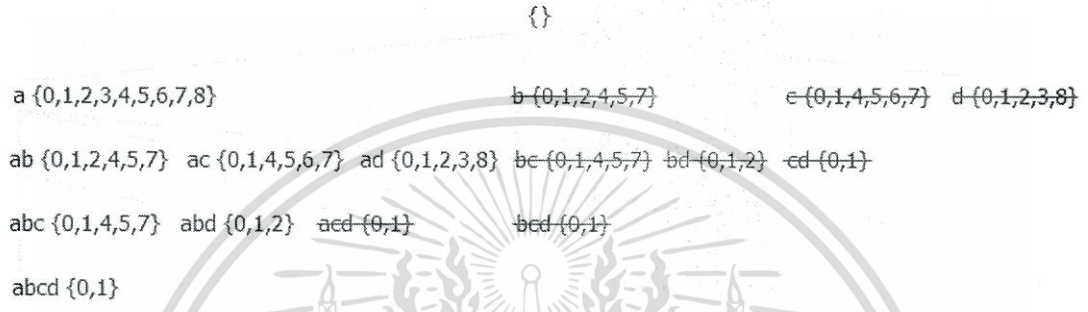
ในส่วนนี้ได้นำเสนองานวิจัยที่เกี่ยวข้องเกี่ยวกับการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดพร้อมทั้งโครงสร้างข้อมูลที่แต่ละอัลกอลิทึมใช้งาน

2.2.1 วิธีการ CHARM

วิธีการ CHARM และโครงสร้างข้อมูลไอที-ทรี (Itemset-Tidset Search Tree: IT-TREE) ได้ถูกนำเสนอเพื่อใช้ในการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดในปี 2002 โดยนักวิจัยสองท่านคือ M.J. Zaki และ C. Hsiao [17] แนวคิดในการออกแบบโครงสร้างข้อมูลนี้ก็เพื่อที่จะทำให้วิธีการสามารถรองรับการค้นหากลุ่มรายการในฐานข้อมูลที่มีความหนาแน่นของข้อมูลทั้งในปริมาณมากและในปริมาณที่เบาบางได้อย่างรวดเร็ว ซึ่งโครงสร้างข้อมูลหลักของวิธีการนี้ประกอบด้วยโครงสร้างข้อมูลไอที-ทรีที่เป็นโครงสร้างข้อมูลพรีฟิคซ์ทรีและตารางแฮส (Hashing Table)

โดยในแต่ละโหนดของโครงสร้างข้อมูลไอที-ทรีจะจัดเก็บชื่อของกลุ่มรายการและหมายเลขทรานแซคชันที่กลุ่มรายการนี้ปรากฏเอาไว้เพื่อให้ง่ายต่อการคำนวณหาความถี่ (จัดเก็บข้อมูลเป็นไบท์) โดยใช้เวลาในการสร้างโครงสร้างข้อมูลเท่ากับ $O(mT) + O(f_i n)$ เมื่อ m คือจำนวนกลุ่มรายการที่เกิดบ่อย f_i คือความถี่ของโหนด i ใด ๆ และ n คือจำนวนโหนดที่มีในโครงสร้างข้อมูล

วิธีการ CHARM ค้นหากลุ่มรายการที่เกิดบ่อยโดยใช้ความสัมพันธ์กันของข้อมูลที่มีอยู่ในแต่ละทรานแซคชันมาจัดวางเป็นเลเวล (Level) ของโครงสร้างไอที-ทรีดังแสดงในรูปที่ 2.6



รูปที่ 2.6 กลุ่มรายการที่เกิดบ่อยแบบปิดในโครงสร้างข้อมูลไอที-ทรี

โครงสร้างข้อมูลไอที-ทรีเริ่มสร้างจากส่วนที่เป็นโหนดรูท (Root Node) ก่อน จากนั้นจึงนำเอากลุ่มรายการขนาด 1 รายการที่ได้จากฐานข้อมูลมาสร้างเป็นเลเวลแรกของโครงสร้างข้อมูลไอที-ทรีต่อจากโหนดรูท และสร้างโครงข่ายที่สัมพันธ์กันจากแต่ละโหนดเป็นเลเวลต่อ ๆ ไป จากรูปที่ 2.6 สมมติให้ค่าสนับสนุนขั้นต่ำเท่ากับ 1 โหนดที่เชื่อมต่อจากโหนดรูทคือ a, b, c และ d ซึ่งเป็นกลุ่มรายการที่เกิดบ่อยขนาด 1 รายการที่ได้จากฐานข้อมูลตัวอย่างในรูปที่ 1.1 (ก) กลุ่มความสัมพันธ์เลเวลแรกของ $\{a\}$ ประกอบด้วยกลุ่มรายการที่มี $\{a\}$ เป็นโหนดพรีฟิคซ์ (Prefix Node) โดยมีกลุ่มรายการขนาด 2 รายการทั้งหมดได้แก่ $\{ab, ac, ad\}$ โดยแต่ละโหนดของโครงสร้างไอที-ทรีจะจัดเก็บชื่อกลุ่มรายการ ($\leq m$ ไบต์) ค่าสนับสนุน (หมายเลขทรานแซคชัน) พอยเตอร์เชื่อมโยงหาโหนดแม่ และพอยเตอร์เชื่อมโยงหาโหนดลูกทั้งหมด ($\leq m$) โดยจำนวนของโหนดในโครงสร้างข้อมูลพรีฟิคซ์นั้นจะเท่ากับ 2^m-1 โดย m คือกลุ่มรายการขนาด 1 รายการ.

วิธีการนี้ค้นหากลุ่มรายการที่เกิดบ่อยแบบปิดโดยเริ่มด้วยการคำนวณหาอินเตอร์เซกชั่น (Intersection) ของตัวสร้างเปรียบเทียบกับกลุ่มรายการที่เป็นโพสเซต (Post set) เช่น หากต้องการหาว่า b เป็นกลุ่มรายการที่เกิดบ่อยแบบปิดหรือไม่ จะเริ่มจากตรวจสอบก่อนว่า b เป็นตัวสร้างที่เป็นไปได้หรือไม่ หากปรากฏว่าในทุก ๆ ทรานแซคชันที่มี b จะมี a ด้วย กลุ่มรายการ b จึงไม่ใช่ตัวสร้างที่เป็นไปได้ ต้องนำเอา ab มาเป็นตัวสร้างที่เป็นไปได้แทน จากนั้นจึงนำไปตรวจสอบกับกลุ่มรายการที่เป็นโพสเซตของ ab ซึ่งก็คือ c และ d นั้นเอง เพื่อหาว่า ab เป็นกลุ่มรายการที่เกิดบ่อยแบบ

ปิดหรือไม่ ด้วยการคำนวณหาอินเตอร์เซกชั่นของเซตดังกล่าว จากกลุ่มรายการตัวอย่างนั้นในบางทรานแซคชันที่ *ab* ปรากฏไม่มี *c* หรือ *d* ดังนั้น *ab* จึงสามารถเป็นกลุ่มรายการที่เกิดบ่อยแบบปิดได้ หากต้องการหากลุ่มรายการที่เกิดบ่อยแบบปิดต่อก็นำเอากลุ่มรายการที่เกิดบ่อยแบบปิด *ab* และ *c* มาสร้างเป็นตัวสร้าง *abc* แล้วคำนวณหากลุ่มรายการที่เกิดบ่อยแบบปิดโดยคำนวณหาอินเตอร์เซกชั่นกับกลุ่มรายการที่เป็นโพสเซตต่อไปเรื่อย ๆ

ข้อดีของวิธีการนี้คือการใช้โครงสร้างข้อมูลพรีฟิคซ์ทรีที่สามารถเข้าถึงแต่ละโหนดได้อย่างรวดเร็วโดยใช้ตารางแฮสมาช่วยในการเข้าถึงระหว่างโหนดแม่และโหนดลูก ($O(mn)$) ซึ่งถือว่าเป็นวิธีที่มีประสิทธิภาพ แต่อย่างไรก็ตามในขั้นตอนการเตรียมข้อมูลเพื่อสร้างโครงสร้างข้อมูลยังคงมีปัญหาอยู่ เนื่องจากจะต้องเริ่มคำนวณจากโหนดรูทก่อนเสมอ โดยเฉพาะอย่างยิ่งในกรณีที่ฐานข้อมูลมีความหนาแน่นของข้อมูลจำนวนมากอาจจะใช้เวลาในการทำงานนาน นอกจากนี้วิธีการนี้นำเอากลุ่มรายการมาจัดเก็บเป็นชื่อของแต่ละโหนดและเก็บหมายเลขทรานแซคชันที่กลุ่มรายการในแต่ละโหนดปรากฏเอาไว้ (จัดเก็บเป็นไบต์) จะต้องใช้เนื้อที่ในการจัดเก็บข้อมูลในหน่วยความจำมากขึ้นเรื่อย ๆ อาจจะมีปัญหาในกรณีที่ฐานข้อมูลมีขนาดใหญ่และมีความหนาแน่นของกลุ่มรายการจำนวนมาก

2.2.2 วิธีการ DCI-CLOSED

วิธีการ DCI-CLOSED [26] เป็นอัลกอลิทึมที่มีประสิทธิภาพที่พัฒนาขึ้นสำหรับการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดใช้ลักษณะการทำงานทางลึกก่อน (Depth-first) วิธีการนี้ใช้โครงสร้างข้อมูลบิทแมบ (รูปที่ 2.7) ที่ใช้การจัดเก็บข้อมูลเป็นบิท (0 หรือ 1) โดยหนึ่งบิทจะแทน 1 กลุ่มรายการ ซึ่งเป็นโครงสร้างข้อมูลที่มีประสิทธิภาพมากในเรื่องของการจัดเก็บข้อมูลในหน่วยความจำในขณะประมวลผลโดยเฉพาะอย่างยิ่งในฐานข้อมูลขนาดใหญ่ เพราะจัดเก็บข้อมูล 1 รายการต่อหนึ่งบิทเท่านั้น จึงทำให้ประหยัดพื้นที่ในการจัดเก็บข้อมูล

tid	Item			
	a	b	c	d
0	1	1	1	1
1	1	1	1	1
2	1	1	0	1
3	1	0	0	1
4	1	1	1	0
5	1	1	1	0
6	1	0	1	0
7	1	1	1	0
8	1	0	0	1

รูปที่ 2.7 โครงสร้างข้อมูลบิทแมบแสดงข้อมูลจากฐานข้อมูลตัวอย่าง

วิธีการนี้เริ่มการทำงานโดยการสแกนฐานข้อมูลเพื่อหากลุ่มรายการขนาด 1 รายการ จากนั้นจะสร้างตารางเมทริกซ์ขนาด $T \times m$ ขึ้น โดย m คือจำนวนกลุ่มรายการที่เกิดบ่อยมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ และ T คือจำนวนทรานแซกชั่นทั้งหมด เสร็จแล้วจะแปลงกลุ่มรายการให้อยู่ในรูปของ 0/1 บิทเพื่อจัดเก็บในตารางเมทริกซ์โดยแทน 1 ในตำแหน่งที่มีข้อมูลในทรานแซคชัน และแทน 0 ในตำแหน่งที่ไม่มีข้อมูลในทรานแซคชัน ตัวอย่างเช่นกลุ่มรายการ *abd* ในทรานแซกชั่นที่ 2 ของฐานข้อมูลตัวอย่างในรูปที่ 1.1 (ก) จะถูกจัดรูปแบบในตารางเมทริกซ์ดังนี้คือ 1101 หลังจากแปลงข้อมูลในรูปแบบตารางเมทริกซ์เรียบร้อยก็จะทำการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดต่อไป

ถึงแม้ว่าโครงสร้างข้อมูลนี้จะมีประสิทธิภาพในด้านการจัดเก็บข้อมูลในหน่วยความจำหลักแต่อาจจะใช้เวลาในการค้นหาค่าความถี่ของกลุ่มรายการนานในกรณีที่มีจำนวนทรานแซคชันจำนวนมากและข้อมูลมีความหนาแน่นน้อย ($O(mT^2)$) ซึ่งจะมีบิท 0 จำนวนมากส่งผลให้ไม่สามารถทำงานได้อย่างเต็มประสิทธิภาพอาจจะใช้เวลาในการทำงานเท่ากับ mT^2

2.2.3 วิธีการ LCM

วิธีการ LCM (หรือ Linear time Closed itemset Mining) เป็นวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดที่มีประสิทธิภาพ นำเสนอครั้งแรกในปี 2003 [20] และได้พัฒนาประสิทธิภาพอย่างต่อเนื่องจนถึงเวอร์ชั่นล่าสุดในปี 2005 [24] ในเวอร์ชั่นที่ 1 [20] และ 2 [21] ของวิธีการนี้ใช้โครงสร้างข้อมูลอาร์เรย์ลิตส์ (รูปที่ 2.8) เพื่อจัดเก็บข้อมูลจากฐานข้อมูลไว้ในหน่วยความจำในขณะประมวลผลและใช้โครงสร้างข้อมูลแบบรวม (รูปที่ 2.9) ที่เกิดจากการนำเอาโครงสร้างข้อมูลอาร์เรย์ลิตส์ โครงสร้างข้อมูลบิทแมบ และโครงสร้างข้อมูลพรีฟิคซ์ทรีมาใช้ในเวอร์ชั่นที่ 3

โครงสร้างข้อมูลอาร์เรย์ลิตส์ที่ใช้ในเวอร์ชั่น 1 และ 2 นั้นใช้รูปแบบการจัดเก็บข้อมูลในหน่วยความจำแบบอาร์เรย์ของข้อมูล โดยการนำเอากลุ่มรายการในฐานข้อมูลมาจัดเรียงเป็นอาร์เรย์ตามลำดับความถี่ของแต่ละรายการ วิธีการนี้จะสร้างอาร์เรย์บักเก็ต (Array Bucket) ขึ้นมาเพื่อจัดเก็บทรานแซคชันของข้อมูลแต่ละรายการตามกลุ่มรายการขนาด 1 รายการแต่ละตัว และจะสแกนอาร์เรย์บักเก็ตเพื่อคำนวณหาความถี่ของกลุ่มรายการ การสร้างโครงสร้างข้อมูลอาร์เรย์ลิตส์นั้นเริ่มจากการสแกนฐานข้อมูลในรอบแรกเพื่อหาจำนวนกลุ่มรายการขนาด 1 รายการและความถี่ของแต่ละรายการพร้อมทั้งหาจำนวนทรานแซคชันทั้งหมดที่มีอยู่ในฐานข้อมูล จากนั้นจะทำการจัดเรียงกลุ่มรายการขนาด 1 รายการ ตามความถี่จากมากไปหาน้อย เสร็จแล้วจึงสร้างอาร์เรย์บักเก็ตตามจำนวนกลุ่มรายการที่มีค่าความถี่มากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำที่กำหนดไว้ และทำการสแกนฐานข้อมูลอีกครั้งเพื่ออ่านหมายเลขทรานแซคชันแต่ละรายการมาเก็บไว้ในอาร์เรย์บักเก็ต จากนั้นจึงเข้าสู่ขั้นตอนการคำนวณหากลุ่มรายการที่เกิดบ่อยแบบปิดด้วยอัลกอลิทึม LCM ต่อไป

จากรูปที่ 2.8 (ข) อาร์เรย์บักเก็ต b จัดเก็บหมายเลขทรานแซคชัน 0, 1, 2, 4, 5 และ 7 นั่นเป็นเพราะว่า b ปรากฏอยู่ในทรานแซคชันดังกล่าว โดยความถี่ของ b จะเท่ากับ 6 ซึ่งก็คือจำนวนทรานแซคชันทั้งหมดที่มี b อยู่นั้นเอง

อัลกอริทึม LCM ใช้เทคนิคในการคำนวณหาความถี่ของกลุ่มรายการที่เรียกว่าออคเคอร์เรนซ์ดิลิเวอร์ (Occurrence Deliver) ซึ่งเป็นเทคนิคที่มีประสิทธิภาพ ในเวอร์ชั่น 1 และ 2 จะคำนวณหาความถี่ของตัวสร้างและโคลสเซอร์ (Closure) จากหมายเลขทรานแซคชันที่เก็บไว้ในอาร์เรย์บักเก็ตของตัวสร้างขนาด 1 รายการ โคลสเซอร์คือกลุ่มรายการที่มีความสัมพันธ์กับตัวสร้างเพื่อใช้สำหรับค้นหากลุ่มรายการที่เกิดบ่อยแบบปิด ซึ่งอาร์เรย์บักเก็ตของโคลสเซอร์ที่สร้างขึ้นนี้สามารถใช้ในการหากลุ่มรายการที่เกิดบ่อยแบบปิดได้อีกโดยไม่ต้องสร้างใหม่ในทุกรอบการค้นหา จึงทำให้ประหยัดพื้นที่หน่วยความจำ โดยเนื้อที่ในการจัดเก็บข้อมูลในหน่วยความจำของโครงสร้างข้อมูลอาร์เรย์ลิตส์เท่ากับ $mf_i(\max(T))$ โดย f_i คือความถี่ของ i และ $\max(T)$ คือค่าสูงสุดของจำนวนทรานแซกชั่น อย่างไรก็ตามถึงแม้ว่าโครงสร้างข้อมูลนี้สามารถทำงานได้อย่างมีประสิทธิภาพ แต่ในบางกรณีที่ฐานข้อมูลมีขนาดใหญ่ ข้อมูลมีความหนาแน่นมาก ประสิทธิภาพของโครงสร้างข้อมูลนี้จะลดลงและหน่วยความจำอาจจะไม่สามารถรองรับกับข้อมูลทั้งหมดได้

tid	Item
0	a b c d
1	a b c d
2	a b d
3	a d
4	a b c
5	a b c
6	a c
7	a b c
8	a d

(ก)

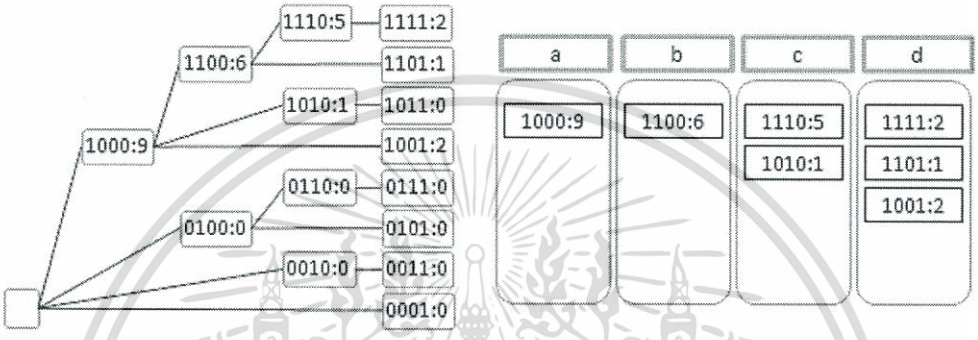
Occurrence Deliver			
a	b	c	d
0	0	0	0
1	1	1	1
2	2	4	2
3	4	5	3
4	5	6	8
5	7	7	
6			
7			
8			

(ข)

รูปที่ 2.8 ตัวอย่าง (ก) โครงสร้างข้อมูลอาร์เรย์ลิตส์, (ข) ข้อมูลออคเคอร์เรนซ์ดิลิเวอร์เพื่อคำนวณหาความถี่จากข้อมูลตัวอย่างในรูปที่ 2.8 (ก)

อัลลกอริทึม LCM ได้พัฒนาโครงสร้างข้อมูลขึ้นมาอีกหนึ่งโครงสร้างเรียกว่า โครงสร้างข้อมูลแบบรวม (Collaboration of Array lists, Vertical Bitmap, and Prefix-tree data structure) (รูปที่ 2.9) โดยโครงสร้างข้อมูลนี้เกิดจากการรวมเอาข้อดีของโครงสร้างข้อมูลพื้นฐาน 3 โครงสร้างข้อมูลเพื่อให้ได้ประสิทธิภาพในการทำงานเพิ่มขึ้นทั้งในเรื่องการคำนวณหาความถี่และทำให้ประหยัดพื้นที่หน่วยความจำ

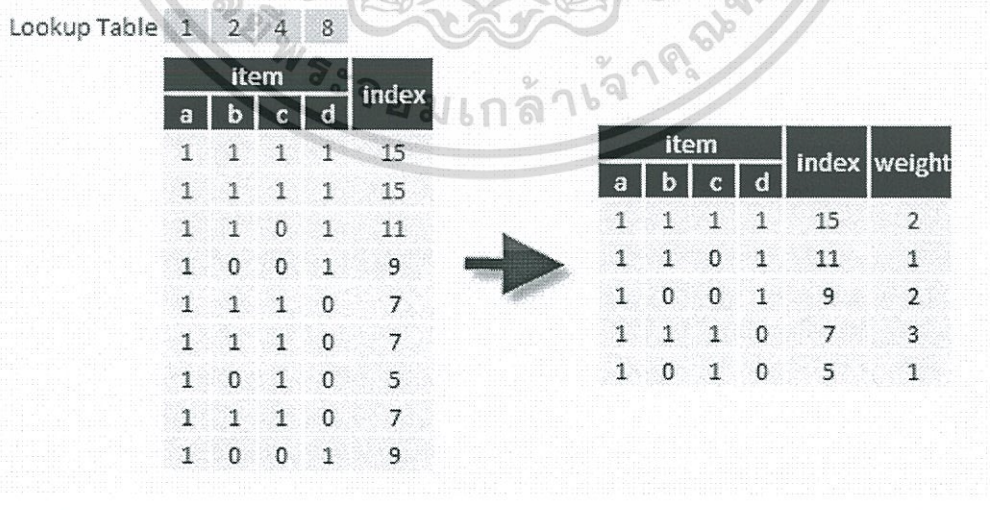
โครงสร้างข้อมูลพื้นฐานของวิธีการนี้คือ โครงสร้างข้อมูลอาร์เรย์ลิตส์ โครงสร้างข้อมูลบิทแมบ และโครงสร้างข้อมูลพรีฟิคซ์ทรี ซึ่งพัฒนาขึ้นมาเพื่อทำงานกับ LCM-3 [24] เพื่อให้สามารถรองรับการทำงานในฐานข้อมูลที่มีข้อมูลจำนวนมากขึ้นทั้งฐานข้อมูลที่มีปริมาณข้อมูลหนาแน่น (Dense) และฐานข้อมูลที่มีปริมาณข้อมูลเบาบาง (Spare) โครงสร้างข้อมูลหลักของวิธีการนี้คือโครงสร้างข้อมูลอาร์เรย์ลิตส์ที่จัดรูปแบบข้อมูลแยกออกตามเลเวลของโครงสร้างข้อมูลพรีฟิคซ์ทรี และจัดเก็บชื่อของแต่ละโหนดในรูปแบบ 0/1 บิทเหมือนโครงสร้างข้อมูลบิทแมบ ด้วยเหตุนี้เองโครงสร้างข้อมูลแบบรวมจึงช่วยให้อัลกอริทึม LCM ทำงานได้มีประสิทธิภาพมากยิ่งขึ้น



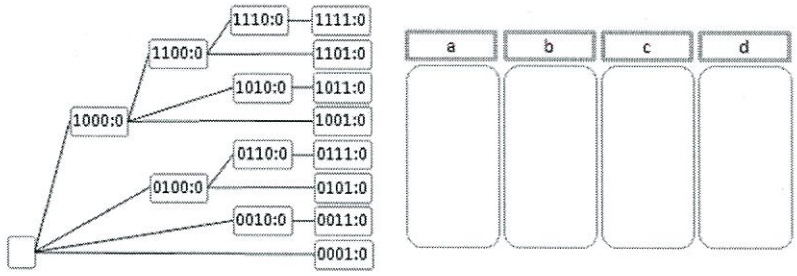
รูปที่ 2.9 โครงสร้างข้อมูลแบบรวมและการออคเคอร์เรนซ์ ดิลิเวอร์เพื่อคำนวณความถี่

สำหรับโครงสร้างข้อมูลนี้มีขั้นตอนการสร้างดังนี้

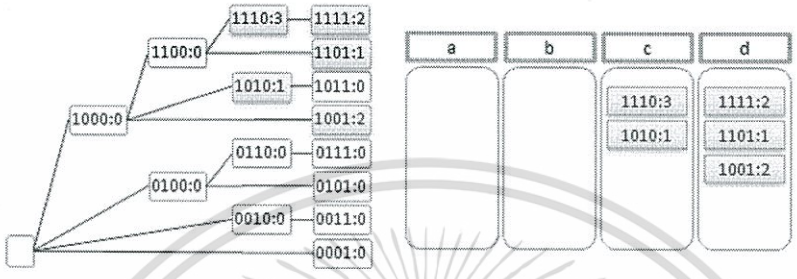
ขั้นตอนที่ 1 เริ่มต้นด้วยการสแกนฐานข้อมูลเพื่อหาจำนวนกลุ่มรายการขนาด 1 รายการทั้งหมด จำนวนทรานแซคชันและจำนวนกลุ่มรายการขนาด 1 รายการที่มีค่าความถี่ผ่านค่าสนับสนุนขั้นต่ำที่กำหนดไว้จากนั้นจะทำการจัดเรียงกลุ่มรายการที่เกิดบ่อยขนาด 1 รายการจากมากไปหาน้อย



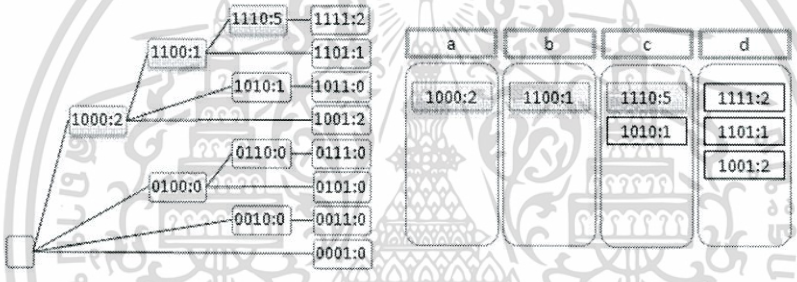
รูปที่ 2.10 คำนวณค่าดัชนีของแต่ละทรานแซกชั่นจากตารางดัชนีและรวมแถวที่ซ้ำกัน



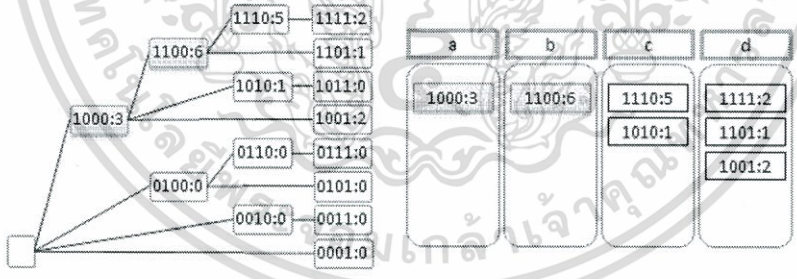
(ก)



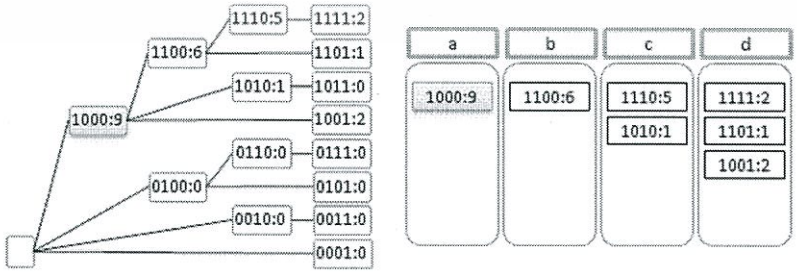
(ข)



(ค)



(ง)



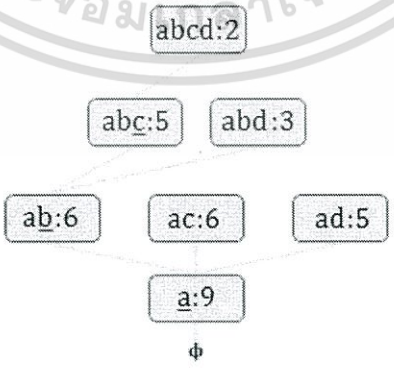
(จ)

รูปที่ 2.11 ขั้นตอนการสร้างโครงสร้างข้อมูลแบบรวม

จากนั้นวิธีการนี้จะคำนวณค่าดัชนีของแต่ละทรานแซคชันจากตำแหน่งบิทที่ข้อมูลปรากฏโดยใช้ค่าจากตารางดัชนี (Lookup Table) การคำนวณค่าของวิธีการนี้ทำโดยสแกนข้อมูลบิทแมบทรานแซคชันที่มีค่าบิทเป็น 1 โดยจะบวกค่าตามตารางดัชนีไปเรื่อย ๆ จนข้อมูลหมดทั้งแถวเสร็จแล้วจะทำการรวมทรานแซคชันที่ซ้ำกันเข้าด้วยกัน ดังแสดงในรูปที่ 2.10 ในการรวมแถวนั้นจะใช้วิธีการจัดเรียงแต่ละทรานแซคชันด้วยวิธีเรดิกซ์ (Radix Sort) ซึ่งตรงจุดนี้เป็นงานที่หนักที่สุดในขั้นตอนการเตรียมข้อมูลที่ค่อนข้างจะมีผลต่อประสิทธิภาพของโครงสร้างข้อมูลด้วย

ขั้นตอนที่ 2 เตรียมอาร์เรย์บักเก็ตสำหรับจัดเก็บข้อมูล โดยจำนวนอาร์เรย์บักเก็ตจะเท่ากับจำนวนของกลุ่มรายการที่เกิดบ่อยขนาด 1 รายการ ดังแสดงในรูปที่ 2.11 (ก) รูปด้านขวาคืออาร์เรย์บักเก็ตที่เตรียมไว้สำหรับจัดเก็บข้อมูล จากนั้นทำการอ่านค่าของแต่ละทรานแซคชันที่รวมกันเอาไว้แล้วในขั้นตอนก่อนหน้านี้เข้ามาใส่ในแต่ละอาร์เรย์บักเก็ตดังแสดงในรูปที่ 2.11 (ข)

ขั้นตอนที่ 3 เติมความถี่ให้กับโหนดแม่ โดยเริ่มทำงานจากอาร์เรย์บักเก็ตที่อยู่ทางด้านขวามือสุด เริ่มจากคำนวณหาโหนดแม่ ซึ่งวิธีการนี้ไม่ได้เก็บลิงค์เชื่อมโยงระหว่างโหนดแม่และโหนดลูกเอาไว้จึงใช้วิธีการตรวจสอบบิทข้อมูลว่าโหนดแม่อยู่ในอาร์เรย์บักเก็ตใด เมื่อทราบตำแหน่งอาร์เรย์บักเก็ตของโหนดแม่แล้วก็จะทำการคำนวณหาค่าดัชนีของโหนดแม่ด้วยการลบค่าตำแหน่งบิทของโหนดลูกออก หากเจอโหนดแม่อยู่ในอาร์เรย์บักเก็ตก็จะเพิ่มความถี่ด้วยการบวกค่าความถี่ให้กับโหนดแม่ หากไม่เจอโหนดแม่ก็จะเพิ่มข้อมูลของโหนดแม่เข้าไปในอาร์เรย์บักเก็ต ซึ่งจะทำขั้นตอนนี้ไปเรื่อย ๆ จนกว่าข้อมูลจะหมด ตัวอย่างการเติมข้อมูลจากอาร์เรย์บักเก็ตที่อยู่ขวาสุดไปเรื่อย ๆ จนถึงอาร์เรย์บักเก็ตที่อยู่ซ้ายสุดแสดงในรูปที่ 2.11 (ค)-(จ) โดยเริ่มทำงานที่เลเวลขวาสุดคืออาร์เรย์บักเก็ตของ d ไปจนถึง b ซึ่งจะได้โครงสร้างข้อมูลดังแสดงในรูปที่ 2.9 เวลาในการทำงานในขั้นตอนนี้เท่ากับ $O(T')$ โดย T' คือจำนวนทรานแซคชันที่ไม่ซ้ำกัน วิธีการนี้ใช้เวลาทำงานทั้งหมดเท่ากับ $O(mT) + O(mn_iT')$

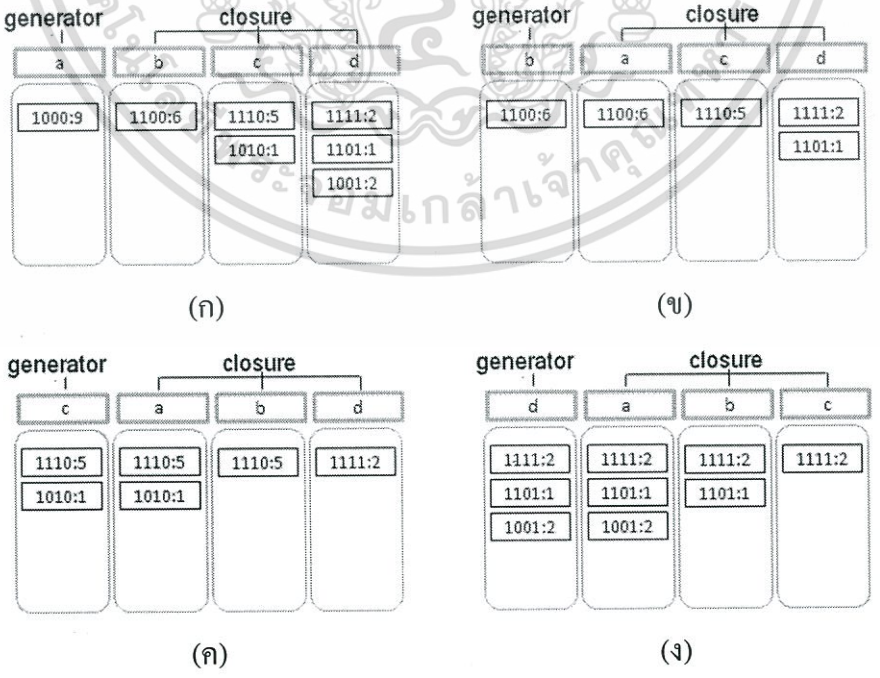


รูปที่ 2.12 โครงข่ายพีซีเอ็กเทนชั่นที่แสดงคีร์หลัก (Core Index: กลุ่มรายการที่ขีดเส้นใต้) ของแต่ละกลุ่มรายการที่เกิดบ่อยแบบปิด

การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดด้วยอัลกอลิทึม LCM นั้นใช้ได้ปรับปรุงการทำงานด้วยการใช้โคลสเซอร์ให้มีประสิทธิภาพมากขึ้น โดยเรียกวิธีการนี้ว่าพีพีซีเอ็กเทนชั่น (ppc-extension) ดังแสดงในรูปที่ 2.12 ซึ่งเป็นวิธีการที่ออกแบบมาเพื่อลดปริมาณข้อมูลโดยคัดกรองกลุ่มรายการที่เป็นโคลสเซอร์ให้ลดน้อยลงและไม่จำเป็นจะต้องเก็บกลุ่มรายการที่เกิดบ่อยแบบปิดที่ค้นพบก่อนหน้านี้ไว้เพื่อใช้ตรวจสอบความซ้ำซ้อน โดยพีพีซีเอ็กเทนชั่นกำหนดไว้ในนิยามดังนี้

นิยามที่ 2.3 *หาก P คือกลุ่มรายการที่เกิดบ่อยแบบปิด โดยค่าคีร์หลัก (Core Index) ของ P กำหนดให้เป็น core(P) คือกลุ่มรายการ i ที่เป็นกลุ่มรายการระดับต่ำสุดใน P โดยกลุ่มรายการ P' จะถูกเรียกว่าเป็นพีพีซีเอ็กเทนชั่นของ P ก็ต่อเมื่อ P' = closure(P ∪ {i}) โดยที่ P' จะมีกลุ่มรายการ i เพิ่มขึ้นมาจาก P และมีโคลสเซอร์ร่วมกัน ดังนั้นหาก P' เป็นพีพีซีเอ็กเทนชั่นของ P กลุ่มรายการ i จึงเป็นคีร์หลักของ P' (core(P'))*

วิธีการ LCM-3 ใช้วิธีพีพีซีเอ็กเทนชั่นทำงานร่วมกับออคเคอร์เรนซ์ดิลิเวอร์ โดยจะคัดกรองเอาเฉพาะข้อมูลที่สัมพันธ์กับตัวสร้างออกมาเพื่อให้สามารถทำงานได้เร็วขึ้น วิธีการนี้ใช้เวลาในการคำนวณหาความถี่เท่ากับ $O(m^2n)$ โดย m คือจำนวนกลุ่มรายการที่เกิดบ่อยขนาด 1 รายการและ n คือจำนวนโหนดในแต่ละอาร์เรย์บักเก็ต จากตัวอย่างในรูปที่ 2.13 ออคเคอร์เรนซ์ดิลิเวอร์ของตัวสร้างเริ่มต้นขนาด 1 รายการคือตัวสร้าง *a, b, c* และ *d* ซึ่งเตรียมไว้สำหรับหากลุ่มรายการที่เกิดบ่อยแบบปิดโดยใช้พีพีซีเอ็กเทนชั่น



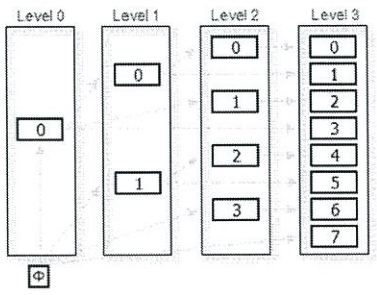
รูปที่ 2.13 ออคเคอร์เรนซ์ดิลิเวอร์ของตัวสร้าง *a, b, c* และ *d*

ปัจจัยหลักในการวัดประสิทธิภาพของโครงสร้างข้อมูลก็คือ วิธีการจัดเก็บข้อมูลในหน่วยความจำหลักในขณะประมวลผล และมีโครงสร้างที่สนับสนุนวิธีการคำนวณค่าความถี่ให้สามารถทำงานได้สะดวกรวดเร็ว อย่างที่ได้กล่าวไปแล้วว่าภาระงานที่หนักที่สุดของการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดก็คือการคำนวณหาค่าความถี่ของแต่ละกลุ่มรายการ ดังนั้นโครงสร้างข้อมูลที่ดีจะต้องเอื้อต่อการคำนวณหาค่าความถี่ให้สามารถทำงานได้อย่างมีประสิทธิภาพ ซึ่งจะส่งผลต่อประสิทธิภาพโดยรวมของวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดนั้นเอง จากที่ได้กล่าวมาแล้วเบื้องต้นว่า โครงสร้างข้อมูลและวิธีการค้นหาค่าความถี่เป็นปัจจัยหลักต่อประสิทธิภาพการทำงานของวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด ซึ่งทั้งสองปัจจัยนี้ช่วยให้วิธีการต่าง ๆ สามารถทำงานได้รวดเร็วขึ้นและรองรับการทำงานกับข้อมูลที่มีขนาดใหญ่ขึ้น ดังนั้นในงานวิจัยฉบับนี้จึงได้นำเสนอโครงสร้างข้อมูลอีบีพีเอ (EBPA) เพื่อใช้ในขั้นตอนการเตรียมข้อมูลสำหรับการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดให้ทำงานได้อย่างมีประสิทธิภาพ และได้ออกแบบโครงสร้างข้อมูลโดยพิจารณาจากปัญหาที่เกิดขึ้นกับโครงสร้างข้อมูลที่มีอยู่แล้วในปัจจุบัน โดยนำเอาโครงสร้างข้อมูลแบบรวมมาเป็นต้นแบบในการพัฒนาเพื่อให้ได้โครงสร้างข้อมูลที่มีประสิทธิภาพมากยิ่งขึ้น

บทที่ 3
โครงสร้างข้อมูลอีบีพีเอ

ในบทนี้นำเสนอเกี่ยวกับโครงสร้างข้อมูลอีบีพีเอ (an Efficient Bitmap Prefix-tree Array data structure: EBPA) สำหรับการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด ซึ่งเป็นโครงสร้างข้อมูลที่เกิดจากการนำเอาข้อดีของ 3 โครงสร้างข้อมูลที่มีประสิทธิภาพมารวมกันเพื่อให้ได้ประสิทธิภาพมากยิ่งขึ้น โดยได้นำเอาโครงสร้างข้อมูลแบบรวม (Collaboration data structure) [24] มาเป็นต้นแบบในการพัฒนา ซึ่งเป็นโครงสร้างข้อมูลแรกที่นำเอาโครงสร้างข้อมูลอาร์เรย์ลิตส์ (Array list) โครงสร้างข้อมูลบิทแมบ (Vertical Bitmap) และโครงสร้างข้อมูลพรีฟิคซ์ทรี (Prefix-tree) มาใช้ร่วมกันเพื่อช่วยให้วิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดสามารถทำงานได้ดียิ่งขึ้น โครงสร้างข้อมูลแบบรวมได้ให้ความสำคัญกับการจำกัดพื้นที่ในการค้นหาข้อมูลด้วยการกรองเอาเฉพาะข้อมูลที่เกี่ยวข้องเท่านั้น ถ้าหากขอบเขตของข้อมูลที่จะต้องค้นหาแคบลงก็จะใช้เวลาในการทำงานน้อยลงตามไปด้วย นอกจากนี้ยังให้ความสำคัญกับวิธีการจัดเก็บข้อมูลในหน่วยความจำหลักในขณะประมวลผลด้วยการออกแบบการจัดเก็บข้อมูลให้ใช้งานร่วมกันได้ในระหว่างกลุ่มรายการที่มีความสัมพันธ์กัน

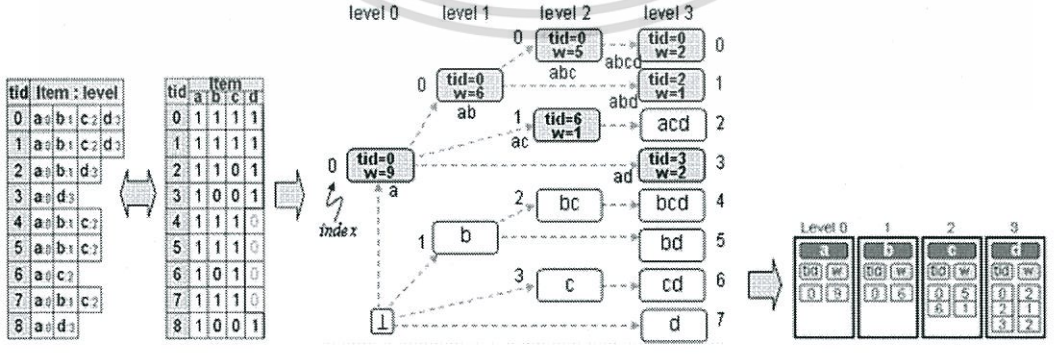
การปรับปรุงประสิทธิภาพของโครงสร้างข้อมูลนั้นได้นำเอาโครงสร้างข้อมูลแบบรวมมาเป็นต้นแบบและปรับปรุงใน 2 ส่วนหลัก ๆ ก็คือ 1) การเติมค่าความถี่จากทรานแซกชั่นข้อมูลลงในโครงสร้างข้อมูลด้วยการใช้เทคนิคกำหนดค่าดัชนี (index) ของสมาชิกแต่ละโหนดให้กับพรีฟิคซ์อาร์เรย์ (Prefix Array) ที่เป็นส่วนหนึ่งของโครงสร้างข้อมูลอีบีพีเอ ค่าดัชนีนี้ได้มาจากการคำนวณตำแหน่งโหนดต่าง ๆ ในแต่ละเลเวล (level) ของโครงสร้างข้อมูลพรีฟิคซ์ทรีแบบสมบูรณ์ (Complete Prefix-tree) พร้อมทั้งออกแบบการเชื่อมโยงระหว่างโหนดแม่และโหนดลูกเพื่อความรวดเร็ว ทำให้การเพิ่มความถี่ของแต่ละโหนดสามารถทำได้โดยไม่จำเป็นต้องจัดเรียงและรวมแถวข้อมูล และไม่ต้องเก็บลิงค์เชื่อมโยงระหว่างโหนดแม่กับโหนดลูกทำให้ประหยัดพื้นที่ในหน่วยความจำเพิ่มขึ้น และ 2) ลดการจัดเก็บรายละเอียดข้อมูลในแต่ละโหนดเพื่อช่วยให้ประหยัดพื้นที่หน่วยความจำหลักโดยใช้บิทร่วมกันในกลุ่มรายการที่มีเส้นทางเดียวกัน



รูปที่ 3.1 การจัดเลเวลต่าง ๆ ในโครงสร้างข้อมูลพรีฟิคซ์ทรีแบบสมบูรณ์ของ 4 กลุ่มรายการ

จากรูปที่ 3.1 เป็นตัวอย่างของโครงสร้างข้อมูลพรีฟิคซ์ทรีแบบสมบูรณ์ที่ประกอบด้วย 4 เลเวล เริ่มจากเลเวล 0 ถึงเลเวล 3 โดยในแต่ละเลเวลมีจำนวนโหนดเท่ากับ 2^{level} ในเลเวล 0 (level 0) มีจำนวนโหนดทั้งหมดเท่ากับ 2^0=1 ในเลเวลที่ 1 (level 1) มีจำนวนโหนดทั้งหมดเท่ากับ 2^1=2 ในเลเวลที่ 2 (level 2) มีจำนวนโหนดเท่ากับ 2^2=4 และในเลเวลที่ 3 (level 3) มีจำนวนโหนดเท่ากับ 2^3=8 หากนำเอาโหนดในแต่ละเลเวลมาจัดเรียงให้อยู่ในรูปแบบของพรีฟิคซ์อาร์เรย์ก็จะทำให้สามารถอ้างอิงตำแหน่งและสามารถเข้าถึงทุกโหนดได้โดยไม่จำเป็นต้องเข้าจากรูทโหนดและไม่ต้องสร้างโครงสร้างข้อมูลพรีฟิคซ์ทรีแบบสมบูรณ์ ในส่วนของการสร้างโหนดในพรีฟิคซ์อาร์เรย์นั้นจะสร้างเพียงแค่โหนดที่มีข้อมูลตรงกับข้อมูลในฐานข้อมูลเท่านั้นเพื่อจะได้ประหยัดพื้นที่และใช้หน่วยความจำได้อย่างมีประสิทธิภาพ

ตัวอย่างในรูปที่ 3.2 (ข) แสดงข้อมูลจากฐานข้อมูลตัวอย่างในรูปที่ 3.2 (ก) เมื่อนำเอาข้อมูลมาเติมเข้าในโครงสร้างข้อมูลพรีฟิคซ์ทรีจะมีโหนดข้อมูลเพียง 7 โหนดเท่านั้น โดยเลเวล 0 มีจำนวน 1 โหนด *a:9* (ค่าดัชนี = 0) เลเวล 1 มีจำนวน 1 โหนด คือ *ab:6* (ค่าดัชนี = 0) เลเวล 2 มีจำนวน 2 โหนด คือ *abc:5* (ค่าดัชนี = 0) และ *ac:1* (ค่าดัชนี = 1) และเลเวล 3 มีจำนวน 3 โหนด คือ *abcd:3* (ค่าดัชนี = 0), *abd:1* (ค่าดัชนี = 1) และ *ad:2* (ค่าดัชนี = 3) เมื่อนำข้อมูลที่ได้มาจัดอยู่ในพรีฟิคซ์อาร์เรย์ก็จะได้ข้อมูลตามรูปที่ 3.2 (ค) นั้นเอง สังเกตว่าจากข้อมูลที่มีนั้นไม่จำเป็นจะต้องสร้างโหนดทั้งหมด (จำนวน 15 โหนด) โดยจะสร้างเพียงโหนดที่มีข้อมูลเท่านั้นทำให้ประหยัดพื้นที่ในการจัดเก็บข้อมูลได้มากยิ่งขึ้น การออกแบบโครงสร้างข้อมูลด้วยการแบ่งข้อมูลออกตามเลเวลของข้อมูลที่สัมพันธ์กันเหมือนเลเวลของโครงสร้างพรีฟิคซ์ทำให้สามารถคำนวณความถี่ของกลุ่มรายการในแต่ละเลเวลได้รวดเร็วมากยิ่งขึ้นนั้นเป็นเพราะว่าข้อมูลที่อยู่ในแต่ละเลเวลนั้นเป็นข้อมูลที่มีความสัมพันธ์กับกลุ่มรายการที่ต้องการค้นหาความถี่ ซึ่งจะมีจำนวนน้อยกว่าหากเทียบกับการค้นหาข้อมูลจากฐานข้อมูลโดยตรง เมื่อขอบเขตในการค้นหาน้อยลงการคำนวณความถี่ในแต่ละเลเวลจะใช้เวลาในการคำนวณที่น้อยลงตามไปด้วยทำให้มีประสิทธิภาพในการทำงานมากยิ่งขึ้น



(ก) (ข) (ค)

รูปที่ 3.2 ตัวอย่าง (ก) การแปลงข้อมูลเป็นบิทแมป (ข) โครงสร้างข้อมูลพรีฟิคซ์ทรี (ค) พรีฟิคซ์อาร์เรย์ในโครงสร้างข้อมูลอีบีพีเอ

ตัวอย่างเช่น หากเปรียบเทียบการหาความถี่ของกลุ่มรายการ c ในโครงสร้างข้อมูลอีบีพีเอที่อยู่ในรูปที่ 3.2 (ค) ทำได้โดยเข้าสู่เลเวลที่ 1 ซึ่งเก็บข้อมูลที่สัมพันธ์กับกลุ่มรายการ c ได้โดยตรง จากนั้นสแกนข้อมูลสมาชิกทั้งหมด (จำนวนสมาชิกเท่ากับ 2) แล้วรวมค่าความถี่ของสมาชิกเข้าด้วยกัน โหนด 0 มีความถี่เท่ากับ 5 และโหนด 1 มีความถี่เท่ากับ 1 ความถี่ของทั้งสองโหนดจึงเท่ากับ 6 สังเกตว่าหากคำนวณความถี่ของกลุ่มรายการ c ในโครงสร้างข้อมูลอีบีพีเอจะสแกนข้อมูลจากพรีฟิคซ์อาร์เรย์เพียง 2 ครั้งเท่านั้น แต่หากคำนวณความถี่จากโครงสร้างข้อมูลบิทแมบและโครงสร้างข้อมูลอาร์เรย์ลิตส์ต้องสแกนถึง 9 ครั้งและ 6 ครั้งตามลำดับ ดังนั้นโครงสร้างข้อมูลที่นำเสนอในงานวิจัยนี้จึงใช้เวลาทำงานสั้นกว่าโดยเฉพาะอย่างยิ่งในกรณีที่มีข้อมูลจำนวนมาก

ตารางที่ 3.1 เปรียบเทียบเวลาการทำงานของโครงสร้างข้อมูลพรีฟิคซ์ทรีในวิธีการต่าง ๆ

วิธีการ	เวลา
โครงสร้างข้อมูล IT-TREE ในวิธีการ CHARM [17] ข้อดี : ไม่ต้องมีการจัดเรียงข้อมูลทรานแซคชัน ข้อด้อย : ใช้พื้นที่ในการจัดเก็บข้อมูลรวมทั้งพอยเตอร์และตารางแฮส	$O(mT) + O(f_i n)$ *where* $f_i<T$
โครงสร้างข้อมูลแบบรวมใน LCM-3 [24] ข้อดี : ไม่ต้องมีพอยเตอร์และตารางแฮส ข้อด้อย : ต้องมีการจัดเรียงและรวมแถวข้อมูลทรานแซคชัน	$O(mT) + O(mn_iT')$
โครงสร้างข้อมูลอีบีพีเอ ข้อดี : ไม่ต้องมีการจัดเรียงและรวมแถวข้อมูลทรานแซคชัน ไม่ต้องมีพอยเตอร์และตารางแฮส ใช้การคำนวณค่าดัชนีแบบไดนามิกในการเข้าถึง ($O(1)$)	$O(mT) + O(mn)$

ในงานวิจัยนี้จึงได้นำเสนอโครงสร้างข้อมูลที่เกิดจากการนำเอาโครงสร้างข้อมูลแบบรวมมาปรับปรุงประสิทธิภาพในส่วนของการเข้าถึงระหว่างโหนดแม่และโหนดลูกด้วยการคำนวณค่าดัชนีแบบไดนามิก นอกจากนี้ยังนำเสนออัลกอริทึม EBPA-CLOSED ซึ่งเป็นวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดที่ปรับปรุงเวลาในการทำงานให้ดียิ่งขึ้น ดังแสดงในตารางที่ 3.1 นั้นเป็นการเปรียบเทียบเวลาในการทำงานของวิธีต่าง ๆ ที่ใช้โครงสร้างข้อมูลพรีฟิคซ์ทรีเป็นพื้นฐาน จากตารางจะเห็นว่าวิธีการในงานวิจัยนี้ใช้เวลาในการทำงานได้มีประสิทธิภาพมากกว่าวิธีการอื่น ๆ

โดยในบทนี้จะกล่าวถึงโครงสร้างข้อมูลอีบีพีเอซึ่งประกอบด้วยส่วนต่าง ๆ ดังนี้คือ

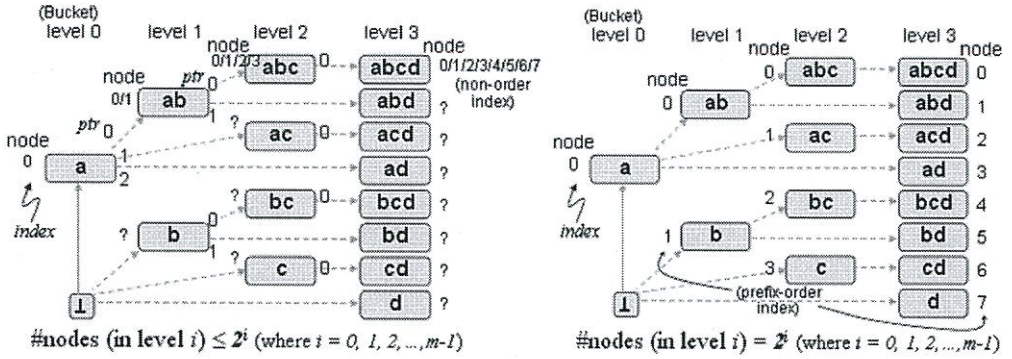
1. ทฤษฎีเกี่ยวข้องกับวิธีการเข้าถึงระหว่างโหนดแม่และโหนดลูกที่ใช้ในงานวิจัยชิ้นนี้ โดยได้นำเสนอประสิทธิภาพในการเข้าถึง ($O(1)$) โดยประยุกต์ใช้ค่าดัชนีที่ได้จากการคำนวณค่าแบบไดนามิก โดยไม่จำเป็นต้องสร้างการเชื่อมโยงและไม่ต้องใช้ตารางแฮสมาช่วยในการเข้าถึง นำเสนอในหัวข้อที่ 3.1

2. ขั้นตอนการสร้างโครงสร้างข้อมูลอีบีพีเอโดยใช้พรีฟิคซ์อาร์เรย์ร่วมกับการใช้บิทแมบทรานแซคชัน นำเสนอในหัวข้อที่ 3.2
3. เทคนิคในการปรับปรุงประสิทธิภาพของการเข้าถึงระหว่างโหนดแม่และโหนดลูก นำเสนอในหัวข้อที่ 3.3
4. ขั้นตอนการสร้างโครงสร้างข้อมูลอีบีพีเอด้วยเทคนิคปรับปรุงประสิทธิภาพ นำเสนอในหัวข้อที่ 3.4

3.1 ประสิทธิภาพในการติดต่อระหว่างโหนดแม่และโหนดลูก

เนื่องจากโครงสร้างข้อมูลอีบีพีเอในงานวิจัยนี้พัฒนาขึ้นมาโดยใช้โครงสร้างข้อมูลพรีฟิคซ์ทรีเป็นต้นแบบร่วมกับโครงสร้างข้อมูลบิทแมบและโครงสร้างข้อมูลอาร์เรย์ลิตส์ ซึ่งได้จัดเลเวลต่าง ๆ ของโครงสร้างข้อมูลพรีฟิคซ์ทรีให้อยู่ในรูปแบบพรีฟิคซ์อาร์เรย์และกำหนดให้ดัชนีของสมาชิกอาร์เรย์แทนโหนดต่าง ๆ ที่เกิดขึ้นในแต่ละเลเวลของโครงสร้างข้อมูลพรีฟิคซ์ทรี ดังแสดงในรูปที่ 3.1

สำหรับการเชื่อมโยงระหว่างโหนดแม่และโหนดลูกที่อยู่ต่างเลเวลกันเพื่อเพิ่มความถี่หรือตรวจสอบความสัมพันธ์ของข้อมูลในขั้นตอนการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด โดยธรรมชาติของโครงสร้างข้อมูลพรีฟิคซ์ทรีแล้วโหนดต่าง ๆ จะต้องจัดเก็บเส้นทางการเชื่อมโยงระหว่างกัน ซึ่งแต่ละโหนดจะต้องเก็บเส้นทางเชื่อมโยงหาโหนดแม่และเส้นทางการเชื่อมโยงหาโหนดลูกทั้งหมด หากโหนดแม่ต้องการที่จะติดต่อกับลูกโหนดใด ๆ ก็ตามจะต้องค้นหาเส้นทางของโหนดลูกที่ต้องการติดต่อจากเส้นทางของโหนดลูกทั้งหมดที่เก็บเอาไว้ ซึ่งจุดนี้เองที่ส่งผลต่อประสิทธิภาพของโครงสร้างข้อมูล ยิ่งมีโหนดลูกมากเท่าใดก็ยิ่งต้องใช้เวลาในการตรวจสอบเส้นทางนานขึ้นเท่านั้น ด้วยเหตุนี้เองในงานวิจัยนี้จึงได้ออกแบบรูปแบบการเชื่อมโยงกันระหว่างโหนดแม่และโหนดลูกให้มีการทำงานที่มีประสิทธิภาพมากยิ่งขึ้น โดยไม่จำเป็นต้องเก็บข้อมูลเส้นทางและไม่จำเป็นต้องค้นหาเส้นทางของโหนดลูกจากข้อมูลทั้งหมด การออกแบบนี้อาศัยความสัมพันธ์ของข้อมูลที่ได้แปลงทรานแซคชันจากฐานข้อมูลให้อยู่ในรูปแบบของบิทแมบทรานแซคชันหรือบิทแมบของกลุ่มรายการ ($b_0b_1b_2...b_{m-2}b_{m-1}$ เมื่อ m คือกลุ่มรายการที่เกิดบ่อย) โดย b_j ในบิทแมบทรานแซคชันนั้นแทนบิทของข้อมูลในเลเวล i ใด ๆ ด้วยรูปแบบของบิทแมบของกลุ่มรายการและการแบ่งข้อมูลออกเป็นเลเวลตามลักษณะของโครงสร้างข้อมูลพรีฟิคซ์ทรีนี้เอง งานวิจัยนี้จึงได้พัฒนาฟังก์ชั่นคำนวณตำแหน่งสมาชิก (Node's index addressing) แบบไดนามิกเพื่อที่จะเข้าถึงสมาชิกของพรีฟิคซ์อาร์เรย์ได้โดยไม่จำเป็น ต้องเก็บเส้นทางการเชื่อมโยงดังแสดงสมการที่ 1 โดยตำแหน่งของสมาชิกในแต่ละเลเวล i ใด ๆ นั้นจะเริ่มจาก $0, 1, 2, ..., n_i-1$ (เมื่อ n_i คือจำนวนของโหนดในเลเวล i ใด ๆ และ $i = 0, 1, 2, ..., m-1$)



(ก) (ข)

รูปที่ 3.3 (ก) โครงสร้างข้อมูลพรีฟิคซ์ทรีแบบกระชับที่ไม่มีการเรียงลำดับโหนด (ข) โครงสร้างข้อมูลพรีฟิคซ์ทรีแบบปกติที่มีการจัดเรียงลำดับของโหนดในแต่ละเลเวล

$$index = \sum_{j=0}^{i} w_j \quad where\ w_j = 2^{i-j-1}\ if\ b_j = 0 \qquad (1)$$

$$= 0 \qquad otherwise$$

การคำนวณตำแหน่งสมาชิกแบบไดนามิกนั้นเป็นการแปลงบิทแมบอาร์เรย์ของแต่ละกลุ่มรายการไปเป็นดัชนีของโหนดจากพรีฟิคซ์ของกลุ่มรายการที่สอดคล้องกัน โดยตำแหน่งสมาชิกคำนวณจากค่าน้ำหนัก (*weight (w)*) ของตำแหน่งบิทที่มีค่าเป็น 0 ($w_j = 2^{i-j-1}$) เริ่มคำนวณจากเลเวล 0 จนถึงเลเวล i ใด ๆ ที่โหนดที่คำนวณสังกัดอยู่ จากสมการที่ 1 นั้น j (b_j) คือตำแหน่งของบิท ซึ่งจะบ่งบอกถึงเลเวลของโหนดที่จะอยู่ในโครงสร้างข้อมูล ($j = 0, 1, 2, ..., i \leq m-1$)

ตารางที่ 3.2 การคำนวณค่าดัชนีของสมาชิกด้วยสมการที่ 1 ของทุกโหนดในพรีฟิคซ์อาร์เรย์

กลุ่มรายการ	บิทแมบ	เลเวล i	ค่าดัชนีของสมาชิก
a	**1**000	0	0 [i=0; (no 0 before level *i*)]
ab	**11**00	1	0 [i=1; (no 0 before level *i*)]
b	**01**00	1	1 [i=1; j=0; ($2^{1-0-1} = 1$)]
abc	**111**0	2	0 [i=2; (no 0 before level *i*)]
ac	**101**0	2	1 [i=2; j=1; ($2^{2-1-1} = 1$)]
bc	**011**0	2	2 [i=2; j=0; ($2^{2-0-1} = 2$)]
c	**001**0	2	3 [i=2; j=0,1; ($2^{2-0-1} + 2^{2-1-1} = 3$)]
abcd	**1111**	3	0 [i=3; (no 0 before level *i*)]
abd	**1101**	3	1 [i=3; j=2; ($2^{3-2-1} = 1$)]
acd	**1011**	3	2 [i=3; j=1; ($2^{3-1-1} = 2$)]
ad	**1001**	3	3 [i=3; j=1,2; ($2^{3-1-1} + 2^{3-2-1} = 3$)]
bcd	**0111**	3	4 [i=3; j=0; ($2^{3-0-1} = 4$)]
bd	**0101**	3	5 [i=3; j=0,2; ($2^{3-0-1} + 2^{3-2-1} = 5$)]
cd	**0011**	3	6 [i=3; j=0,1; ($2^{3-0-1} + 2^{3-1-1} = 6$)]
d	**0001**	3	7 [i=3; j=0,1,2; ($2^{3-0-1} + 2^{3-1-1} + 2^{3-2-1} = 7$)]

จากรูป 3.3 (ก) เป็นตัวอย่างตำแหน่งของโหนดในแต่ละเลเวลจากข้อมูลที่ประกอบด้วยกลุ่มรายการจำนวน 4 รายการคือ *a, b, c* และ *d* ($m = 4$) โดยในเลเวล 0 (level 0) มีจำนวนโหนดเท่ากับ 1 ($n_0=2^0 =1$) คือโหนด *a*(1) ค่าดัชนีของโหนดนี้ คำนวณโดยนำเอาบิทของโหนดนี้ที่มีค่าเป็น 0 มาคำนวณหาตำแหน่ง ซึ่งในโหนดนี้ไม่มีบิทที่มีค่าเป็น 0 ดังนั้นค่าดัชนีของโหนดนี้ คือ 0 นั้นเอง ต่อมาเลเวล 1 (level 1) มีจำนวนโหนดสูงสุดเท่ากับ 2 ($n_1 = 2^1 = 2$) คือโหนด *ab*(11) มีค่าดัชนีเท่ากับ 0 และโหนด *b*(01) มีค่าดัชนีเท่ากับ 1 ในเลเวลถัดมาเลเวล 2 (level 2) มีจำนวนโหนดสูงสุดเท่ากับ 4 ($n_2 = 2^2 = 4$) คือโหนด *abc*(111) ค่าดัชนีเท่ากับ 0 โหนด *ac*(101) ค่าดัชนีเท่ากับ 1 โหนด *bc*(011) ค่าดัชนีเท่ากับ 2 และโหนดสุดท้าย *c*(001) ค่าดัชนีเท่ากับ 3 ในเลเวลสุดท้ายของโครงสร้างข้อมูลคือ เลเวล 3 (level 3) มีจำนวนโหนดสูงสุดเท่ากับ 8 ($n_3 =2^3 =8$) คือ โหนด *abcd*(1111) ค่าดัชนีเท่ากับ 0 โหนด *abd*(1101) ค่าดัชนีเท่ากับ 1 โหนด *acd*(1011) ค่าดัชนีเท่ากับ 2 โหนด *ad*(1001) ค่าดัชนีเท่ากับ 3 โหนด *bcd*(0111) ค่าดัชนีเท่ากับ 4 โหนด *bd*(0101) ค่าดัชนีเท่ากับ 5 โหนด *cd*(0011) ค่าดัชนีเท่ากับ 6 และโหนดสุดท้าย *d*(0001) ค่าดัชนีเท่ากับ 7 สำหรับรายละเอียดของโหนดในแต่ละเลเวลนั้นดูข้อมูลเพิ่มเติมในตารางที่ 3.2 ซึ่งแสดงตัวอย่างของการคำนวณค่าดัชนีของสมาชิกทั้งหมดไว้ในคอลัมภ์ค่าดัชนีของสมาชิกนั้นเอง

จากค่าดัชนีของตำแหน่งในสมการที่ 1 นั้นประสิทธิภาพในการเข้าถึงระหว่างโหนดแม่และโหนดลูกในเวลา $O(1)$ ได้มาจากข้อกำหนดด้านล่างนี้

กำหนดให้ *ix* แทนค่าดัชนีของกลุ่มรายการขนาด 1 รายการที่ได้จากสมการที่ 2

cix แทนค่าดัชนีของกลุ่มรายการขนาด *k* รายการที่ได้จากสมการที่ 3

pix แทนค่าดัชนีของกลุ่มรายการขนาด (*k*-1)รายการ (โหนดแม่ของโหนด *cix*)

ค่าดัชนี *ix* ของกลุ่มรายการขนาด 1 รายการสามารถคำนวณได้โดยตรงจากสมการที่ 1 เมื่อ $j = 0, 1, 2, ..., i$ (เลเวลของกลุ่มรายการขนาด 1 รายการ) บิทแมบความสัมพันธ์ของกลุ่มรายการขนาด 1 รายการ คือ [000…0(b_i=1)]0...0 เมื่อบิททั้งหมดมีค่าเป็น 0 ยกเว้นบิทที่ b_i ที่มีค่าเป็น 1 ดังนั้น

$$
\begin{aligned}
ix &= \sum_{j=0}^{i-1} 2^{i-j-1} \\
&= 2^{i-1} + 2^{i-2} + 2^{i-3} + ... + 2^{i-(i-2)-1} + 2^{i-(i-1)-1} \\
&= 2^{i-1} + 2^{i-2} + 2^{i-3} + ... + 2^{1} + 2^{0} \\
&= 2^{i} - 1
\end{aligned}
$$

ซึ่งการคำนวณค่าตำแหน่งนั้นถูกคำนวณแบบไดนามิกในเวลา $O(1)$ สำหรับกลุ่มรายการขนาด 1 รายการ (*a, b, c, d*) เป็นกลุ่มรายการที่จะเป็นโหนดเริ่มต้นของโครงสร้างข้อมูลที่เชื่อมต่อกับโหนดรูท [เมื่อ $x^0 + x^1 + x^2 + ... + x^n = (x^{n+1} - 1) / (x - 1)$] ดังสมการที่ 2

$$ix = \sum_{j=0}^{i=1} 2^{i-j-1} = 2^{i-1} + 2^{i-2} + \ldots + 2^1 + 2^0 = 2^i - 1 \quad (2)$$

จากตัวอย่าง (ดูข้อมูลประกอบในตารางที่ 3 และรูปที่ 3.3 (ข)) หากต้องการเข้าถึงโหนดที่มีความสัมพันธ์กันเพื่อเพิ่มความถี่ของโหนดความสัมพันธ์กันในพรีฟิคซ์อาร์เรย์จะเริ่มคำนวณค่าดัชนีของโหนดเริ่มต้น *a, b, c, d* ที่เชื่อมต่อกับโหนดรูทโดยตรงด้วยสมการที่ 2 ซึ่งจะเป็นโหนดแม่ของโหนดอื่น ๆ ต่อไป ในเลเวล 0 (level 0) ค่าดัชนีของโหนด *a* (**1**000) จึงเท่ากับ $ix = 2^0 - 1 = 0$ ในเลเวล 1 (level 1) ค่าดัชนีของโหนด *b* (**01**00) คือ $ix = 2^1 - 1 = 1$ ในเลเวล 2 (level 2) ค่าดัชนีของโหนด *c* (**001**0) คือ $ix = 2^2 - 1 = 3$ และในเลเวลสุดท้ายเลเวล 3 (level 3) ค่าดัชนีของโหนด *d* (**0001**) คือ $ix = 2^3 - 1 = 7$ ตามลำดับ

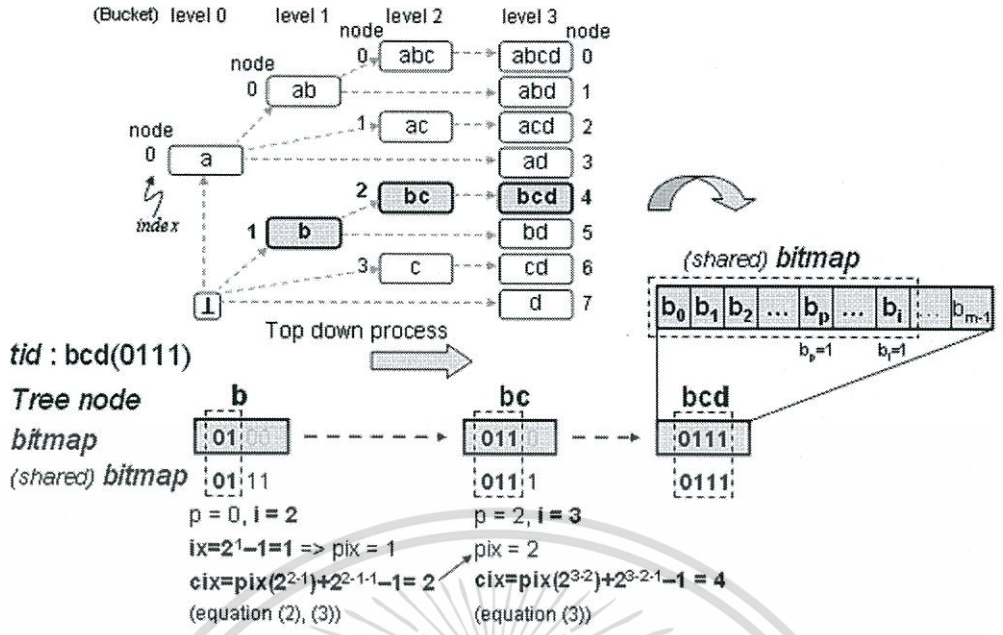
เมื่อได้ตำแหน่งของโหนดแม่แล้วก็จะคำนวณค่าดัชนีของโหนดลูก *cix* ซึ่งสามารถคำนวณต่อจากค่าดัชนีของโหนดแม่ *pix* โดยอาศัยลำดับความสัมพันธ์พรีฟิคซ์ของโหนดแม่และโหนดลูกที่คำนวณจากบิทแมบข้อมูลที่ใช้ร่วมกันจากโหนดรูท (1 รายการ) จนถึงโหนดสุดท้าย (*k* รายการ) ที่อยู่ในเส้นทางความสัมพันธ์เดียวกัน สมมุติว่าบิทแมบความสัมพันธ์ของกลุ่มรายการขนาด *k* รายการคือ $[b_0 b_1 .. b_{p-1}(b_p{=}1)0..0(b_i{=}1)]b_{i+1}..b_{m-1}$ เมื่อบิทข้อมูลต่าง ๆ อยู่ในรูปแบบ 0/1 ยกเว้น b_p ในเลเวล *p* และ b_i ในเลเวล *i* ที่มีค่าเป็น 1 เมื่อค่าดัชนีของโหนดแม่ *pix* คำนวณจากบิท b_0 จนถึงบิท b_p ในเลเวล *p* ด้วยการรวมค่าน้ำหนักของบิทที่มีค่า 0 (จากเลเวล $p{+}1$ ถึงเลเวล $i{-}1$, เมื่อ $p < i < m$) ซึ่งค่าดัชนีของโหนดลูก *cix* คำนวณได้จากการสมการที่ 1 ร่วมกับค่าดัชนีของโหนดแม่ดังนี้

$$\begin{aligned} cix &= \sum_{j=0}^{i} 2^{i-j-1} \\ &= \sum_{j=0}^{p} 2^{i-j-1} + \sum_{j=p+1}^{i-1} 2^{i-j-1} \\ &= \sum_{j=0}^{p} 2^{p-j-1}(2^{i-p}) + \sum_{j=p+1}^{i-1} 2^{i-j-1} \\ &= pix(2^{i-p}) + (2^{i-p-2} + 2^{i-p-3} + \ldots + 2^1 + 2^0) \\ &= pix(2^{i-p}) + 2^{i-p-1} - 1 \end{aligned}$$

จากการย่อสูตรทำให้สามารถคำนวณค่าดัชนีด้วยความเร็วเท่ากับ *O(1)* โดยใช้สมการที่ 3

$$cix = pix(2^{i-p}) + \sum_{j=p+1}^{i-1} 2^{i-j-1} = pix\left(2^{i-p}\right) + 2^{i-p-1} - 1 \quad (3)$$

จากตัวอย่างในรูปที่ 3.4 เป็นตัวอย่างการคำนวณค่าดัชนี *ix* ด้วยสมการที่ 2 และ *cix* ด้วยสมการที่ 3 เพื่อเพิ่มความถี่จากบิทแมบทรานแซคชัน *bcd*(0111) โดยเริ่มคำนวณค่าดัชนีของโหนด *b* ที่มีค่า $ix = 1$ ไปสู่โหนดลูก *bc* มีค่า $cix = 2$ และโหนด *bcd* มีค่า $cix = 4$ หากใช้สมการที่ 2 คำนวณค่าดัชนีให้กับโหนดของกลุ่มรายการขนาด 1 รายการและใช้สมการที่ 3 คำนวณค่าดัชนีสำหรับกลุ่มรายการขนาด *k* รายการที่เป็นโหนดลูก ดังแสดงในตารางที่ 3.3



รูปที่ 3.4 ตัวอย่างการคำนวณค่าดัชนีแบบไดนามิกของสมาชิกจากทรานแซกชั่น *bcd(0111)*

จากตารางที่ 3.2 และ 3.3 นั้นแสดงผลการค่าดัชนีของโหนดทั้งหมดที่คำนวณด้วยเวลาเท่ากับ *O(1)* ด้วยการประยุกต์ใช้สมการที่ 2 และสมการที่ 3 ซึ่งทำงานได้เร็วกว่าการคำนวณโดยใช้สมการที่ 1 โดยตรงที่ใช้เวลาเท่ากับ *O(m)* ด้วยเหตุนี้เองการเข้าถึงระหว่างโหนดแม่และโหนดลูกในโครงสร้างข้อมูลในงานวิจัยนี้จึงเท่ากับ *O(1)* โดยอาศัยความสัมพันธ์พรีฟิคซ์ทำให้สามารถใช้ค่าดัชนีของโหนดแม่ *ix* ที่ได้จากสมการที่ 2 ร่วมกับค่าดัชนีของโหนดลูก *cix* ที่ได้จากสมการที่ 3 ได้อย่างมีประสิทธิภาพ

ตารางที่ 3.3 ประสิทธิภาพการเข้าถึงโหนดแม่และโหนดลูกด้วยเวลา *O(1)* จากสมการที่ 2 และ 3

กลุ่มรายการ	บิทแมบ	โหนดแม่ :ลูก	เลเวล *p* : *i*	ค่าดัชนี *pix* : *cix*	สมการ
a	**1**000	- : *a*	-1 : 0	- : 0 [$2^0 - 1 = 0$]	(2)
b	**0**100	- : *b*	-1 : 1	- : 1 [$2^1 - 1 = 1$]	(2)
c	**00**10	- : *c*	-1 : 2	- : 3 [$2^2 - 1 = 3$]	(2)
d	**0001**	- : *d*	-1 : 3	- : 7 [$2^3 - 1 = 7$]	(2)
ab	**11**00	*a* : *b*	0 : 1	0 : 0 [$0(2^{1-0})+2^{1-0-1}$ -1=0]	(3)
abc	**111**0	*(a)b* : *c*	1 : 2	0 : 0 [$0(2^{2-1})+2^{2-1-1}$ -1=0]	(3)
abcd	**1111**	*(ab)c* : *d*	2 : 3	0 : 0 [$0(2^{3-2})+2^{3-2-1}$ -1=0]	(3)
abd	**1101**	*(a)b* : *d*	1 : 3	0 : 1 [$0(2^{3-1})+2^{3-1-1}$ -1=1]	(3)
acd	**1011**	*(a)c* : *d*	2 : 3	1 : 2 [$1(2^{3-2})+2^{3-2-1}$ -1=2]	(3)
ac	**101**0	*a* : *c*	0 : 2	0 : 1 [$0(2^{2-0})+2^{2-0-1}$ -1=1]	(3)
ad	**1001**	*a* : *d*	0 : 3	0 : 3 [$0(2^{3-0})+2^{3-0-1}$ -1=3]	(3)
bc	**011**0	*b* : *c*	1 : 2	1 : 2 [$1(2^{2-1})+2^{2-1-1}$ -1=2]	(3)
bcd	**0111**	*(b)c* : *d*	2 : 3	2 : 4 [$2(2^{3-2})+2^{3-2-1}$ -1=4]	(3)
bd	**0101**	*b* : *d*	1 : 3	1 : 5 [$1(2^{3-1})+2^{3-1-1}$ -1=5]	(3)
cd	**0011**	*c* : *d*	2 : 3	3 : 6 [$3(2^{3-2})+2^{3-2-1}$ -1=6]	(3)

3.2 ขั้นตอนการสร้างโครงสร้างข้อมูลอีบีพีเอ

การสร้างโครงสร้างข้อมูลอีบีพีเอนั้นประกอบไปด้วยสองขั้นตอนหลัก ๆ คือ 1) การเตรียมข้อมูล และ 2) การสร้างโครงสร้างข้อมูล ดังแสดงในอัลกอลิทึมที่ 3.1 สำหรับรายละเอียดการทำงานของแต่ละขั้นตอนมีรายละเอียดดังนี้

ขั้นตอนที่ 1 การเตรียมข้อมูลโดยจะเริ่มจากการโหลดข้อมูลจากฐานข้อมูลเข้ามาเก็บไว้ในหน่วยความจำพร้อมทั้งสแกนทรานแซคชันทั้งหมดเพื่อค้นหากลุ่มรายการขนาด 1 รายการ (m) ที่มีค่าความถี่มากกว่าค่าสนับสนุนขั้นต่ำ (m คือกลุ่มรายการที่เกิดบ่อย $\geq min\_sup$) และหาจำนวนทรานแซคชันทั้งหมดในฐานข้อมูล (T) จากนั้นจัดเรียงลำดับกลุ่มรายการ m ตามลำดับของค่าความถี่จากมากไปหาน้อย (a:9, b:6, c:6 และ d:5) หลังจากเรียงลำดับกลุ่มรายการเรียบร้อยแล้วจะทราบว่าแต่ละ ทรานแซคชันอยู่ในเลเวลใดของโครงสร้างข้อมูลพรีฟิคซ์ทรี โดยเลเวลของแต่ละทรานแซคชันจะขึ้นอยู่กับกลุ่มรายการ m ที่อยู่ในลำดับสุดท้ายของลิตส์ เช่น กลุ่มรายการ $abcd$ ค่า m สุดท้ายคือ d ดังนั้นกลุ่มรายการนี้จึงอยู่ในเลเวล 3 และกลุ่มรายการ abc ค่า m สุดท้ายคือ c อยู่ในเลเวล 2 เป็นต้น

อัลกอลิทึม 3.1: การสร้างโครงสร้างข้อมูลอีบีพีเอ

ขั้นตอนที่ 1: การเตรียมข้อมูล **[$O(kT)$]**

1.1 สแกนฐานข้อมูลทั้งหมดเพื่อหาข้อมูลเบื้องต้น: T (ทรานแซคชัน) k (กลุ่มรายการทั้งหมด) f (ความถี่ของแต่ละกลุ่มรายการ), l (เลเวลของแต่ละทรานแซคชัน) และ m (กลุ่มรายการที่เกิดบ่อย $\geq min\_sup$)

1.2 จัดเรียงกลุ่มรายการ m ตามความถี่จากมากไปหาน้อย

1.3 แปลงทรานแซคชันให้อยู่ในรูปแบบของบิทแมบ 0/1 (m บิท)

ขั้นตอนที่ 2: สร้างโครงสร้างข้อมูล **[$O(mT)+(mn)$]**

2.1 สแกนบิทแมบทรานแซคชันเพื่อค้นหาโหนดแรกของแต่ละทรานแซคชันและใช้สมการที่ 2 คำนวณค่าดัชนี ix

2.2 ตรวจสอบโหนดเริ่มต้นในเลเวล i หากยังไม่สร้าง ให้สร้างโหนดพร้อมกำหนดค่าเริ่มต้น (tid, ix และ w) หากมีแล้วให้เพิ่มค่าความถี่ให้กับโหนดเริ่มต้น ($++w$)

2.3 สแกนบิทต่อไปในแมบทรานแซคชันเดิมเพื่อสร้างโหนดลูกโดยใช้สมการที่ 3 หากโหนดลูกยังไม่ถูกสร้างให้สร้างโหนดลูกพร้อมกำหนดค่าให้โหนดลูก (tid, ix และ w) หากสร้างแล้วเพิ่มความถี่ ($++w$) และวนทำขั้นตอนที่ 2.3 ไปจนกว่าจะหมดทุกบิทในแถว

2.4 สแกนบิทแมบทรานแซคชันแถวถัดไปแล้วทำซ้ำขั้นตอนที่ 2.1 – 2.4 จนกว่าจะหมดทุกแถว

ขั้นตอนถัดมาสร้างตารางบิทแมบที่มีขนาดเท่ากับ $T \times m$ พร้อมทั้งกำหนดค่าเริ่มต้นให้แต่ละอาร์เรย์มีค่าเท่ากับ 0 จากนั้นทำการสแกนทรานแซคชันโดยกำหนดค่า 1 ในตารางบิทแมบในตำแหน่งที่มีกลุ่มรายการตรงกับตำแหน่งของกลุ่มรายการ m จะได้ผลลัพธ์ดังแสดงในรูปที่ 3.2 (ก) ซึ่งประกอบด้วย คอลัมน์ *tid* เป็นทรานแซคชันไอดีจากฐานข้อมูล คอลัมน์ของกลุ่มรายการตามจำนวนของกลุ่มรายการขนาด 1 รายการ (*a, b, c* และ *d*) และคอลัมน์ *level* คือเลเวลที่ทรานแซคชันนี้สังกัดอยู่นั้นเอง

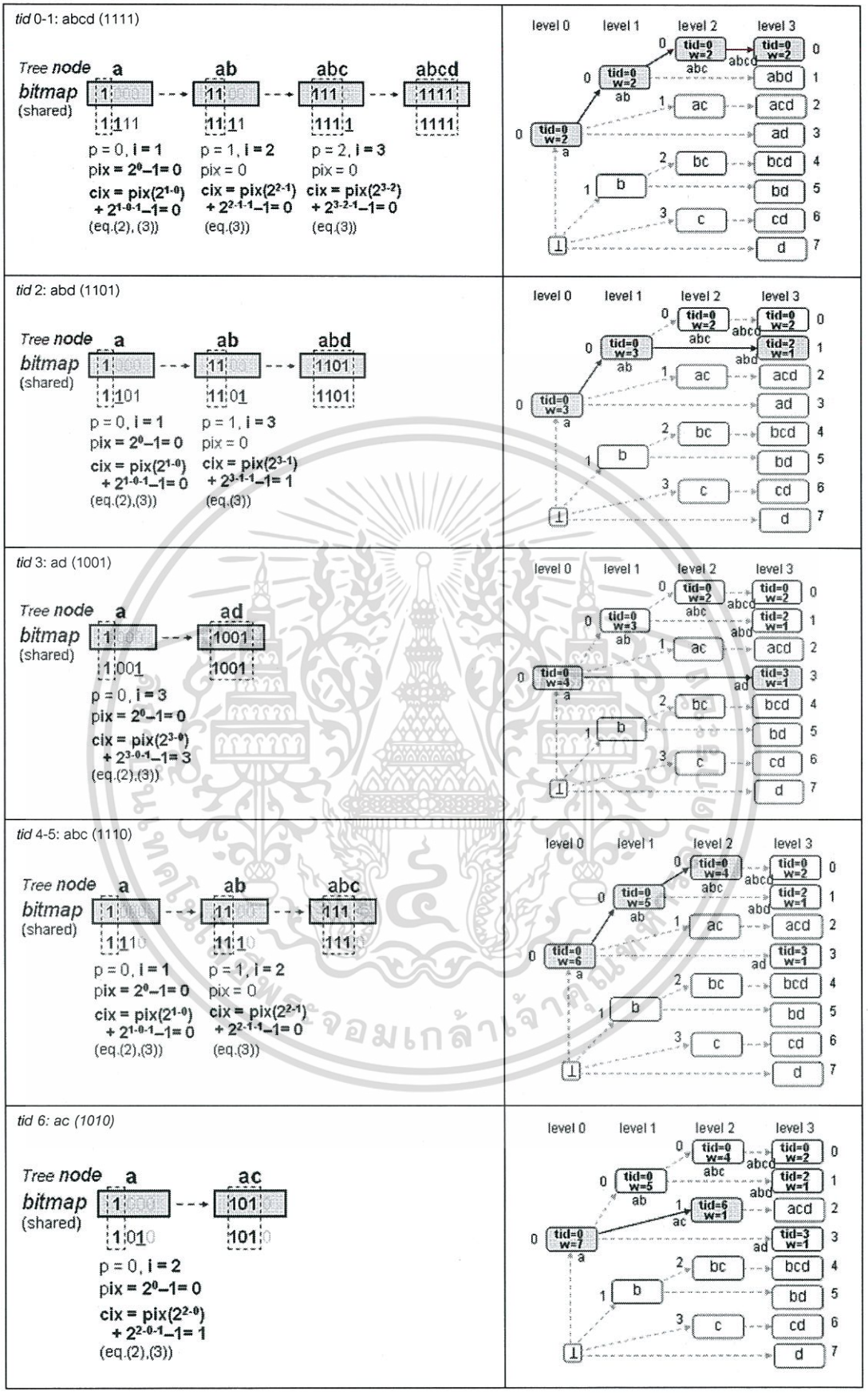
ในขั้นตอนที่ 2 การสร้างโครงสร้างข้อมูลอีบีพีเอ ในขั้นตอนนี้โครงสร้างข้อมูลแบบรวมจะใช้การจัดเรียงและรวมแถวข้อมูลก่อนการเติมความถี่ย้อนกลับจากโหนดลูกในเลเวล *m-1* ย้อนกลับไปยังโหนดแม่ในเลเวล 0 ซึ่งในขั้นตอนนี้ใช้เวลาในการทำงานนานส่งผลให้ประสิทธิภาพของวิธีการลดลง โดยเฉพาะอย่างยิ่งในกรณีที่มีจำนวน m ในปริมาณมาก ๆ ซึ่งจะส่งผลให้เลเวลในโครงสร้างข้อมูลพรีฟิคซ์ทรีมีหลายเลเวลตามไปด้วย ด้วยเหตุนี้เองโครงสร้างข้อมูลอีบีพีเอจึงได้ปรับปรุงการทำงานตรงส่วนนี้ด้วยการประยุกต์ใช้พรีฟิคซ์อาร์เรย์ร่วมกับตารางบิทแมบในการจัดเก็บข้อมูล โดยใช้การเติมความถี่ของสมาชิกแต่ละโหนดจากรูทโหนดหรือจากเลเวล 0 จนถึงเลเวล *m-1* โดยลดขั้นตอนการจัดเรียงข้อมูลและการรวมแถวทรานแซคชันข้อมูลที่ซ้ำกัน ในส่วนของเวลาในการเติมความถี่นั้นจะขึ้นอยู่กับจำนวนของกลุ่มรายการที่เกิดบ่อย ($\leq m$) ที่มีในแต่ละทรานแซคชัน โดยเฉพาะอย่างยิ่งในเรื่องของประสิทธิภาพของการติดต่อระหว่างโหนดแม่และโหนดลูก ซึ่งวิธีการในงานวิจัยนี้ได้ปรับปรุงประสิทธิภาพการติดต่อนี้ โดยใช้การคำนวณค่าดัชนีในการอ้างถึงแบบไดนามิกที่ใช้เวลาเพียง $O(1)$ ดังนั้นการเติมความถี่ของแต่ละทรานแซคชันเท่ากับ $O(m)$ และใช้เวลาในการทำงานกับทรานแซคชันทั้งหมดในฐานข้อมูลเท่ากับ $O(mT)$

ขั้นตอนที่ 2.1 เริ่มจากสแกนบิทแมบทรานแซคชันเพื่อหาโหนดเริ่มต้นของแต่ละทรานแซคชัน (บิทที่มีค่าเป็น 1 ตัวแรกของแต่ละทรานแซกชั่นถือว่าเป็นโหนดเริ่มต้น) และประยุกต์ใช้สมการที่ 2 ในการคำนวณหาค่าดัชนี ($ix = 2^i$-1) จากนั้นขั้นตอนที่ 2.2 ทำการตรวจสอบเลเวล i ของโหนดเริ่มต้นว่ามีการสร้างโหนดในเลเวลนั้นแล้วหรือยัง หากยังไม่มีก็ทำการสร้างโหนดในเลเวล i โดยในแต่ละโหนดนั้นจะจัดเก็บค่าดัชนี (*ix*) หมายเลขบิทแมบทรานแซกชั่น (*tid*) ค่าความถี่ (*w*) และกำหนดค่าความถี่เริ่มต้นให้เป็น 1 หากโหนดแรกของเลเวล i ใด ๆ ถูกสร้างแล้ว ก็จะบวกเฉพาะค่าความถี่เพิ่มขึ้นเท่านั้น (++*w*) ขั้นตอนที่ 2.3 สร้างโหนดลูกต่อจากโหนดแม่ โดยใช้ค่าดัชนีของโหนดแม่ (*pix*) เป็นหลักและสแกนบิทถัดไปเรื่อย ๆ หากเจอบิทที่มีค่าเป็น 0 จะใช้สมการที่ 3 คำนวณเก็บค่าดัชนีของโหนดลูกไปเรื่อย ๆ ($cix = pix(2^{i-p}) + 2^{i-p-1}$-1) จนกว่าจะเจอบิทที่มีค่าเป็น 1 โดยบิทที่มีค่าเป็น 1 อยู่ในเลเวลใดแสดงว่าโหนดลูกเป็นสมาชิกของเลเวลนั้น ให้วนทำงานในขั้นตอนที่ 2.3 ไปเรื่อย ๆ จนกว่าจะถึงเลเวลของทรานแซกชั่นนี้ ขั้นตอนสุดท้าย สแกนบิทแมบทรานแซคชันแถวถัดไปและวนทำงานซ้ำในขั้นตอนที่ 2.1 ถึง 2.4 ไปเรื่อย ๆ จนกว่าจะหมดทุกทรานแซคชัน

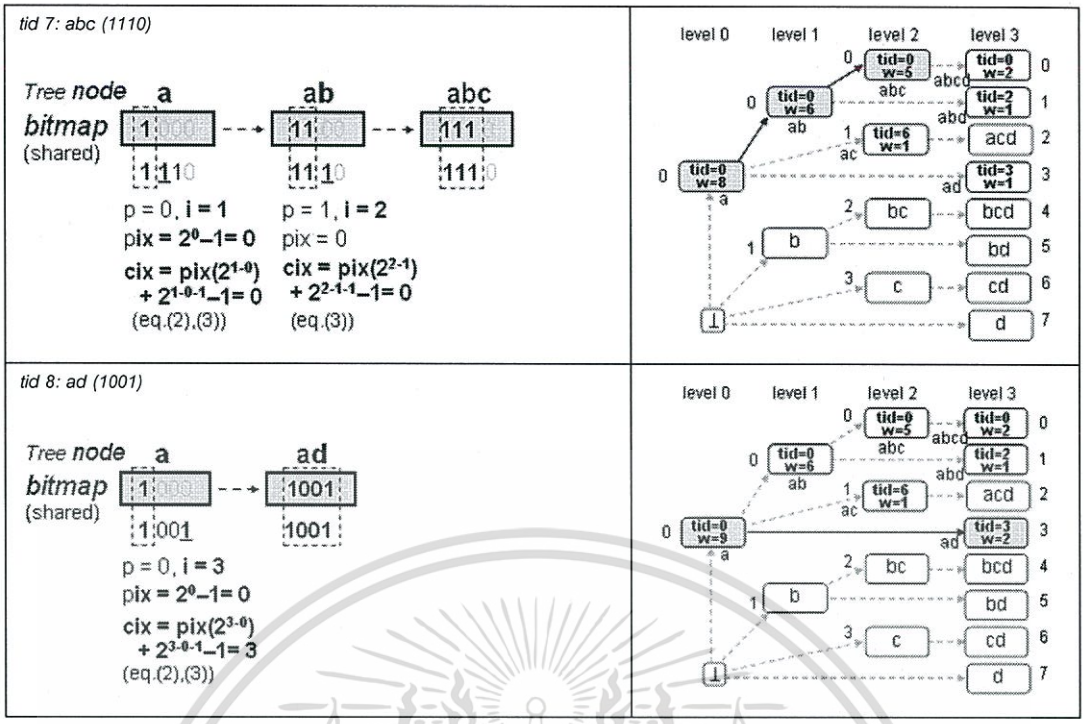
เพื่อให้เกิดความเข้าใจในขั้นตอนการสร้างโครงสร้างข้อมูลอีบีพีเอมากยิ่งขึ้น ดูตัวอย่างจากรูปที่ 3.5 ซึ่งแสดงขั้นตอนการสร้างโครงสร้างข้อมูลจากทรานแซคชันตัวอย่างในรูปที่ 3.6 (ก) โดยเริ่มจากบิทแมบทรานแซคชัน *tid* 0 (*abcd*: 1111) กลุ่มรายการแรกคือ *a* อยู่ในเลเวล 0 มีค่าดัชนีเท่ากับ $ix = 2^0\text{-}1 = 0$ เมื่อได้ข้อมูลโหนดสมาชิกจึงสร้างโหนด (*ix*=0, *tid*=0, *w*=1) ลงในเลเวล 0 จากนั้นทำการเพิ่มข้อมูลโหนดลูกต่อโดยบิทต่อไปของทรานแซกชั่นนี้อยู่ในเลเวล 1 (*b*) *pix*=0 ดังนั้นค่าดัชนีเท่ากับ $ix = cix = pix(2^{1\text{-}0}) + 2^{1\text{-}0\text{-}1}\text{-}1=0$ จากนั้นเพิ่มข้อมูลโหนดลูก (*ix*=0, *tid*=0, *w*=1) โหนดลูกลำดับถัดไปคือ *c* อยู่ในเลเวล 2 *pix*=0 ค่าดัชนีเท่ากับ $ix = cix = pix(2^{2\text{-}1}) + 2^{2\text{-}1\text{-}1}\text{-}1=0$ เมื่อได้ข้อมูลโหนดลูกแล้วทำการเพิ่มข้อมูลโหนดลูกในเลเวลที่ 2 (*ix*=0, *tid*=0, *w*=1) โหนดลูกลำดับสุดท้ายในทรานแซคชันนี้คือ *d* ในเลเวลที่ 3 *pix*=0 ค่าดัชนีคือ $cix = pix(2^{3\text{-}2}) + 2^{3\text{-}2\text{-}1}\text{-}1=0$ จากนั้นจึงสร้างโหนดข้อมูลในเลเวลที่ 3 (*ix*=0, *tid*=0, *w*=1) สำหรับทรานแซคชัน *tid* 1 มีข้อมูลซ้ำกันกับ *tid*=0 จึงบวกเพิ่มแค่ค่าความถี่ของแต่ละโหนดเท่านั้น (++*w*=2)

บิทแมบทรานแซคชันถัดมา *tid* 2 (*abd*: 1101) เริ่มต้นด้วยโหนด *a* ในเลเวล 0 เพิ่มความถี่ให้กับโหนด *a* (++*w*=3) เสร็จแล้วคำนวณค่าโหนดลูก *b* ในเลเวล 1 ซึ่งมีโหนด *a* อยู่ในเลเวล 0 เป็นโหนดแม่ ($pix = ix = 2^{0\text{-}1}$=0) ค่าดัชนีของโหนดนี้จึงเท่ากับ $cix = pix(2^{1\text{-}0}) + 2^{1\text{-}0\text{-}1}\text{-}1=0$ ซึ่งโหนดนี้มีข้อมูลอยู่แล้วจึงบวกเพิ่มแค่ค่าความถี่ (++*w* = 3) จากนั้นโหนดสุดท้ายคือ *d* อยู่ในเลเวล 3 โดยมีโหนด *b* ที่อยู่ในเลเวล 1 เป็นโหนดแม่ (*p*=1, *pix*= 0) ค่าดัชนีของโหนดนี้เท่ากับ $cix = pix(2^{3\text{-}1}) + 2^{3\text{-}1\text{-}1}\text{-}1=1$ ซึ่งโหนดนี้ยังไม่มีจึงต้องเพิ่มโหนดลงในเลเวล 3 (*ix* = 1, *tid*=2, *w*=1) จากนั้นสแกนแถวต่อไป *tid*=3 (*ad*:1001) โหนดแม่คือ *a* อยู่ในเลเวล 0 ($pix = ix = 2^0\text{-}1=0$) ซึ่งมีข้อมูลอยู่แล้วจึงบวกเพิ่มเพียงค่าความถี่ (++*w*=4) ส่วนโหนดลูกคือ *d* อยู่ในเลเวล 3 ค่าดัชนีเท่ากับ $cix = pix(2^{3\text{-}0}) + 2^{3\text{-}0\text{-}1}\text{-}1=3$ ซึ่งยังไม่มีข้อมูลในเลเวลนี้จึงเพิ่มข้อมูลของโหนดนี้เข้าไปในเลเวล 3 (*ix*=3, *tid*=3, *w*=1)

สำหรับทรานแซคชัน *tid* 4 (*abc*:1110) โหนดแม่คือ *a* อยู่ในเลเวล 0 (*ix*=0 *tid*=2 *w*=1) ซึ่งมีข้อมูลอยู่แล้วจึงบวกเพิ่มเพียงความถี่ (++*w*=5) โหนดลูกตัวแรกคือ *b* โดยมีค่า $pix = ix = 2^{0\text{-}1}$=0 และ $cix = pix(2^{1\text{-}0}) + 2^{1\text{-}0\text{-}1}\text{-}1=0$ ซึ่งโหนด *b* ก็มีอยู่แล้วเช่นกันจึงบวกเพิ่มแค่ค่าความถี่ (++*w*=4) โหนดลูกสุดท้ายคือ *c* ในเลเวล 2 โดยมีโหนด *b* (*p*=1, *pix*=0) เป็นโหนดแม่ ค่า $cix = pix(2^{2\text{-}1}) + 2^{2\text{-}1\text{-}1}\text{-}1=0$ ซึ่งโหนดนี้ก็มีข้อมูลอยู่แล้วจึงบวกเพิ่มแค่ค่าความถี่ (++*w*=3) ถัดมาทรานแซคชัน *tid* 5 (*abc*:1110) นั้นจะเหมือนกับ *tid* 4 ดังนั้นจึงทำการเพิ่มข้อมูลเหมือนกันโดยการบวกเพิ่มเพียงความถี่ของแต่ละโหนดเท่านั้น (โหนด *a* (++*w*=6), โหนด *b* (++*w*=5) และโหนด *c* (++*w*=4)) ในทรานแซคชัน *tid* 6 (*ac*:1010), *tid* 7 (*abc*:1110) และ *tid* 8 (*ad*:1001) เป็นทรานแซคชันที่มีค่ากับทรานแซคชันก่อนหน้านี้เช่นกันจึงบวกเพิ่มเพียงแค่ค่าความถี่ ดูรายละเอียดเพิ่มเติมจากรูปที่ 3.5 โดยรูปทางด้านซ้ายเป็นตัวอย่างการคำนวณค่าดัชนีของแต่ละทรานแซคชันจากข้อมูลตัวอย่างในโครงสร้างข้อมูลอีบีพีเอ ส่วนรูปทางด้านขวาเป็นตัวอย่างการจัดวางโหนดในโครงสร้างข้อมูลพรีฟิคซ์ทรีในลักษณะปกติเพื่อให้เข้าใจรูปแบบความสัมพันธ์ของแต่ละเลเวลในพรีฟิคซ์อาร์เรย์ได้ง่ายยิ่งขึ้น



รูปที่ 3.5 ตัวอย่างการสร้างโครงสร้างข้อมูลอีบีพีเอ



รูปที่ 3.5 ตัวอย่างการสร้างโครงสร้างข้อมูลอีบีพีเอ (ต่อ)

นอกจากนี้เพื่อให้ประหยัดพื้นที่หน่วยความจำ ในงานวิจัยนี้ใช้วิธีการอ้างถึงบิทแมบทรานแซคชันร่วมกันระหว่างโหนดข้อมูลที่อยู่ในเส้นทางเดียวกันเพื่อประหยัดพื้นที่ในการจัดเก็บข้อมูลของแต่ละโหนด โดยใช้การอ้างถึงหมายเลขทรานแซคชัน (*tid*) ซึ่งเป็นการประยุกต์ใช้ตำแหน่งบิทแมบทรานแซคชันในการอ้างถึงตำแหน่งสมาชิกแต่ละเลเวลในพรีฟิคซ์อาร์เรย์

หลังจากเสร็จสิ้นการเติมความถี่ของข้อมูลจากฐานข้อมูลลงในพรีฟิคซ์อาร์เรย์เรียบร้อยแล้วก็จะได้โครงสร้างข้อมูลที่พร้อมสำหรับขั้นตอนการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดต่อไป โดยในแต่ละเลเวลอาร์เรย์นั้นใช้เวลาในการคำนวณความถี่เท่ากับ $O(n)$ เมื่อ n คือจำนวนโหนดของแต่ละเลเวล ซึ่งใช้เวลาในการคำนวณความถี่ทั้งหมดเท่ากับ $O(mn)$ หากเปรียบเทียบการคำนวณความถี่ของโครงสร้างข้อมูลบิทแมบและโครงสร้างข้อมูลอาร์เรย์ลิตส์ถือว่า การคำนวณความถี่ในพรีฟิคซ์อาร์เรย์สามารถทำงานได้รวดเร็วกว่า เนื่องจากขอบเขตของรายการข้อมูลที่ใช้ในการคำนวณนั้นสั้นกว่าจึงใช้เวลาน้อยกว่านั้นเอง หากเปรียบเทียบขั้นตอนการสร้างโครงสร้างข้อมูลกับโครงสร้างข้อมูลแบบรวมที่เป็นต้นแบบในการพัฒนาในครั้งนี้ซึ่งใช้เวลาในการทำงานเท่ากับ $O(mT) + O(mn_iT')$ โดยเวลาทำงานหลักอยู่ที่ปริมาณของทรานแซคชันที่ไม่ซ้ำและจำนวนโหนดสมาชิกของเลเวล i ใด ๆ หากมีปริมาณมากก็จะใช้เวลานานขึ้น ส่วนโครงสร้างข้อมูลอีบีพีเอใช้เวลาในการทำงานเท่ากับ $O(mT) + O(mn)$ ซึ่งใช้เวลาในการทำงานที่ดีกว่าโดยเฉพาะอย่างยิ่งในกรณีที่มีจำนวนทรานแซคชันข้อมูลมากและมีปริมาณของโหนดในแต่ละเลเวลมีจำนวนน้อย

3.3 เทคนิคสำหรับเพิ่มประสิทธิภาพในการเข้าถึงระหว่างโหนดแม่และโหนดลูก

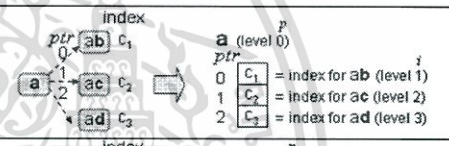
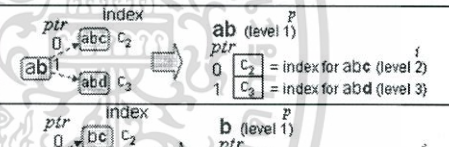
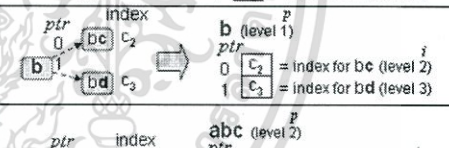
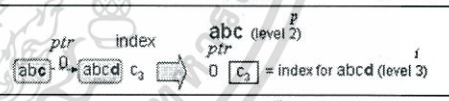
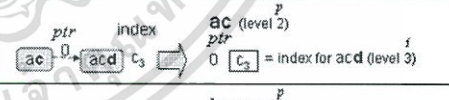
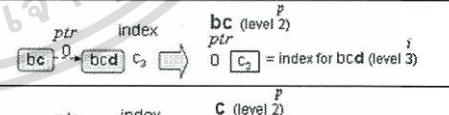
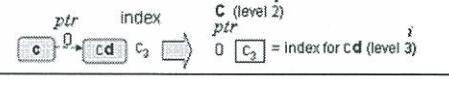
เนื่องจากวิธีการที่นำเสนอในส่วนที่ 3.1 อาจจะมีปัญหาหากกำหนดขนาดข้อมูลในหน่วยความจำเป็นชุดข้อมูลขนาดใหญ่ ซึ่งโครงสร้างข้อมูลแบบรวมให้การแบ่งข้อมูลออกเป็นชุดย่อยหลาย ๆ ชุดในกรณีที่มีกลุ่มรายการขนาด 1 รายการเกินกว่าที่จะสร้างได้ ซึ่งการแบ่งส่วนในลักษณะนั้นส่งผลให้ขั้นตอนการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดมีการทำงานที่ซับซ้อนขึ้น ในงานวิจัยนี้จึงได้ออกแบบวิธีการเข้าถึงระหว่างโหนดแม่และโหนดลูกที่สัมพันธ์กันในโครงสร้างข้อมูล โดยนำเอาพอยเตอร์อาร์เรย์ (แบบชั่วคราว) มาใช้ร่วมกับโครงสร้างข้อมูลพรีฟิคซ์อาร์เรย์ (แบบกระชับ) เพื่อเพิ่มประสิทธิภาพในการเข้าถึงที่รวดเร็วมากยิ่งขึ้น [29] โดยที่โครงสร้างข้อมูลพรีฟิคซ์อาร์เรย์มีตัวนับค่า (counter) จำนวนโหนดลูกที่มีในเลเวล i ใด ๆ เพื่อเก็บจำนวนโหนดที่เกิดขึ้นจริงในแต่ละเลเวลของอาร์เรย์บักเก็ตโดยไม่จำเป็นต้องเชื่อมโยงกันจริง ๆ ในรูปที่ 3.3 (ก) แสดงภาพตัวอย่างของการใช้โครงสร้างข้อมูลพรีฟิคซ์ทรีแบบกระชับกับกลุ่มรายการจำนวน 4 รายการ คือ *a, b, c* และ *d* ซึ่งแต่ละโหนดของโครงสร้างข้อมูลถูกสร้างโดยไม่มีการจัดเรียงโหนดตามเลเวล (level) ปกติของโครงสร้างข้อมูลพรีฟิคซ์ทรี แต่โหนดถูกสร้างจากข้อมูลที่เกิดขึ้นจริงตามทรานแซคชันที่นำมาเพิ่มข้อมูลเข้าไปในโครงสร้าง ส่วนในรูปที่ 3.3 (ข) คือโครงสร้างปกติของโครงสร้างข้อมูลพรีฟิคซ์ทรี โหนดของแต่ละเลเวลถูกจัดเรียงลำดับตามลักษณะปกติของโครงสร้างข้อมูล (จาก 0, 1, 2, …, n_i-1 โดยที่ n_i คือจำนวนของโหนดในเลเวล i (= 2^i) และ i = 0, 1, 2, …, m-1)

กำหนดให้ $b_0b_1b_2 \ldots b_{m-2}b_{m-1}$ แทน ตำแหน่งบิทของกลุ่มรายการทรานแซกชั่น *ptr* แทนพอยเตอร์อาร์เรย์ (แบบชั่วคราว) l_p แทน เลเวลของโหนดแม่ โดยที่ $l_p = p$ และ $0 \leq p \leq m\text{-}1$ l_i แทนเลเวลของโหนดลูก โดยที่ $l_i = i$ ($> p$) และ $0 \leq i \leq m\text{-}1$ c_i แทนตำแหน่งโหนดลูกในเลเวล i โหนดแม่ (p) หนึ่งโหนดจะมีโหนดลูกเท่ากับ $m\text{-}p\text{-}1$ ซึ่งจำนวนของโหนดลูกจะขึ้นอยู่กับจำนวนของกลุ่มรายการ (m) และตำแหน่งของโหนดแม่ว่าอยู่ในเลเวลใดในโครงสร้างข้อมูล โดยปกติแล้วหากโหนดแม่ต้องการติดต่อกับโหนดลูกจะต้องตรวจสอบเส้นทางของโหนดลูกที่ต้องการติดต่อจากเส้นทางของโหนดลูกที่มีอยู่ทั้งหมด หากมีโหนดลูกจำนวนมากก็จะใช้เวลาในการตรวจสอบเส้นทางนานขึ้นตามไปด้วย ดังนั้นพอยเตอร์อาร์เรย์ (*ptr*) จึงถูกออกแบบมาเพื่อช่วยในการเข้าถึงข้อมูลระหว่างโหนดแม่และโหนดลูกได้โดยตรงด้วยความเร็วในการเข้าถึงเท่ากับ $O(1)$ โดยไม่จำเป็นต้องตรวจสอบเส้นทางจากโหนดลูกที่มีอยู่ทั้งหมด ในพอยเตอร์อาร์เรย์จะเก็บลำดับของโหนดลูกด้วยการอ่านค่าจากตัวนับจำนวนโหนดลูกในเลเวลที่โหนดลูกเกิดขึ้นมาเก็บไว้เพื่อใช้อ้างอิงตำแหน่งของโหนดลูก ตำแหน่ง (location of *ptr*) ของพอยเตอร์อาร์เรย์ ($m\text{-}p\text{-}1$ รายการ) ใช้สำหรับบ่งชี้โหนดแม่ลูกแต่ละตัวนั้นเริ่มจาก 0, 1, 2, …, $m\text{-}p\text{-}2$ โดยกำหนดนิยามไว้ในสมการที่ 4

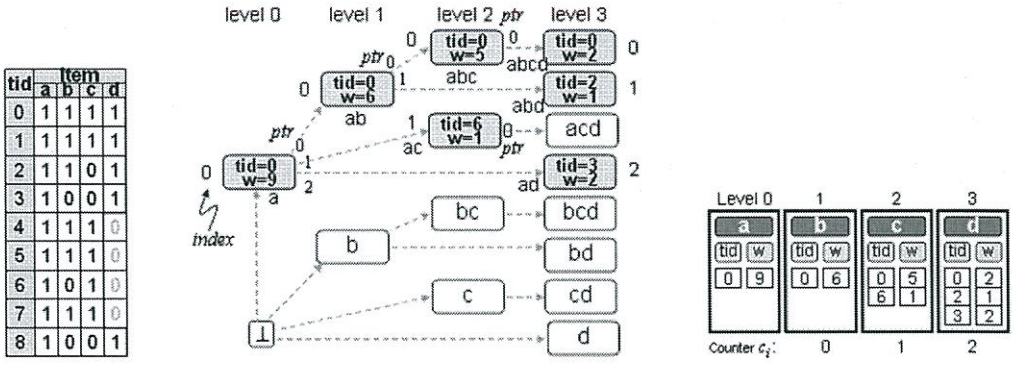
$$location\ of\ ptr\ (lptr) = l_i - l_p - 1,\ \text{โดย}\ \#childs = m - p - 1 \qquad (4)$$

โดยแต่ละแถวของบิทแมบทรานแซคชันนั้น *lptr* จะถูกคำนวณโดยใช้สมการที่ 4 ที่ใช้เวลาในการทำงานเท่ากับ *O(1)* เพื่อชี้ไปยังโหนดที่ต้องการติดต่อโดยตรง ซึ่งเป็นค่าความสัมพันธ์กันระหว่างเลเวลของโหนดลูก ($l_i = i$) และโหนดแม่ ($l_p = p$) ดังแสดงตัวอย่างในตารางที่ 3.4 ลำดับในพอยเตอร์อาร์เรย์นั้นจะอ้างอิงไปยังเลเวลของลูกที่จะเกิดขึ้น เช่น หากโหนดแม่อยู่ในเลเวลที่ 1 (level 1) อาร์เรย์ 0 เก็บค่าตำแหน่งของโหนดลูกในเลเวลที่ 2 (level 2) อาร์เรย์ 1 เก็บค่าตำแหน่งของโหนดลูกในเลเวลที่ 3 (level 3) ส่วนค่าตำแหน่งของโหนดลูกที่เก็บในพอยเตอร์อาร์เรย์แต่ละอาร์เรย์นั้นจะเป็นค่าใดนั้นก็ขึ้นอยู่กับว่าโหนดลูกโหนดนี้ถูกสร้างขึ้นเป็นลำดับที่เท่าไหร่ในเลเวลนั้น ๆ (c_i) เช่น ในอาร์เรย์ 0 เก็บค่า 1 เอาไว้ก็แสดงว่าโหนดนี้ถูกสร้างขึ้นในลำดับที่ 1 นั้นเอง

ตารางที่ 3.4 ตัวอย่างพอยเตอร์อาร์เรย์ของโหนดทั้งหมดในรูปที่ 3.3 (ก)

โหนด	บิทแมบ	เลเวล p	l_i=i	ตำแหน่งของ *ptr*	ตัวอย่างการคำนวณ
A	**1**000	0	i=1 i=2 i=3	*lptr* =1-0-1 = 0 *lptr* =2-0-1 = 1 *lptr* =3-0-1 = 2	
Ab	**11**00	1	i=2 i=3	*lptr* =2-1-1 = 0 *lptr* =3-1-1 = 1	
B	**01**00	1	i=2 i=3	*lptr* =2-1-1 = 0 *lptr* =3-1-1 = 1	
Abc	**111**0	2	i=3	*lptr* =3-2-1 = 0	
Ac	**101**0	2	i=3	*lptr* =3-2-1 = 0	
Bc	**011**0	2	i=3	*lptr* =3-2-1 = 0	
C	**001**0	2	i=3	*lptr* =3-2-1 = 0	

พอยเตอร์อาร์เรย์แบบชั่วคราวช่วยเพิ่มประสิทธิภาพในการทำงานมากยิ่งขึ้น ด้วยการใช้ตำแหน่งที่ตั้ง (*lptr* ที่ได้จากสมการที่ 4) ร่วมกับค่าตัวนับจำนวนโหลดลูกในเลเวล i ใด ๆ ในการเข้าถึงโหนดลูกได้โดยตรงโดยไม่ใช้ตารางแฮส และไม่จำเป็นจะต้องค้นหาเส้นทางของโหนดที่ต้องการติดต่อจากเส้นทางของโหนดลูกทั้งหมด โดยค่าตำแหน่งโหนดลูกเริ่มต้นของแต่ละเลเวลจะถูกตั้งค่าเป็น -1 เมื่อมีโหนดเกิดขึ้นก็จะเก็บค่าตำแหน่งที่ได้จากการคำนวณและบวกเพิ่มค่าตัวนับจำนวนโหนดขึ้นที่ละ 1 ไปเรื่อย ๆ ตามจำนวนโหนดที่เกิดขึ้นในแต่ละเลเวล โดยหลังจากเสร็จสิ้น



(ก) (ข) (ค)

รูปที่ 3.6 (ก) ตารางบิทแมป (ข) ตัวอย่างโครงสร้างอีบีพีเอในรูปแบบโครงสร้างข้อมูลพรีฟิคซ์ทรีแบบกระชับ (ค) โครงสร้างข้อมูลอีบีพีเอในงานวิจัยนี้ที่ใช้งานร่วมกับตารางบิทแมป

ขั้นตอนการสร้างโครงสร้างข้อมูลก็จะลบพอยเตอร์อาร์เรย์แบบชั่วคราวออกจากหน่วยความจำ เพื่อเป็นการคืนพื้นที่หน่วยความจำเตรียมไว้สำหรับเก็บข้อมูลในการค้นหากลุ่มรายการแบบปิดต่อไป

โครงสร้างข้อมูลอีบีพีเอใช้โครงสร้างข้อมูลพรีฟิคซ์อาร์เรย์แบบกระชับเป็นหลัก โดยจัดเก็บเฉพาะข้อมูลที่เกิดขึ้นจริงและได้แบ่งข้อมูลที่สัมพันธ์กันออกเป็นเลเวลเหมือนเลเวลของโครงสร้างข้อมูลพรีฟิคซ์ทรี โดยคำนึงถึงประสิทธิภาพในการจัดเก็บข้อมูลในหน่วยความจำและเวลาในการประมวลผลเป็นหลัก สำหรับเวลาที่ใช้ในเพื่อเพิ่มความถี่ให้กับโหนดลูกของแต่ละทรานแซกชั่นเท่ากับ $O(m)$ ในเส้นทางที่มีโหนดอยู่แล้วและเท่ากับ $O(m^2)$ ในเส้นทางที่ยังไม่มีการสร้างโหนดเอาไว้ เวลาที่ใช้สำหรับทรานแซกชั่นทั้งหมดเท่ากับ $O(m^2n) + O(mT)$ เมื่อ $n = max(n_i)$ และ $n \leq 2^i$, $i = 1, 2, ..., m - 1$

เพื่อให้ง่ายต่อการทำความเข้าใจในโครงสร้างข้อมูลอีบีพีเอ (รูปที่ 3.6 (ค)) จึงจัดโครงสร้างอีบีพีเอให้อยู่ในรูปแบบของโครงสร้างข้อมูลพรีฟิคซ์ทรีแบบกระชับให้ดูในรูปที่ 3.6 (ข) โดยใช้รูปแบบของความสัมพันธ์ระหว่างโหนดแม่และโหนดลูกเช่นเดียวกัน ซึ่งจำนวนของเลเวลในโครงสร้างข้อมูลนั้นจะเท่ากับจำนวนของกลุ่มรายการขนาด 1 รายการ (เช่น *a, b, c, d* จะมี 4 เลเวล) ในแต่ละโหนดจะจัดเก็บเฉพาะค่าความถี่หรือค่าสนับสนุน (*weight w*) และหมายเลขทรานแซคชัน (*tid*) ที่ใช้อ้างอิงถึงตารางบิทแมทในรูปที่ 3.6 (ก) ที่ออกแบบให้ใช้ข้อมูลร่วมกันในโครงสร้างที่มีความ สัมพันธ์กันเพื่อประหยัดพื้นที่ในหน่วยความจำ เช่น 1111 แทนกลุ่มรายการ *abcd* (4 บิท) ในเลเวลที่ 3 ถ้าอ่านข้อมูลเพียง 3 บิทแรก (111 : *abc*) ก็จะแทนข้อมูลในเลเวลที่ 2 หากอ่านข้อมูลเพียง 2 บิทแรก (11 : *ab*) ก็จะเป็นข้อมูลในเลเวลที่ 1 และหากอ่านข้อมูลเพียงบิทแรก (1 : *a*) ก็จะหมายถึงข้อมูลในเลเวลที่ 0 ของโครงสร้างข้อมูลนั้นเอง ด้วยความสัมพันธ์นี้เราจึงใช้บิทแมบทรานแซคชันร่วมกันได้ ขนาดของตารางบิทแมบเท่ากับ $T \times m$ โดยที่ T คือจำนวนทรานแซคชันทั้งหมดในฐาน ข้อมูล และ m คือจำนวนกลุ่มรายการขนาด 1 รายการที่มีความถี่ผ่านค่าสนับสนุนขั้นต่ำ

3.4 ขั้นตอนการสร้างโครงสร้างข้อมูลอีบีพีเอด้วยเทคนิคเพิ่มประสิทธิภาพการเข้าถึง

การสร้างโครงสร้างข้อมูลอีบีพีเอด้วยเทคนิคเพิ่มประสิทธิภาพการเข้าถึงระหว่างโหนดแม่และโหนดลูกประกอบไปด้วยขั้นตอนหลัก 2 ขั้นตอนเช่นเดียวกัน คือ 1) การเตรียมข้อมูล และ 2) การสร้างโครงสร้างข้อมูล ดังแสดงในอัลกอลิทึมที่ 3.2 รายละเอียดของแต่ละขั้นตอนมีดังนี้

ขั้นตอนที่ 1 การเตรียมข้อมูลโดยจะเริ่มจากการโหลดไฟล์ข้อมูลจากฐานข้อมูลเข้ามาเก็บไว้ในหน่วยความจำพร้อมทั้งสแกนทรานแซคชันทั้งหมดเพื่อค้นหากลุ่มรายการที่เกิดบ่อย m (กลุ่มรายการขนาด 1 รายการที่มีความถี่ $\geq$ min_sup) และหาจำนวนทรานแซคชันทั้งหมด (T) ในฐานข้อมูล จากนั้นทำการเรียงลำดับกลุ่มรายการ m ตามลำดับของค่าความถี่จากมากไปหาน้อย (a:9 b:6, c:6 และ d:5) หลังจากเรียงลำดับกลุ่มรายการเรียบร้อยแล้วจะทราบว่าแต่ละทรานแซคชันอยู่ในเลเวลใดของโครงสร้างข้อมูลพรีฟิคซ์ทรีซึ่งจะนำเลเวลที่ได้ไปใช้เป็นตำแหน่งอ้างอิงในพรีฟิคซ์อาร์เรย์ต่อไป จากนั้นทำการแปลงข้อมูลในอยู่ในรูปแบบบิทแมบที่มีขนาด $T \times m$ พร้อมทั้งกำหนด

อัลกอลิทึม 3.2: การสร้างโครงสร้างข้อมูลอีบีพีเอด้วยเทคนิคเพิ่มประสิทธิภาพ

ขั้นตอนที่ 1: การเตรียมข้อมูล ***O(kT)***]

1.1 สแกนฐานข้อมูลทั้งหมดเพื่อหาข้อมูลเบื้องต้น: T (ทรานแซคชัน) k (กลุ่มรายการทั้งหมด) f (ความถี่ของแต่ละกลุ่มรายการ), l (เลเวลของแต่ละทรานแซคชัน) และ m (กลุ่มรายการที่มีความถี่ $\geq$ min_sup)

1.2 ทำการจัดเรียงกลุ่มรายการ m ตามความถี่จากมากไปหาน้อย

1.3 แปลงทรานแซคชันให้อยู่ในรูปแบบบิทแมบ 0/1 (m-บิท)

ขั้นตอนที่ 2: สร้างโครงสร้างข้อมูล [$O(m^2T)+(mT)$]

2.1 สแกนบิทแมบทรานแซคชันเพื่อค้นหาโหนดแรกในทรานแซคชัน

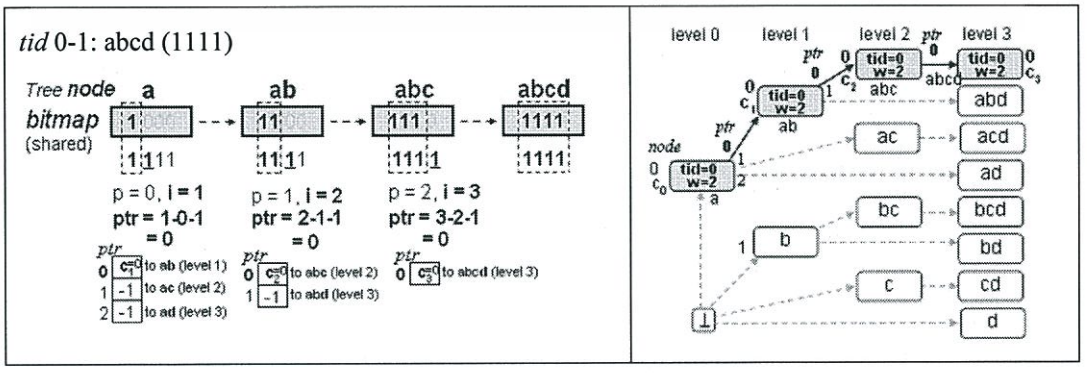
2.2 ตรวจสอบโหนดเริ่มต้นในเลเวล i หากยังไม่มีโหนด ให้สร้างโหนดพร้อมกำหนดค่าเริ่มต้น (tid w=1 และ ptr) และกำหนดค่าตัวนับเริ่มต้น(++c) ในเลเวล i หากมีโหนดอยู่แล้วเพิ่มค่าความถี่ให้กับโหนดเริ่มต้น (++w)

2.3 สแกนบิทต่อไปเพื่อสร้างโหนดลูก i โดยตรวจสอบกับข้อมูลพอยเตอร์อาร์เรย์ของโหนดแม่ว่ามีข้อมูลโหนดลูกแล้วหรือยัง หากยังให้สร้างโหนดลูกในพรีฟิคซ์อาร์เรย์พร้อมทั้งกำหนดค่าเริ่มต้น (tid และ w=1) หากถูกสร้างแล้วเพิ่มความถี่ (++w) วนทำงานในขั้นตอนที่ 2.3 ไปจนกว่าจะหมดข้อมูลในบิทแมบทรานแซคชัน

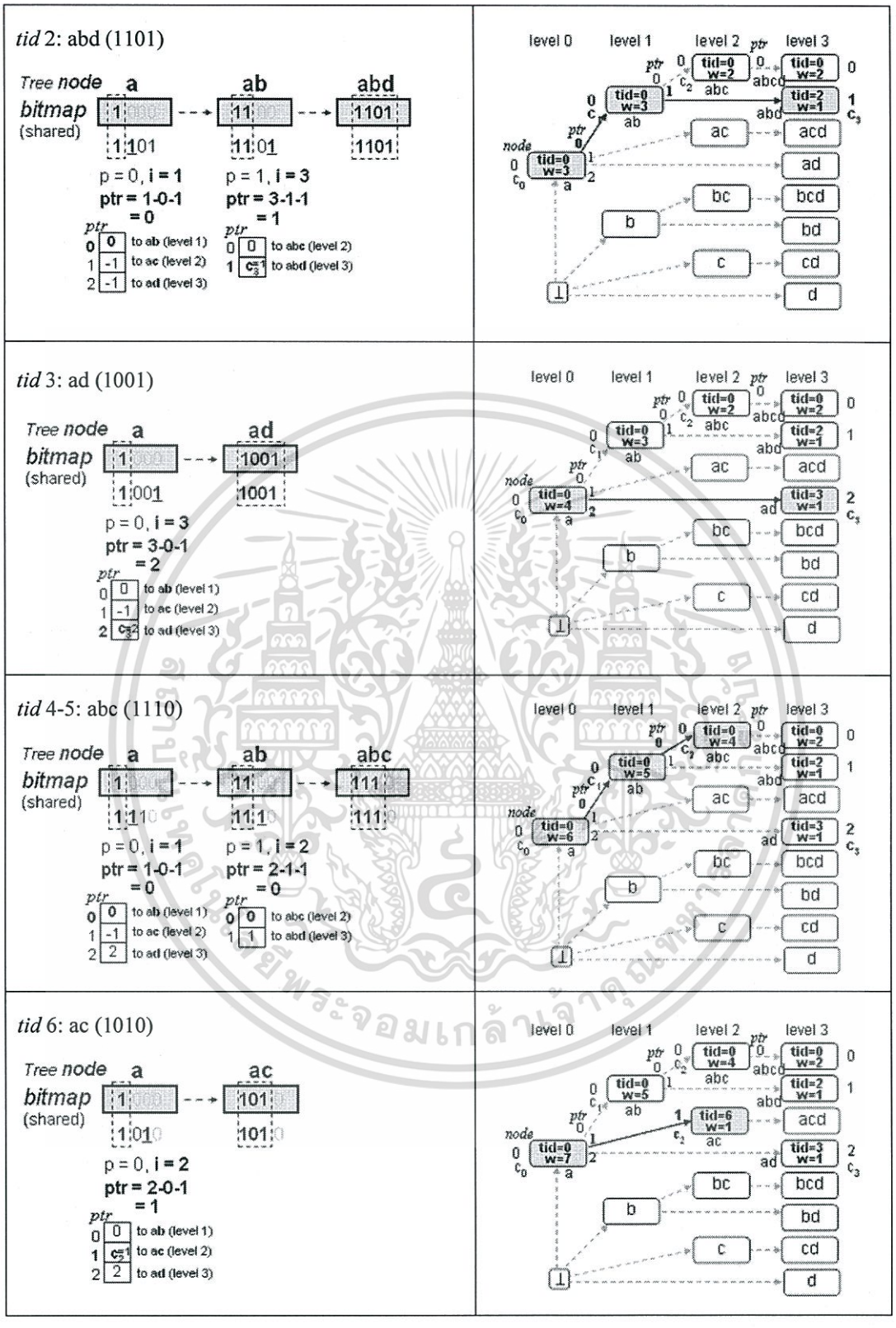
2.4 สแกนบิทแมบทรานแซคชันแถวถัดไปแล้วทำซ้ำขั้นตอนที่ 2.1 – 2.4 จนกว่าจะหมดทุกแถว

ค่าเริ่มต้นในแต่ละอาร์เรย์เท่ากับ 0 จากนั้นทำการสแกนทรานแซคชันโดยกำหนดค่า 1 ในตารางบิทแมบในตำแหน่งที่มีกลุ่มรายการตรงกับตำแหน่งของกลุ่มรายการ *m* จะได้ผลลัพธ์ดังแสดงในรูปที่ 3.6 (ก) ซึ่งประกอบไปด้วยคอลัมน์ *tid* เก็บหมายเลขทรานแซคชันจากฐานข้อมูล คอลัมน์กลุ่มรายการที่ประกอบด้วยข้อมูลกลุ่มรายการขนาด 1 รายการ (*a, b, c* และ *d*) ในแต่ละทรานแซกชั่นจะเก็บข้อมูลเลเวลเอาไว้ด้วยเพื่อให้ทราบตำแหน่งบิทสุดท้ายในบิทแมบทรานแซกชั่น

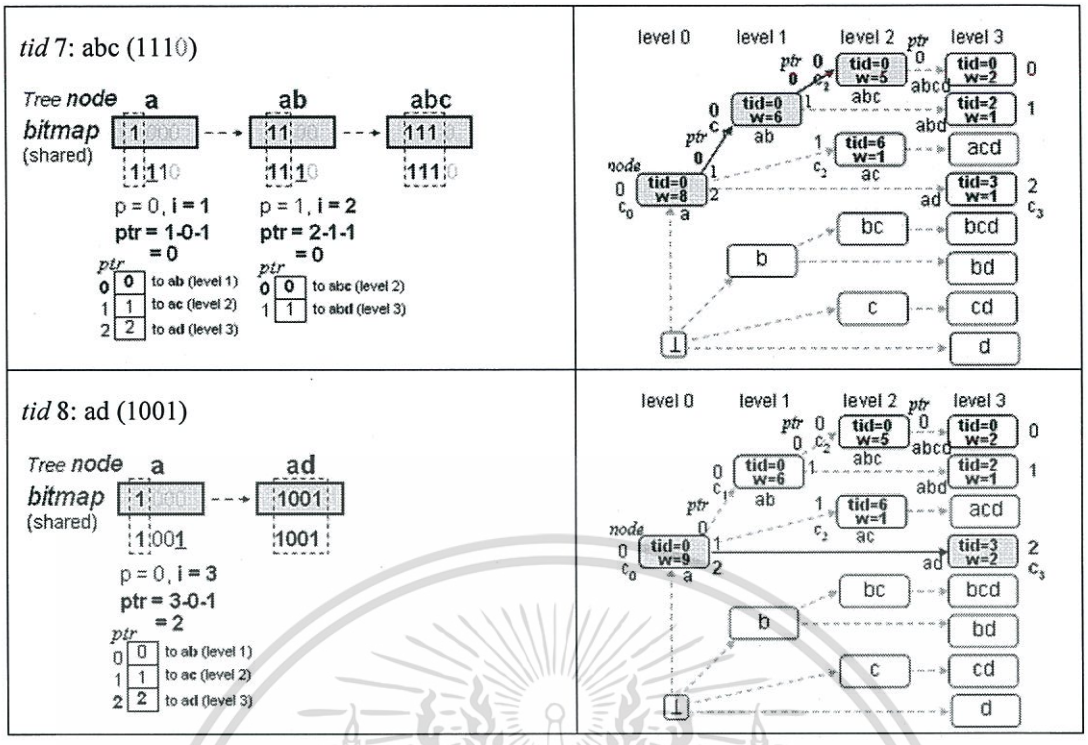
ขั้นตอนที่ 2 คือการสร้างโครงสร้างข้อมูลอีบีพีเอ โดยได้ปรับปรุงการทำงานตรงส่วนนี้ด้วยการประยุกต์ใช้พรีฟิคซ์อาร์เรย์ร่วมกับพอยเตอร์อาร์เรย์แบบชั่วคราวเพื่อเติมความถี่ของสมาชิกในแต่ละโหนดจากรูทโหนดหรือจากเลเวล 0 จนถึงเลเวล *m-1* โดยลดขั้นตอนการจัดเรียงข้อมูลและรวมแถวทรานแซคชันข้อมูลที่ซ้ำกันลง โดยเริ่มจากขั้นตอนที่ 2.1 สแกนบิทแมบทรานแซคชันแถวแรกเพื่อหาบิทเริ่มต้นของแถว (บิทที่มีค่า 1 ตัวแรกของแถว) ขั้นตอนที่ 2.2 หลังจากได้เลเวลของบิทแรกให้ตรวจสอบว่าในเลเวลที่ได้มานั้นมีโหนดแรกแล้วหรือยัง หากยังไม่มีให้เพิ่มค่าในตัวนับจำนวนของเลเวลที่ได้และเชื่อมโยงโหนดข้อมูลกับพอยเตอร์อาร์เรย์ชั่วคราว พร้อมทั้งสร้างโหนดแรกลงในเลเวลที่ค้นพบและกำหนดข้อมูลต่าง ๆ (*tid, w* และ *ptr*) ให้กับโหนดแรก หากโหนดแรกถูกสร้างไปแล้วให้เพิ่มเพียงค่าความถี่ (*++w*) ขั้นตอน 2.3 ให้สแกนหาบิทที่มีค่าเป็น 1 ตัวถัดไปเพื่อสร้างโหนดลูก เมื่อได้เลเวลของโหนดลูกก็นำเอาข้อมูลที่ได้ไปตรวจสอบจากข้อมูลของโหนดแม่ว่าในเลเวลของโหนดลูกที่ได้มานั้นมีการสร้างโหนดลูกไปแล้วหรือยัง หากยังไม่มีการสร้าง ก็ให้สร้างโหนดลูกลงในเลเวลที่ได้พร้อมทั้งกำหนดข้อมูลต่าง ๆ (*tid, w* และ *ptr*) ให้กับโหลดลูก พร้อมทั้งทำการเชื่อมโยงกับพอยเตอร์อาร์เรย์ชั่วคราว หากในกรณีที่มีข้อมูลโหนดลูกแล้วก็บวกเพิ่มเพียงค่าความถี่เท่านั้น จากนั้นให้วนทำงานในขั้นตอนที่ 2.3 ไปเรื่อย ๆ จนถึงบิทที่อยู่ในเลเวลสุดท้ายของแถว สุดท้ายขั้นตอนที่ 2.4 ให้สแกนบิทแมบทรานแซคชันแถวถัดไปแล้ววนทำงานในขั้นตอนที่ 2.1 - 2.4 ไปเรื่อย ๆ จนกว่าจะหมดทุกทรานแซคชันเพียงเท่านี้ก็จะเสร็จสิ้นขั้นตอนการสร้างโครงสร้างข้อมูล ดูจากรูปตัวอย่างในตารางด้านล่างนี้ นอกจากนี้ในแต่ละเลเวลจะจัดเก็บข้อมูลเลเวลของโหนดลูกที่เชื่อมโยงโดยตรงเอาไว้ด้วยเพื่อความรวดเร็วในการตรวจสอบความซ้ำซ้อนในขั้นตอนการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด สำหรับตัวอย่างขั้นตอนการสร้างนั้น



รูปที่ 3.7 ตัวอย่างการสร้างโครงสร้างข้อมูลอีบีพีเอด้วยเทคนิคเพิ่มประสิทธิภาพ



รูปที่ 3.7 ตัวอย่างการสร้างโครงสร้างข้อมูลอีบีพีเอด้วยเทคนิคเพิ่มประสิทธิภาพ (ต่อ)



รูปที่ 3.7 ตัวอย่างการสร้างโครงสร้างข้อมูลอีบีพีเอด้วยเทคนิคเพิ่มประสิทธิภาพ (ต่อ)

โครงสร้างข้อมูลอีบีพีเอในงานวิจัยนี้เวลาในการทำงานเท่ากับ $O(mT) + O(mn)$ เวลาในการทำงานขึ้นอยู่กับปริมาณของโหนดสมาชิกในแต่ละเลเวลเป็นหลัก ส่วนโครงสร้างข้อมูลอีบีพีเอที่พัฒนาด้วยเทคนิคการเพิ่มประสิทธิภาพอีกหนึ่งวิธีที่พัฒนาขึ้นในงานวิจัยนี้ใช้เวลาทำงานเท่ากับ $O(m^2T) + O(mT)$ ซึ่งหากเปรียบเทียบเวลาในการทำงานจะเห็นว่าวิธีการแรกจะมีประสิทธิภาพมากกว่าแต่จะมีปัญหาในเรื่องของการจองพื้นที่ในหน่วยความจำ หากมีข้อมูลในปริมาณมากจะมีปัญหาในการจองพื้นที่ข้อมูลให้มีขนาดใหญ่ตามไปด้วย ส่วนวิธีที่สองนั้นออกแบบมาเพื่อให้ง่ายต่อการแก้ปัญหาดังกล่าว ซึ่งหากมีข้อมูลจำนวนมากนั้นวิธีการที่สองสามารถทำงานโดยแบ่งพอยเตอร์อาร์เรย์ชั่วคราวออกในแนวนอนเป็นก้อน ๆ ได้ ซึ่งจะช่วยแก้ปัญหาในการจองพื้นที่หน่วยความจำเป็นก้อนเดียวที่มีขนาดใหญ่ได้

หากเปรียบเทียบการทำงานโครงสร้างข้อมูลแบบรวมนั้นหากในบางฐานข้อมูลที่ปริมาณของทรานแซคชันซ้ำกันมีน้อยกว่าปริมาณของโหนดสมาชิกอาจจะส่งผลให้โครงสร้างข้อมูลแบบรวมทำงานได้ดีว่าโครงสร้างข้อมูลอีบีพีเอและหากปริมาณของโหนดในแต่ละเลเวลมีจำนวนน้อยกว่าจะส่งผลให้โครงสร้างข้อมูลอีบีพีเอสามารถทำงานได้ดีกว่า ซึ่งโดยปกติของข้อมูลแล้วจำนวนของโหนดจะมีน้อยกว่าจำนวนทรานแซคชันที่ไม่ซ้ำกันอยู่แล้วจึงส่งผลให้วิธีการในงานวิจัยนี้มีแนวโน้มการทำงานที่เร็วกว่าในทุก ๆ ฐานข้อมูล

บทที่ 4
การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด

การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดเป็นเทคนิคสำคัญที่ถูกนำมาประยุกต์ใช้ในงานด้านต่าง ๆ ของการทำเหมืองข้อมูล และมีอัลกอลิทึมที่เกี่ยวข้องออกมาหลากหลายวิธีการ โดยปัจจัยสำคัญที่จะทำให้วิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดสามารถทำงานได้อย่างเต็มประสิทธิภาพก็คืออัลกอลิทึมที่มีประสิทธิภาพและโครงสร้างข้อมูลที่เอื้อต่อการคำนวณหาค่าความถี่ ซึ่งเป็นส่วนที่มีการทำงานหนักที่สุด ด้วยเหตุนี้เองงานวิจัยชิ้นนี้จึงได้ออกแบบโครงสร้างข้อมูลอีบีพีเอขึ้นมาเพื่อใช้ในการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด โดยได้อธิบายองค์ประกอบและการทำงานไว้แล้วในบทที่ 3 สำหรับบทนี้จะกล่าวถึงวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดโดยใช้วิธีพีพีซีเอ็กเทนชั่นและ LCM-3 เป็นต้นแบบในการพัฒนาประสิทธิภาพ ซึ่งเป็นวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดที่ใช้การทำงานแบบบนลงล่าง โดยไม่จำเป็นจะต้องเก็บข้อมูลกลุ่มรายการที่เกิดบ่อยแบบปิดที่ค้นพบก่อนหน้านี้เอาไว้เพื่อตรวจสอบความซ้ำซ้อนของข้อมูล ซึ่งเป็นวิธีที่มีประสิทธิภาพ โดยเรียกวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดว่า EBPA-CLOSED โดยได้ปรับปรุงวิธีในการคัดกรองข้อมูลด้วยเทคนิคที่เรียกว่าพรีเทส (pre-test) เพื่อให้สามารถลดปริมาณของข้อมูลที่ใช้สำหรับตรวจสอบว่ากลุ่มรายการที่เป็นตัวสร้าง (generator) สามารถที่จะเป็นกลุ่มรายการที่เกิดบ่อยแบบปิดได้หรือไม่ โดยจะคัดกรองกลุ่มรายการที่เป็นโคลสเซอร์เซต (closure set คือกลุ่มรายการที่เป็นพรีฟิคซ์ของตัวสร้าง) และกลุ่มรายการที่เป็นโพสเซต (post set คือกลุ่มรายการที่เป็นซัฟฟิคซ์ของตัวสร้าง) ให้เหลือเฉพาะข้อมูลที่จำเป็นจะต้องตรวจสอบจริง ๆ ทำให้ใช้เวลาในการทำงานน้อยลง ซึ่งได้อธิบายไว้ในบทนิยามที่ 4.1 และ 4.2

จากอัลกอลิทึมที่ 3 เป็นวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดที่ออกแบบให้มีประสิทธิภาพมากยิ่งขึ้น โดยจะทำงานร่วมกับโครงสร้างข้อมูลอีบีพีเอที่ใช้การแบ่งข้อมูลออกตามเลเวลของข้อมูล ซึ่งเป็นข้อมูลที่มีความสัมพันธ์กันจึงเอื้อต่อการคำนวณค่าความถี่ทำให้สามารถทำงานได้รวดเร็วยิ่งขึ้น การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดในงานวิจัยชิ้นนี้ถูกแบ่งออกเป็น 2 ขั้นตอนคือ 1) การคัดกรองข้อมูล และ 2) การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด

ขั้นตอนที่ 1 การคัดกรองข้อมูลเป็นขั้นตอนการจัดเตรียมข้อมูลที่มีความสัมพันธ์กับข้อมูลที่เป็นกลุ่มรายการเริ่มต้นในการค้นหาขนาด 1 รายการหรือกลุ่มรายการ m นั้นเอง ซึ่งประกอบไปด้วยข้อมูลของตัวสร้าง ข้อมูลของโคลสเซอร์เซต และข้อมูลของโพสเซต กลุ่มรายการเหล่านี้จะถูกใช้สำหรับตรวจสอบระหว่างตัวสร้างกลับข้อมูลโคลสเซอร์เซตและโพสเซตของมันว่าสามารถที่จะเป็นกลุ่มรายการที่เกิดบ่อยแบบปิดได้หรือไม่ ข้อมูลนี้ยังถูกใช้ในการหากลุ่มรายการที่เกิดบ่อยแบบปิดที่เป็นซัฟฟิคซ์ (suffix closed itemset) ของตัวสร้างขนาด 1 รายการไปเรื่อย ๆ จนกว่าจะหมด

อัลกอลิทึม 2: EBPA-CLOSED.

ขั้นตอนที่ 1: การคัดกรองข้อมูลใส่ข้อมูลเริ่มต้น (Initial Data) **[*O(mn)*]**

1.1 สำหรับตัวสร้างขนาด 1 รายการ แต่ละตัวจะทำการสแกนและนับค่าความถี่โดยตรงจากเลเวลของตัวสร้างเองเพื่อคัดกรองข้อมูลไปเก็บไว้ในข้อมูลเริ่มต้น ($n_i \leq n$) โดย n_i คือจำนวนโหนดในเลเวล i ($\leq 2^i$) และ $n = max(n_i)$, $0 \leq i \leq m\text{-}1$ พร้อมทั้งเก็บข้อมูลโคลสเซอร์เซต

1.2 สแกนหาโพสเซตโดยคัดกรองเฉพาะข้อมูลที่สัมพันธ์กันกับตัวสร้างจากเลเวลที่สูงกว่ากลุ่มรายการที่เป็นคีย์หลัก (core item) ของตัวสร้าง

ขั้นตอนที่ 2: การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด **[*O(mn)*]**

2.1 สร้างตัวสร้างจากกลุ่มรายการเริ่มต้นขนาด 1 รายการรวมกับข้อมูลที่เป็นโคลสเซอร์เซต เปรียบเทียบความถี่ของตัวสร้างว่า $\geq min\_supp$ หากผ่านทำขั้นตอนต่อไป

2.2 เปรียบเทียบความถี่ของตัวสร้างกับกลุ่มรายการที่เป็นโพสเซตหากความถี่เท่ากันแสดงว่าตัวสร้างไม่เป็นกลุ่มรายการที่เกิดบ่อยแบบปิด หากความถี่ไม่เท่ากันกับกลุ่มรายการที่เป็นโพสเซต ตัวสร้างนี้จะเป็นกลุ่มรายการที่เกิดบ่อยแบบปิด

2.3 วนทำงานจากขั้นตอนที่ 2.1 – 2.3 ไปเรื่อย ๆ จนกว่าจะหมดข้อมูลในกลุ่มโคลสเซอร์เซต

ขั้นตอนที่ 3: วนทำงานค้นหากลุ่มรายการที่เกิดบ่อยแบบปิด *m* ตัวต่อไป **[$O(m^2n)$]**

วนทำงานกับตัวสร้างขนาด 1 รายการไปเรื่อย ๆ จนกว่าจะหมดข้อมูล (m-1)

นิยามที่ 4.1 *กำหนดให้ P_l เป็นตัวสร้าง โคลสเซอร์เซตของ P_l คือ กลุ่มรายการที่มีความสัมพันธ์กับ P_l และอยู่ในพรีฟิคซ์อาร์เรย์เลเวลที่ต่ำกว่ากลุ่มรายการที่เป็นคีย์หลักของ $core(P_l)$ และมีค่าความถี่มากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ $\geq min\_supp$ กำหนดให้เป็น $closure(P_l) = \{i_0, i_1, i_2, \ldots, i_{l-1}\}$ สังเกตุว่า กลุ่มรายการ $i \in closure(P_l) \leftrightarrow supp(i) \geq min\_supp$ และเลเวลของ i น้อยกว่าเลเวลของ P_l*

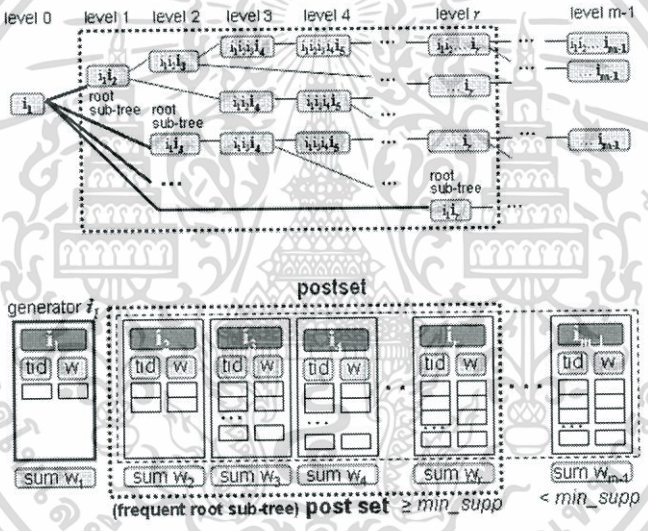
นิยามที่ 4.2 *กำหนดให้ P_l เป็นตัวสร้าง โดยโพสเซตของ P_l คือกลุ่มรายการที่มีความสัมพันธ์กับ P_l แบบรูทซับทรี และอยู่ในพรีฟิคซ์อาร์เรย์เลเวลที่สูงกว่ากลุ่มรายการที่เป็นคีย์หลักของ $core(P_l)$ และมีค่าความถี่ $\geq min\_sup$ กำหนดให้เป็น $post(P_l) = \{i_{l+1}, i_{l+2}, i_{l+3}, \ldots, i_{m-1}\}$*

การประยุกต์ใช้นิยามที่ 4.1 และ 4.2 ในการคัดกรองเอาเฉพาะข้อมูลที่สัมพันธ์กันกับตัวสร้างจริง ๆ มาเก็บไว้ในบักเก็ตอาร์เรย์เบื้องต้น ซึ่งจะส่งผลให้ข้อมูลเริ่มต้นมีปริมาณที่สั้นลงกว่าข้อมูลที่อยู่ในโครงสร้างข้อมูลอีบีพีเอ หากประยุกต์ใช้ข้อกำหนดในนิยามที่ 4.1 กลุ่มรายการที่เป็นโคลสเซอร์เซต คือ กลุ่มรายการที่ถูกคัดกรองเอาเฉพาะข้อมูลที่เกี่ยวข้อง ซึ่งอยู่ในเลเวลต่ำกว่ากลุ่มรายการที่เป็นคีย์หลักของตัวสร้างและมีค่าความถี่ของข้อมูลที่สัมพันธ์กันนั้นผ่านค่าความถี่ขั้นต่ำที่กำหนดเอาไว้

ส่วนนิยามที่ 4.2 ได้กำหนดขอบเขตข้อมูลโพสเซต ซึ่งช่วยให้สามารถคัดกรองเอาเฉพาะข้อมูลที่เกี่ยวข้องที่อยู่ในเลเวลที่สูงกว่ากลุ่มรายการที่เป็นคีย์หลักของตัวสร้างและมีค่าความถี่ของข้อมูลผ่านค่าความถี่ขั้นต่ำ โดยจะคัดกรองเอาเฉพาะโหนดข้อมูลที่เชื่อมโยงกับตัวสร้างขนาด 1 รายการโดยตรงเท่านั้น ซึ่งเรียกเทคนิคนี้ว่าพรีเทส เทคนิคพรีเทสมีโอกาสที่จะเกิดขึ้นในขั้นตอนการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดใน 3 กรณีดังนี้

กรณีที่ 1 ใช้เพียงโพสเซตของตัวสร้าง i_1 (รูทซับทรี: root sub-tree)

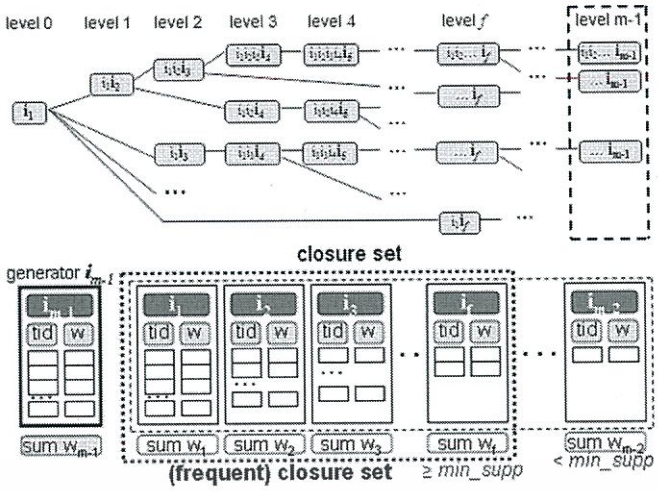
สำหรับตัวสร้าง i_1 นั้น ใช้ข้อมูลโพสเซตที่เป็นกลุ่มของโหนดลูกที่เชื่อมโยงโดยตรงกับกลุ่มรายการขนาด 1 รายการโดยตรงหรือเรียกว่ารูทซับทรี ดังแสดงในรูปที่ 4.1 โหนดข้อมูลที่เชื่อมโยงด้วยเส้นสีเข้มนั้นคือโพสเซตของตัวสร้างเริ่มต้น หากการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดเปรียบเทียบเฉพาะกับกลุ่มโพสเซตเหล่านี้ก็จะใช้เวลาในการเปรียบเทียบน้อยลงกว่าการที่ต้องเปรียบเทียบกับกลุ่มรายการที่อยู่ในเลเวลที่สูงกว่าคีย์หลักของตัวสร้างทั้งหมด



รูปที่ 4.1 ตัวอย่างโพสเซตของตัวสร้าง i_1

กรณีที่ 2 ใช้เพียงโคลสเซอร์เซตของตัวสร้าง i_{m-1}

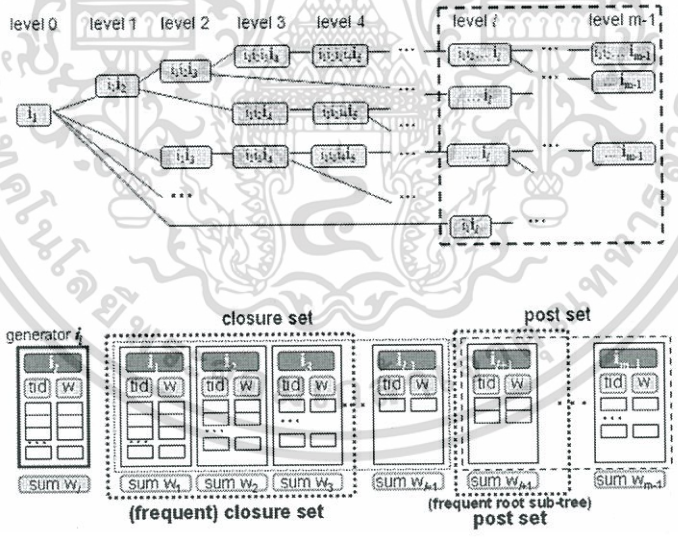
สำหรับตัวสร้าง i_{m-1} นั้น จะใช้ข้อมูลโคลสเซอร์เซตที่เป็นกลุ่มรายการที่จะนำมาใช้ในการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดที่เป็นซัฟฟิคซ์ของตัวสร้างขนาด 1 รายการ โดยจะนำเอาโคลสเซอร์เซตเหล่านี้มาร่วมเข้าเป็นส่วนหนึ่งของกลุ่มรายการที่เกิดบ่อยแบบปิดเพื่อสร้างตัวสร้างตัวใหม่ขึ้นมาสำหรับนำไปค้นหากลุ่มรายการที่เกิดบ่อยแบบปิดตัวถัดไป ดังแสดงในโครงข่ายความสัมพันธ์ในรูปที่ 4.2



รูปที่ 4.2 ตัวอย่างโคลสเซอร์เซตของตัวสร้าง i_{m-1}

กรณีที่ 3 ใช้ทั้งโคลสเซอร์เซตและโพสเซตของตัวสร้าง i_l

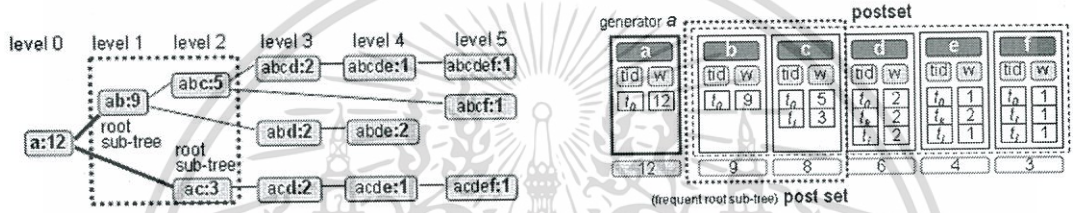
สำหรับตัวสร้าง i_l ใด ๆ (ยกเว้น i_1 และ i_{m-1}) นั้น ใช้ข้อมูลทั้งโคลสเซอร์เซตและโพสเซตร่วมกันเพื่อใช้ในการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดและกลุ่มรายการที่เกิดบ่อยแบบปิดที่เป็นซัฟฟิคซ์ทั้งหมด (จากเลเวล l จนถึงเลเวล m-1) ดังแสดงในโครงข่ายความสัมพันธ์ในรูปที่ 4.3



รูปที่ 4.3 ตัวอย่างโคลสเซอร์เซตและโพสเซตของตัวสร้าง i_l

จากตัวอย่างที่ในรูปที่ 4.4 (ก) แสดงโครงข่ายของพรีฟิคซ์ทรีของข้อมูลตัวอย่าง ส่วนในรูปที่ 4.4 (ข) เป็นข้อมูลเริ่มต้นจากวิธีการในงานวิจัยนี้ ซึ่งประกอบไปด้วยข้อมูลของตัวสร้าง ข้อมูลของโคลสเซอร์เซต และข้อมูลของโพสเซต

สมมุติว่ากำหนดค่าสนับสนุนขั้นต่ำไว้เท่ากับ 1 หากตัวสร้างคือ {*a*:12} ถ้าหากว่าเป็นวิธีการอื่น ๆ นั้นกลุ่มรายการที่เป็นโพสเซตของตัวสร้างนี้ก็คือ {*b*:9}, {*c*:8}, {*d*:6}, {*e*:4} และ {*f*:3} จำนวนทั้งสิ้น 5 รายการ แต่ถ้าหากใช้วิธีการที่นำเสนอในงานวิจัยนี้โพสเซ็ตของตัวสร้าง {*a*:12} จะมีแค่ 2 รายการคือ {*b*:9} และ {*c*:8} ซึ่งเป็นโหนดลูกติดตัวของตัวสร้าง *a* นั้นเอง หากตรวจสอบข้อมูลกับเฉพาะ 2 รายการนี้จะทำให้สามารถลดเวลาในการทำงานลงไปได้ โดยเฉพาะอย่างยิ่งในกรณีที่มีกลุ่มโพสเซ็ตที่ไม่เกี่ยวข้องในปริมาณมาก ๆ หากตรวจสอบจากข้อมูลทั้งหมดเหมือนในกรณีของวิธีการอื่น ๆ ก็จะใช้เวลาในการทำงานนานขึ้น วิธีการในงานวิจัยนี้ในคุณสมบัติพื้นฐานของโครงข่ายพรีฟิคซ์ที่โหนดลูกที่เป็นโหนดซัฟฟิคซ์จะมีค่าความถี่เท่ากับหรือน้อยกว่าโหนดแม่ ดังนั้นหากเราตรวจสอบความซ้ำซ้อนของการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดกับเฉพาะข้อมูลที่เกี่ยวข้องก็จะทำให้ใช้เวลาในการทำงานน้อยลง



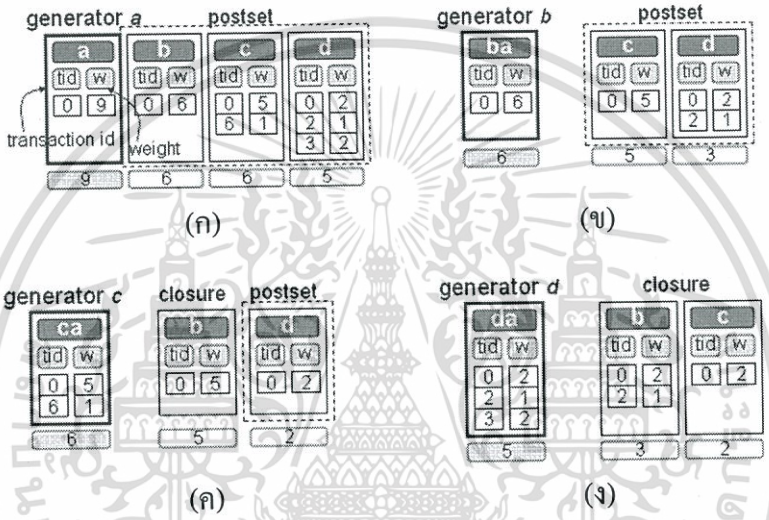
(ก) (ข)

รูปที่ 4.4 (ก) ตัวอย่างโครงข่ายพรีฟิคซ์ทรี และ (ข) การคัดกรองข้อมูลของตัวสร้าง *a*

โดยได้ประยุกต์ใช้เทคนิคพริเทสในการกรองข้อมูลในขั้นตอนที่ 1 นั้น จากโครงสร้างข้อมูลอีบีพีเอเพื่อคำนวณโคลสเซอร์เซตและโพสเซตของตัวสร้างได้อย่างมีประสิทธิภาพซึ่งคล้าย ๆ กับโครงสร้างข้อมูลแบบรวมที่ใช้ออคเคอร์เรนซ์ ดิลิเวอร์ในการคัดกรองเอาเฉพาะข้อมูลที่เกี่ยวข้อง โดยข้อมูลที่ถูกคัดกรองจะถูกสร้างจากข้อมูลที่เกี่ยวข้องกับตัวสร้างขนาด 1 รายการและนำไปใช้เพื่อหากลุ่มรายการที่เกิดบ่อยแบบปิดที่เป็นซัฟฟิคซ์ของตัวสร้างขนาด 1 รายการทั้งหมด ด้วยเหตุนี้เอง การคำนวณความถี่จากข้อมูลที่คัดกรองเอาไว้แล้วจึงทำงานได้รวดเร็วกว่าการคำนวณความถี่จากโครงสร้างข้อมูลที่มีอยู่ เช่น โครงสร้างข้อมูลบิทแมบ อาร์เรย์ลิตส์ และพรีฟิคซ์อื่น ๆ โดยการคัดกรองข้อมูลด้วยวิธีการในงานวิจัยนี้สามารถสร้างตามขั้นตอนดังนี้

อันดับแรก เริ่มต้นโดยจัดการกับข้อมูลของตัวสร้างเริ่มต้นขนาด 1 รายการ โดยบักเก็ตอาร์เรย์เบื้องต้นของตัวสร้างจะจัดเก็บเฉพาะค่าดัชนีที่ใช้ในการอ้างถึงข้อมูลในพรีฟิคซ์อาร์เรย์ จากรูปที่ 4.5 เพื่อให้ง่ายต่อการทำความเข้าใจจึงแสดงตัวอย่างบัคเก็ตอาร์เรย์ที่แสดงข้อมูลจากพรีฟิคซ์อาร์เรย์ที่เชื่อมโยงเอาไว้ แต่ในการจัดเก็บข้อมูลจริง ๆ จะจัดเก็บเพียงค่าดัชนีของแต่ละโหนดที่เชื่อมโยงกันเท่านั้นเพียงอย่างเดียวเท่านั้นเพื่อความรวดเร็วในการในการสลับตำแหน่งข้อมูล บักเก็ตอาร์เรย์เบื้องต้นของตัวสร้าง 4 รายการ คือกลุ่มรายการ *a* ในรูปที่ 4.5 (ก) กลุ่มรายการ *b* ในรูปที่ 4.5 (ข) กลุ่มรายการ *c* ในรูปที่ 4.5 (ค) และกลุ่มรายการ *d* ในรูปที่ 4.5 (ง) ตามลำดับ

อันดับถัดมาเป็นการสแกนข้อมูลในเลเวลที่ต่ำกว่าเลเวลของคีร์หลักในตัวสร้างเพื่อหาข้อมูลโคลสเซอร์เซตด้วยการสแกนบิทข้อมูลโดยอ้างอิงจากโหนดของตัวสร้างเป็นหลัก หากค่าความถี่ของกลุ่มรายการโคลสเซอร์ใดมีผ่านค่าสนับสนุนขั้นต่ำก็เพิ่มเข้าในรายการโคชเซอร์เซตของตัวสร้างและอันดับสุดท้ายเป็นการคัดกรองข้อมูลโพสเซตที่เกี่ยวข้องกับตัวสร้างโดยสแกนเฉพาะข้อมูลที่เป็นของรูทซับทรีของตัวสร้างที่มีค่าความถี่ผ่านค่าสนับสนุนขั้นต่ำ เวลาในการทำงานทั้งหมดของสำหรับตัวสร้างแต่ละตัวเท่ากับ $O(mn)$ ($\leq n$ โหนดในแต่ละเลเวล โดย m คือจำนวนของโคลสเซอร์เซตและโพสเซต) และใช้เวลาในการประมวลผลสำหรับ m ตัวสร้างทั้งหมดเท่ากับ $O(m^2n)$



รูปที่ 4.5 ตัวอย่างข้อมูลเริ่มต้นของตัวสร้าง *a, b, c* และ *d*

จากตัวอย่างในรูปที่ 4.5 (ค) นั้นแสดงข้อมูลเริ่มต้นของตัวสร้าง *c* ที่อยู่ในเลเวลที่ 2 ของพรีฟิคซ์อาร์เรย์ ตามปกติแล้วข้อมูลเริ่มต้นจะเก็บแค่ลิงค์เชื่อมโยงไปยังตำแหน่งสมาชิกในพรีฟิคซ์อาร์เรย์เพื่อความรวดเร็วในการสลับตำแหน่งสมาชิกในการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดที่เป็นซัฟฟิคซ์ของตัวสร้างต่อไป แต่เพื่อให้เกิดความเข้าใจได้ง่ายขึ้นจึงนำเอาข้อมูลของพรีฟิคซ์อาร์เรย์มาแสดงในข้อมูลเบื้องต้น โดยตัวสร้างนี้มีสมาชิกจำนวน 2 โหนด คือโหนดสมาชิก 0 และ 1 โดยโหนด 0 ของพรีฟิคซ์อาร์เรย์เชื่อมโยงไปยังบิทแมบทรานแซคชันแถวที่ 0 มีค่าความถี่เท่ากับ 5 ส่วนโหนด 1 ของพรีฟิคซ์อาร์เรย์เชื่อมโยงไปยังบิทแมบทรานแซคชันแถวที่ 6 มีค่าความถี่เท่ากับ 1 ดังนั้นค่าความถี่ของตัวสร้าง *c* จึงเท่ากับ 6 นั้นเอง โดยตัวสร้าง *c* มีโคลสเซอร์ 2 รายการ คือ *a* และ *b* ที่เป็นกลุ่มรายการที่อยู่ในเลเวลที่ต่ำกว่า *c* แต่เนื่องจาก *a* มีค่าความถี่เท่ากับ *c* จึงถูกรวมเข้าเป็นส่วนหนึ่งของตัวสร้าง *ca* โดยมี *a* เป็นคีร์หลัก (*core*) ของตัวสร้าง ดังนั้นจึงเหลือโคลสเซอร์เซตเพียง 1 รายการ และมีโพสเซต 1 รายการคือ *d* โดยกลุ่มรายการ *b* อยู่ในทรานแซคชัน *tid*=0

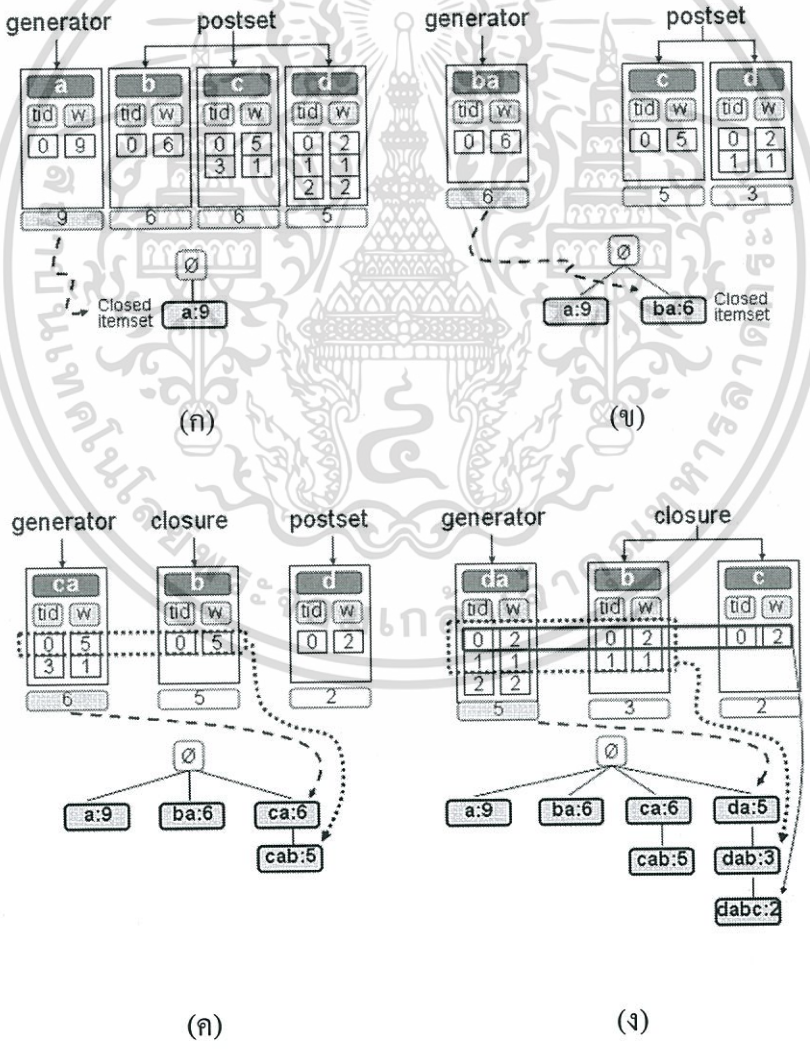
เดียวกันกับตัวสร้าง *ca* โดยสแกนจากข้อมูลของตัวสร้าง *c* ดังนั้น *b* จึงมีค่าความถี่เท่ากับ 5 ส่วนกลุ่มรายการ *d* อยู่ในเลเวล 3 ซึ่งเป็นเลเวลที่สูงกว่าตัวสร้าง *c* จึงเป็นกลุ่มรายการที่เป็นโพสเซต เมื่อตรวจสอบในเลเวล 3 ทั้งหมดที่มีสมาชิก 3 โหนดนั้นพบว่า มีโหนด 0 เท่านั้นที่มีความสัมพันธ์กับตัวสร้าง *c* ดังนั้นโพสเซต *d* จึงมีข้อมูล 1 รายการเชื่อมโยงไปยังบิทแมบทรานแซคชัน 0 มีค่าความถี่เท่ากับ 2 (การตรวจสอบข้อมูลโคชเชอร์เซตจะตรวจสอบจากข้อมูลของตัวสร้าง แต่ถ้าเป็นข้อมูลโพสเซตจะตรวจสอบจากเลเวลของกลุ่มรายการโพสเซตเอง) ส่วนตัวสร้างอื่น ๆ ในรูปที่ 4.5 (ก) – (ง) ก็จะใช้วิธีการเดียวกันกับตัวสร้าง *c* ที่ยกตัวอย่างในข้างต้นนี้

ในการคำนวณความถี่ของกลุ่มรายการนั้น พื้นที่ในการค้นหาข้อมูลถือว่าเป็นจุดสำคัญที่ส่งผลต่อประสิทธิภาพของการทำงานเป็นอย่างมาก เพราะว่าการคำนวณความถี่ใช้เวลาในการทำงานเป็นส่วนใหญ่ของการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด อย่างเช่น โครงสร้างข้อมูลบิทแมบใช้การคำนวณความถี่จากทรานแซคชันทั้งหมด และโครงสร้างข้อมูลอาร์เรย์ลิตส์ใช้การคำนวณความถี่จากบักเก็ตอาร์เรย์ ซึ่งต้องใช้เวลาในการทำงาน โดยเฉพาะอย่างยิ่งในกรณีที่ฐานข้อมูลมีขนาดใหญ่ ด้วยเหตุนี้เองหากใช้การคัดกรองเฉพาะข้อมูลที่เกี่ยวข้องในรอบแรกแล้วนำไปใช้เพื่อหากลุ่มรายการที่เกิดบ่อยแบบปิดที่เป็นซัฟฟิคซ์ทั้งหมด ก็จะใช้เวลาในการทำงานน้อยลง ซึ่งวิธีการ EBPA-CLOSED ใช้หลัก การทำงานเช่นเดียวกันกับวิธีการ LCM-3 ที่ใช้การทำงานแบบพีพีซีเอ็กเทนชั่นร่วมกับบักเก็ตอาร์เรย์โดยการคัดกรองข้อมูลที่เกี่ยวข้อง โดยได้ปรับปรุงการคัดกรองข้อมูลให้เหลือเฉพาะข้อมูลที่จำเป็นจะต้องใช้ในการตรวจสอบจริง ๆ เท่านั้น การเตรียมข้อมูลเริ่มต้นร่วมกับเทคนิคพรีเทสในงานวิจัยนี้ใช้เวลาในการทำงานดีที่สุดเท่ากับ $O(mn)$ และใช้เวลาในการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดสำหรับกลุ่มรายการ *m* ทั้งหมดเท่ากับ $O(m^2n)$

นอกจากใช้เทคนิคพรีเทสเพื่อปรับปรุงประสิทธิภาพของการคัดกรองข้อมูลเริ่มต้นแล้วยังได้ออกแบบให้พรีฟิคซ์อาร์เรย์ใช้บิทแมบทรานแซคชันร่วมกันทำให้ประหยัดพื้นที่หน่วยความจำ โดยโครงสร้างข้อมูลแบบรวมจะจัดเก็บบิทแมบทรานแซคชันไว้ในแต่ละโหนดของข้อมูล ซึ่งจะใช้พื้นที่ในการเพิ่มขึ้น เช่น หาก $m = 4$ คือ *a, b, c* และ *d* โหนด *a* จะจัดเก็บบิทแมบ 1000 โหนด *b* (1100) โหนด *c* (1110) และโหนด *d* (1111) ส่วนวิธีการในงานวิจัยนี้จะจัดเก็บแค่ลิงค์เชื่อมโยงไปยังบิทแมบทรานแซคชันแทนจึงทำให้ประหยัดพื้นที่หน่วยความจำเพิ่มขึ้น

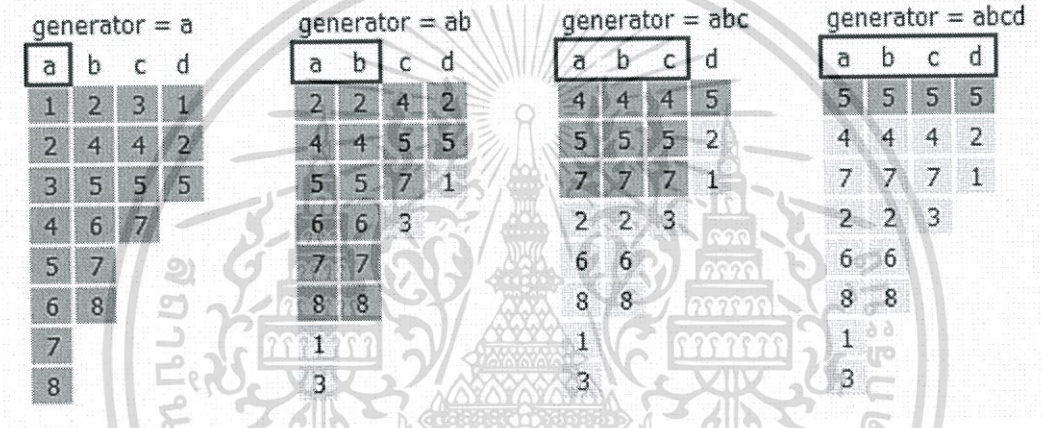
สำหรับขั้นตอนที่ 2 คือ การสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด จะทำหลังจากการจัดเตรียมข้อมูลเริ่มต้นให้กับตัวสร้างขนาด 1 รายการเรียบร้อยแล้ว หากตัวสร้างผ่านค่าสนับสนุนขั้นต่ำที่กำหนดไว้ในการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดก็จะเข้าสู่ขั้นตอนการเปรียบเทียบกับกลุ่มโพสเซตว่าตัวสร้างนี้เป็นส่วนหนึ่งของโพสเซตหรือไม่ หากมีค่าความถี่เท่ากันกับโพสเซตตัวใดตัวหนึ่งจะถือว่าตัวสร้างนี้ไม่เป็นกลุ่มรายการที่เกิดบ่อยแบบปิด หากตรวจสอบกับกลุ่มโพสเซตทั้งหมดแล้วมีค่าความถี่ไม่เท่ากัน แสดงว่าตัวสร้างนี้ เป็นกลุ่มรายการที่เกิดบ่อยแบบปิดนั้นเอง

จากนั้นให้นำกลุ่มรายการที่เกิดบ่อยแบบปิดที่ได้ไปรวมกับโคลสเซอร์เซตตัวแรก แล้วตั้งให้กลุ่มรายการที่รวมเข้ามาใหม่เป็นคีร์หลักของตัวสร้างเพื่อหากลุ่มรายการที่เกิดบ่อยแบบปิดตัวต่อไป กลุ่มรายการที่อยู่ในเลเวลที่น้อยกว่าคีร์หลักจะเป็นโคลสเซอร์เซตและกลุ่มรายการที่อยู่ในเลเวลที่มากกว่าคีร์หลักและไม่ได้เป็นส่วนหนึ่งของเซตนี้จะเป็นโพสเซตของตัวสร้างใหม่ พร้อมทั้งตรวจสอบว่าตัวสร้างใหม่ผ่านค่าสนับสนุนขั้นต่ำหรือไม่ หากผ่านก็ให้ทำขั้นตอนถัดไป ซึ่งจะเป็นขั้นตอนของการตรวจสอบว่าโคลสเซอร์เซตตัวใดเป็นส่วนร่วมกับตัวสร้างใหม่หรือไม่ (มีค่าความถี่เท่ากัน) หากมีก็เพิ่มโคลสเซอร์เซตตัวดังกล่าวเข้าเป็นส่วนหนึ่งในเซตของตัวสร้าง จากนั้นจึงตรวจสอบกับโพสเซตเพื่อสร้างกลุ่มรายการที่เกิดบ่อยแบบปิด โดยจะทำขั้นตอนดังกล่าวไปเรื่อย ๆ จนกว่าจะไม่มีข้อมูลโคลสเซอร์เซ็ต จากนั้นจึงจะวนไปเริ่มทำงานในขั้นตอนที่ 1 เพื่อเตรียมข้อมูลเริ่มต้นของตัวสร้างขนาด 1 รายการตัวต่อไปวนทำงานไปเรื่อย ๆ จนกว่าข้อมูลขนาด 1 รายการจะหมดดังแสดงในรูปที่ 4.6 (ก) – (ง)



รูปที่ 4.6 ตัวอย่างการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดจากตัวสร้าง *a, b, c* และ *d*

ในระหว่างที่ทำเปรียบเทียบค่าเพื่อสร้างตัวสร้างใหม่และการเปรียบเทียบค่ากับข้อมูลโพสเซตจะมีการสลับตำแหน่งอาร์เรย์ในข้อมูลเริ่มต้นเพื่อความรวดเร็ว ดังแสดงในรูปที่ 4.7 ซึ่งจะส่งผลดีต่อการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดที่เป็นซัฟฟิคซ์ นั้นเป็นเพราะว่าขอบเขตของการค้นหาจะน้อยลงเรื่อย ๆ เช่น ในการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดตัวแรกคือ *a* ใช้รอบการค้นหา 8 รอบ เมื่อเราสร้างตัวสร้างใหม่ขึ้นมาคือ *ab* เราทำการสลับตำแหน่งเอาเฉพาะข้อมูลที่ตรงกันมาไว้ข้างบนของอาร์เรย์ ซึ่งจะเหลือรอบในการตรวจสอบเพียง 6 รอบ ต่อมาตัวสร้าง *abc* เหลือรอบการทำงานเพียง 3 รอบและสุดท้าย *abcd* เหลือรอบการทำงานเพียง 1 รอบเท่านั้น หากเป็นการทำงานโดยทั่ว ๆ ไปตัวสร้างที่เป็นซัฟฟิคซ์ทั้งหมดจะมีรอบการทำงานเท่ากันคือ 8 รอบ ด้วยเหตุนี้เองจึงทำให้วิธีการในงานวิจัยนี้ สามารถลดเวลาในการทำงานลงไปได้อีกจุดหนึ่ง



รูปที่ 4.7 ตัวอย่างการสลับตำแหน่งข้อมูลจากตัวสร้าง *a, b, c* และ *d*

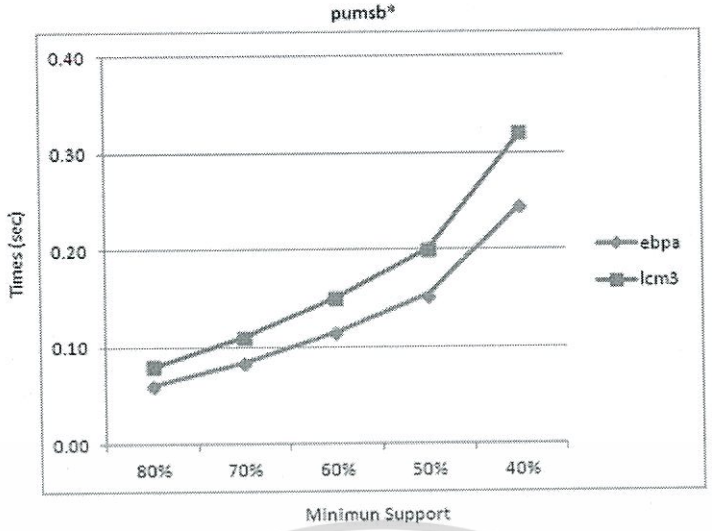
วิธีการในงานวิจัยนี้ได้ปรับปรุงประสิทธิภาพของการทำงานให้ดีขึ้น โดยจะแตกต่างจากอัลกอลิทึม LCM-3 ซึ่งได้นำเอาเทคนิคพรีเทสมาคัดกรองเอาเฉพาะข้อมูลที่จำเป็นจะต้องใช้ในการตรวจสอบข้อมูลเท่านั้นจึงทำให้สามารถลดปริมาณของข้อมูลที่จะต้องตรวจสอบลง ซึ่งส่งผลให้ใช้เวลาในการทำงานได้รวดเร็วขึ้น

บทที่ 5
การประเมินและวิเคราะห์ประสิทธิภาพ

ในส่วนนี้นำเสนอเกี่ยวกับการประเมินผลและการวิเคราะห์ประสิทธิภาพการทำงานของวิธีการ EBPA-CLOSED โดยใช้โครงสร้างข้อมูลอีบีพีเอ (EBPA) ที่ได้นำเสนอรายละเอียดไว้ในบทที่ 3 และบทที่ 4 โดยได้ทำการประมวลผลเปรียบเทียบกับวิธีการ LCM-3 ซึ่งเป็นวิธีการต้นแบบของการพัฒนาในครั้งนี้ด้วยโดยใช้ฐานข้อมูลตัวอย่างและค่าสนับสนุนที่แตกต่างกัน

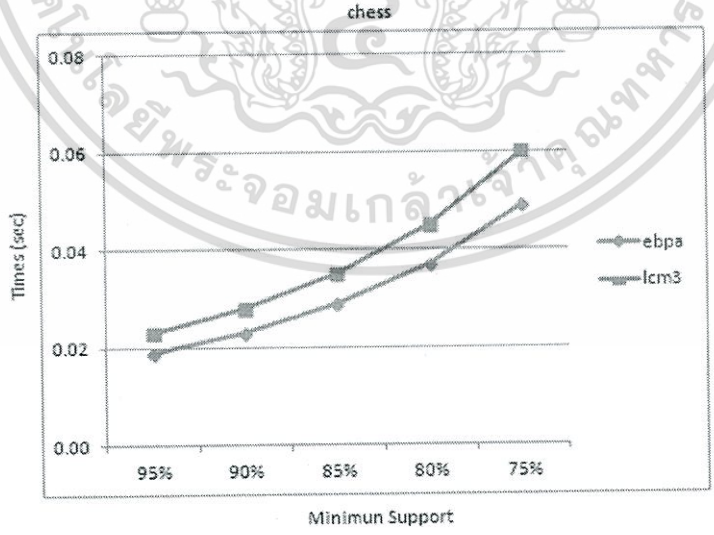
เพื่อประเมินประสิทธิภาพในการทำงาน งานวิจัยนี้ได้พัฒนาวิธีการ EBPA-CLOSED กับโครงสร้างข้อมูลอีบีพีเอทำการประมวลผลเปรียบเทียบกับวิธีการ LCM-3 ที่เป็นต้นแบบในการพัฒนา โดยใช้ภาษาซีในการเขียนโปรแกรม ทำการทดลองบนเครื่อง Notebook PC หน่วยประมวลผล Intel 2.5 GHz Core i5 บนระบบปฏิบัติการ Windows XP ใช้หน่วยความจำ Ram 2048 MB โดยทดสอบกับฐานข้อมูลแบบหนาแน่น (Dense) ที่ได้รับการยอมรับและใช้ในการทดสอบในหลาย ๆ งานวิจัยจำนวน 3 ฐาน ข้อมูลคือ Pumsb*, Chess และ Connect โดยฐานข้อมูลแรกคือ Pumsb* มี 34,776 ทรานแซคชัน และมีกลุ่มรายการขนาด 1 รายการจำนวน 2,001 รายการ, ฐานข้อมูลที่ 2 คือ Chess มี 3,196 ทรานแซคชัน มีกลุ่มรายการขนาด 1 รายการจำนวน 75 รายการและฐานข้อมูลสุดท้าย Connect มี 67,437 ทรานแซคชัน มีกลุ่มรายการขนาด 1 รายการจำนวน 129 รายการ โดยได้จัดเก็บผลการทดลองในส่วนของการใช้เวลาในการทำงาน (วินาที) เพื่อใช้เวลาในการเปรียบเทียบประสิทธิภาพของการทำงานระหว่างวิธีการ EBPA-CLOSED ที่ใช้โครงสร้างข้อมูลอีบีพีเอและวิธีการ LCM-3 ที่ใช้โครงสร้างข้อมูลแบบรวม โดยการทดลองนั้นทำการทดสอบด้วยการกำหนดค่าสนับสนุนขั้นต่ำเป็นเปอร์เซนต์ที่แตกต่างกันและจัดเก็บเวลาที่เกิดจากการกำหนดค่าสนับสนุนที่แตกต่างกันในฐานข้อมูลที่ใช้ในการทดสอบทั้ง 3 ฐานข้อมูล สำหรับการพัฒนาโปรแกรมในงานวิจัยนี้ได้นำเอาซอร์ซโค้ดต้นฉบับของโปรแกรม LCM-3 มาแก้ไขเพิ่มเติมในส่วนที่ได้นำเสนอในงานวิจัยนี้ทั้งในส่วนของการสร้างโครงสร้างข้อมูลและในส่วนของการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด จากนั้นจึงทำการประมวลผลเพื่อเปรียบเทียบเวลาในการทำงานของ EBPA-CLOSED และ LCM-3 บนสภาวะแวดล้อมเดียวกัน

จากผลการทดลองที่แสดงในรูปที่ 5.1 นั้นเป็นผลการทำงานโดยเปรียบเทียบการทำงานระหว่างวิธีการ EBPA-CLOSED และวิธีการ LCM-3 ด้วยฐานข้อมูล Pumsb* ที่มีจำนวน 34,776 ทรานแซคชัน และมีกลุ่มรายการขนาด 1 รายการจำนวน 2,001 รายการ โดยจัดเก็บเวลาในการทำงานจากการกำหนดค่าสนับสนุนขั้นต่ำที่เป็นเปอร์เซนต์แตกต่างกัน โดยเริ่มจาก 80% 70% 60% 50% และ 40% ตามลำดับ จากผลการทดลองพบว่าวิธีการ EBPA-CLOSED ใช้เวลาในการทำงานที่น้อยกว่าวิธีการ LCM-3 ประมาณ 30%

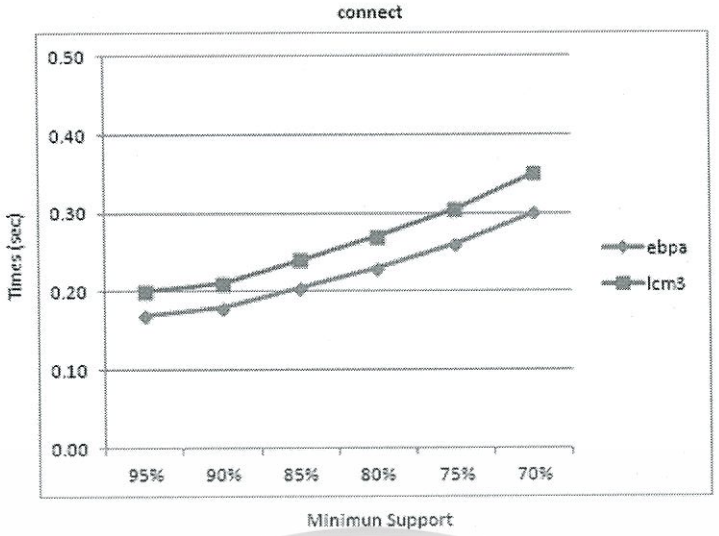


รูปที่ 5.1 เปรียบเทียบเวลาประมวลผลระหว่าง EBPA-CLOSED กับ LCM-3 ฐานข้อมูล Pumsb*

รูปที่ 5.2 เป็นผลการทดลองเพื่อเปรียบเทียบการทำงานระหว่างวิธีการ EBPA-CLOSED และวิธีการ LCM-3 ด้วยฐานข้อมูล Chess ที่มีจำนวน 3,196 ทรานแซคชัน และมีกลุ่มรายการขนาด 1 รายการจำนวน 75 รายการ โดยจัดเก็บเวลาในการทำงานจากการกำหนดค่าสนับสนุนขั้นต่ำเป็นเปอร์เซนต์ที่แตกต่างกัน โดยเริ่มจาก 95% 90% 85% 80% และ 75% สำหรับฐานข้อมูลนี้ วิธีการ EBPA-CLOSED ใช้เวลาในการทำงานที่น้อยกว่าวิธีการ LCM-3 ประมาณ 22%



รูปที่ 5.2 เปรียบเทียบเวลาประมวลผลระหว่าง EBPA-CLOSED กับ LCM-3 ด้วยฐานข้อมูล Chess



รูปที่ 5.3 เปรียบเทียบเวลาประมวลผลระหว่าง EBPA-CLOSED กับ LCM-3 ฐานข้อมูล Connect

ผลการทดสอบสุดท้ายในรูปที่ 5.3 เป็นผลการทดลองเพื่อเปรียบเทียบการทำงานระหว่างวิธีการ EBPA-CLOSED และวิธีการ LCM-3 ด้วยฐานข้อมูล Connect ที่มีจำนวน 67,437 ทรานแซคชัน และมีกลุ่มรายการขนาด 1 รายการจำนวน 129 รายการ โดยจัดเก็บเวลาในการทำงานจากการกำหนดค่าสนับสนุนขั้นต่ำเป็นเปอร์เซนต์ที่แตกต่างกัน โดยเริ่มจาก 95% 90% 85% 80% 75% และ 70% สำหรับฐานข้อมูลนี้วิธีการ EBPA-CLOSED ใช้เวลาในการทำงานที่น้อยกว่าวิธีการ LCM-3 ประมาณ 15%

จากผลการทดลองทั้งหมด พบว่า วิธีการที่นำเสนอซึ่งใช้รูปแบบการทำงาน LCM-3 มาเป็นต้นแบบในการพัฒนา จึงได้ผลการทำงานไปในทิศทางเดียวกันกับวิธีการต้นแบบ แต่วิธีการในงานวิจัยนี้ใช้เวลาในการทำงานที่น้อยกว่า จากผลการทำงานกับทั้ง 3 ฐานข้อมูลพบว่าในฐานข้อมูล Pumsb* วิธีการนี้ทำงานเร็วกว่า 30%, Chess ทำงานเร็วกว่า 22% และ Connect ทำงานเร็วกว่า 15% หากพิจาณาถึงประสิทธิภาพที่เพิ่มขึ้นนั้นเกิดมาจาก 2 ส่วนที่ได้ปรับปรุงประสิทธิภาพนั้นก็คือ การเติมค่าความถี่จากทรานแซกชั่นข้อมูลลงในโครงสร้างข้อมูลด้วยการใช้เทคนิคการกำหนดค่าดัชนีของสมาชิกแต่ละโหนดให้กับพรีฟิคซ์อาร์เรย์ที่เป็นส่วนหนึ่งของโครงสร้างข้อมูลอีบีพีเอ และส่วนที่ 2 คือ การปรับปรุงวิธีการคัดกรองข้อมูลเพื่อลดปริมาณของข้อมูลที่ใช้สำหรับตรวจสอบว่ากลุ่มรายการที่เป็นตัวสร้างสามารถที่จะเป็นกลุ่มรายการแบบปิดได้หรือไม่ด้วยการใช้เทคนิคที่เรียกว่าพรีเทส จากผลการทดลองจึงสรุปได้ว่า โครงสร้างข้อมูลอีบีพีเอและวิธีการ EBPA-CLOSED ที่ได้นำเสนอในงานวิจัยนี้สามารถช่วยให้การค้นหากลุ่มรายการที่เกิดบ่อยแบบปิดทำงานได้อย่างมีประสิทธิภาพมากยิ่งขึ้น

บทที่ 6
บทสรุปและข้อเสนอแนะ

6.1 บทสรุป (Conclusion)

ในปัจจุบันนี้เป็นยุคสาระสนเทศ หน่วยงานต่าง ๆ นิยมนำเอาข้อมูลมาใช้ประมวลผลเพื่อให้ได้สาระสนเทศที่เป็นประโยชน์ ด้วยเหตุนี้เองจึงได้มีการจัดเก็บข้อมูลเอาไว้เป็นจำนวนมาก ซึ่งส่งผลต่อวิธีการคัดกรองข้อมูลที่มีอยู่ในปัจจุบันไม่สามารถทำงานได้อย่างเต็มประสิทธิภาพ โดยวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยก็เป็นอีกหนึ่งในวิธีการที่ได้รับผลกระทบ ดังนั้นจึงมีนักวิจัยหลาย ๆ ท่านได้พัฒนาวิธีการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดขึ้นมาเพื่อแก้ปัญหา สำหรับประสิทธิภาพของการสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิดนั้น นอกจากจะออกแบบวิธีการให้มีประสิทธิภาพแล้วโครงสร้างข้อมูลก็เป็นอีกหนึ่งปัจจัยที่มีผลกระทบต่อการทำงาน ซึ่งแต่ละวิธีการได้มีการนำเสนอโครงสร้างข้อมูลโดยมีจุดดีและจุดด้อยแตกต่างกันไป ด้วยเหตุนี้เอง งานวิจัยฉบับนี้จึงได้นำเสนอโครงสร้างข้อมูลอีบีพีเอสำหรับสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด โดยคำนึงถึงประสิทธิภาพในการทำงานให้รองรับข้อมูลขนาดใหญ่ได้ ซึ่งได้จัดเก็บข้อมูลในรูปแบบของพรีฟิคซ์อาร์เรย์ตามเลเวล โดยออกแบบการเติมค่าความถี่จาก ทรานแซกชันข้อมูลลงในโครงสร้างข้อมูลด้วยการใช้เทคนิคการกำหนดค่าดัชนีของสมาชิกแต่ละโหนดให้กับพรีฟิคซ์อาร์เรย์ที่เป็นส่วนหนึ่งของโครงสร้างข้อมูลอีบีพีเอ โดยค่าดัชนีนี้ได้มาจากการคำนวณตำแหน่งโหนดต่าง ๆ ในแต่ละเลเวลของโครงสร้างข้อมูลพรีฟิคซ์ทรีแบบสมบูรณ์ โดยไม่จำเป็นต้องสร้างโครงสร้างข้อมูลพรีฟิคซ์ทรีแบบสมบูรณ์ พร้อมทั้งออกแบบการเชื่อมโยงระหว่างโหนดแม่และโหนดลูกเพื่อความรวดเร็วในการทำงานทำให้การเพิ่มความถี่สามารถทำได้โดยไม่จำเป็นต้องจัดเรียงและรวมแถวข้อมูลและไม่ต้องเก็บลิงค์เชื่อมโยงระหว่างโหนดแม่กับโหนดลูกทำให้ประหยัดพื้นที่ในหน่วยความจำเพิ่มขึ้น และลดการจัดเก็บรายละเอียดข้อมูลในแต่ละโหนดเพื่อช่วยให้ประหยัดพื้นที่หน่วยความจำหลักโดยใช้บิทร่วมกันในกลุ่มรายการที่มีเส้นทางเดียวกัน นอกจากนี้ยังได้ปรับปรุงขั้นตอนการค้นหากลุ่มรายการที่เกิดบ่อยแบบปิดด้วยการปรับปรุงวิธีการคัดกรองข้อมูลเพื่อลดปริมาณของข้อมูลที่ใช้สำหรับตรวจสอบว่ากลุ่มรายการที่เป็นตัวสร้างสามารถที่จะเป็นกลุ่มรายการที่เกิดบ่อยแบบปิดได้หรือไม่ด้วยการใช้เทคนิคที่เรียกว่า พรีเทสคัดกรองเอาเฉพาะข้อมูลที่จำเป็นจะต้องใช้ในการตรวจสอบข้อมูลเท่านั้นจึงทำให้สามารถลดปริมาณของข้อมูลที่จะต้องตรวจสอบลง ซึ่งส่งผลให้ใช้เวลาในการทำงานได้รวดเร็วขึ้น โดยใช้เวลาในการประมวลผลเท่ากับ $O(mT) + O(mn)$ ซึ่งเร็วกว่าวิธีการต้นแบบคือวิธีการ LCM-3 ที่ใช้โครงสร้างข้อมูลแบบรวม จากผลการทดลองพบว่าวิธีการในงานวิจัยชิ้นนี้ทำงานได้เร็วกว่าประมาณ 15-30%

6.2 ข้อเสนอแนะ (Recommendation)

การวิจัยนี้เป็นการพัฒนาการทำงานเพื่อสืบค้นกลุ่มรายการที่เกิดบ่อยแบบปิด ซึ่งเป็นการค้นหารูปแบบของข้อมูลในเบื้องต้น หากต้องการจะนำข้อมูลที่ได้ไปใช้งานจริงจะต้องมีโปรแกรมที่เกี่ยวข้องเพิ่มเติมมาเรียกใช้งาน เช่น หากต้องการนำไปวิเคราะห์หากฎความสัมพันธ์ของข้อมูลจะต้องใช้โปรแกรมที่ใช้ในการสร้างกฎความสัมพันธ์มาเรียกใช้งานอีกที

นอกจากนี้โครงสร้างข้อมูลอีบีพีเอที่นำเสนอในงานวิจัยนี้ออกแบบในส่วนของการเข้าถึงข้อมูลให้มีประสิทธิภาพแต่ยังรองรับการทำงานของข้อมูลในขอบเขตที่จำกัดตามข้อจำกัดของหน่วยความจำที่ยอมให้สร้างอาร์เรย์ในขนาดที่จำกัด (ทำงานกับพรีฟิคซ์อาร์เรย์เพียงหนึ่งอาร์เรย์เท่านั้น) หากต้องการนำเอาไปใช้งานจริงจะต้องออกแบบในส่วนของพรีฟิคซ์อาร์เรย์แบบชั่วคราวให้กระจายเป็นก้อนข้อมูลขนาดเล็กที่สามารถสร้างได้ในหน่วยความจำหลาย ๆ ก้อน หากกลุ่มรายการที่เกิดบ่อยมีจำนวนมากจะต้องแบ่งพรีฟิคซ์อาร์เรย์ออกเป็นหลาย ๆ อาร์เรย์และจะต้องกำหนดส่วนในการเชื่อมโยงระหว่างแต่ละอาร์เรย์เพิ่มเติม จึงจะสามารถรองรับการทำงานกับกลุ่มรายการที่เกิดบ่อยจำนวนมากขึ้นได้

สำหรับงานวิจัยในอนาคตนั้นได้ให้ความสนใจกับการพัฒนาวิธีการเรียนรู้รูปแบบข้อมูลแบบขนานเพื่อใช้ในการวิเคราะห์รูปแบบข้อมูล และทำนายความเป็นไปได้ของข้อมูลที่จะเกิดขึ้นในอนาคตจากข้อมูลออนไลน์

เอกสารอ้างอิง

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proceedings of 20th Int'l Conference on Very Large Data Bases, pp. 487–499, Sept. 1994.

[2] J.S. Park, M.-S. Chen, and P.S. Yu, "An Effective Hash Based Algorithm for Mining Association Rules," Proceedings of ACM Int'l Conference on Management of Data, May 1995.

[3] H. Mannila, H. Toivonen, "Multiple Uses of Frequent Sets and Condensed Representations," In Proceedings of KDD'96, pp. 189–194, 1996.

[4] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," Proceedings of ACM SIGMOD Int'l Conference on Management of Data, May 1997.

[5] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proceedings of 2000 ACM SIGMOD Int'l Conference on Management of Data, May 2000.

[6] R. Taouil, N. Pasquier, Y. Bastide, L. Lajhal, and G. Stumme, "Mining Frequent Patterns with Counting Inference," SIGKDD Explorations, vol. 2, no. 2, Dec. 2000.

[7] M. Runying, "Adaptive-FP: An Efficient and Effective Method for Multi-Level Multi-Dimensional Frequent Pattern Mining", Apr. 2001.

[8] J. Pei, J. Han, H. Lu, S. Nishio, D. Tang, and S. Yang, "H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases," Proceedings of IEEE Int'l Conference on Data Mining, Nov. 2001.

[9] J. Liu, Y. Pan, K. Wang, and J. Han, "Mining Frequent Item Sets by Opportunistic Projection," Proceedings of Ninth ACM Int'l Conference on Knowledge Discovery and Data Mining, 2002.

[10] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, "Adaptive and Resource-Aware Mining of Frequent Sets," Proceedings of IEEE Int'l Conference on Data Mining, Dec. 2002.

[11] M.J. Zaki and K. Gouda, "Fast Vertical Mining Using Diffsets," Proceedings of Ninth ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining, Aug. 2003.

[12] M.J. Zaki, "Mining Non-Redundant Association Rules," Proceedings of Data Mining and Knowledge Discovery, vol. 9, no. 3, pp. 223-248, 2004.

[13] B. Goethals and M.J. Zaki, "Advances in Frequent Itemset Mining Implementations: Report on Fimi '03," SIGKDD Explorations, vol. 6, no. 1, pp. 109-117, 2004.

[14] N. Pasquier, Y. Bastide, R. Touil, and L. Lakhal, "Discovering frequent closed itemsets for association rules", Proceedings of 7th International Conference on Database Theory (ICDT 1999), LNCS, v.1540, Springer-Verlag, Jerusalem, Israel, pp. 398–416, Jan. 1999.

[15] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Efficient Mining of Association Rules Using Closed Itemset Lattices," Information Systems, vol. 24, no. 1, pp. 25-46, 1999.

[16] J. Pei, J. Han, and R. Mao, "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets," Proceedings of ACM SIGMOD Int'l Workshop Data Mining and Knowledge Discovery, May 2000.

[17] M.J. Zaki and C.-J. Hsiao, "CHARM: An Efficient Algorithm for Closed Itemsets Mining," Proceedings of Second SIAM Int'l Conference on Data Mining, Apr. 2002.

[18] G. Grahne and J. Zhu, "Efficiently Using Prefix-Trees in Mining Frequent Itemsets," Proceedings of ICDM Workshop Frequent Itemset Mining Implementations, Dec. 2003.

[19] J. Pei, J. Han, and J. Wang, "C LOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets," Proceedings of Ninth ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining, Aug. 2003.

[20] T. Uno, T. Asai, Y. Uchida, H. Arimura, "LCM: An Efficient Algorithm for Enumerating Frequent Closed Item Sets", Proceedings of IEEE ICDM'03 Workshop FIMI'03, 2003.

[21] T. Uno, M. Kiyomi, H. Arimura, "LCM ver. 2: Efficient Mining Algorithm for Frequent Closed Maximal Itemsets", Proceedings of IEEE ICDM'04 Workshop FIMI'04, 2004.

[22] M.J. Zaki, "Mining Non-Redundant Association Rules," Data Mining and Knowledge Discovery, vol. 9, no. 3, pp. 223-248, 2004.

[23] C. Lucchesse, S. Orlando, and R. Perego, "DCI-Closed: a fast and memory efficient algorithm to mine frequent closed itemsets", Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04), volume 126 of CEUR Workshop Proceedings, Brighton, UK, Nov. 2004.

[24] T. Uno, M. Kiyomi, H. Arimura, "LCM ver.3: Collaboration of Array, Bitmap and Prefix Tree for Frequent Itemset Mining", Proceedings of Open Source Data Mining Workshop on Frequent Pattern Mining Implementations 2005. Aug. 2005.

[25] T. Uno, T. Asai, Y. Uchida, and H. Arimura," An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases", IEEE ICDM workshop FIMI'04, Zaki & Goethals, Nov, 2004.

[26] C. Lucchese, S. Orlando, and R. Perego, "Fast and Memory Efficient Mining of Frequent Closed Itemsets", IEEE Transactions on Knowledge and Data Engineering, Vol. 18, No. 1, pp. 21-36, Jan. 2006.

[27] J. Sribuaban, V. Boonjing, and J. Werapun, "Frequent Closed Multi-dimensional Multi-level pattern mining," Proceedings of the Fourth IASTED International Conference on Advances in Computer Science and Technology, Malaysia, pp. 201-206, Apr. 2008.

[28] J. Wachiramethin and J. Werapun, "BPA: A Bitmap-Prefix-tree Aarray data structure for frequent closed pattern mining," Proceedings of the Eighth International Conference on Machine Learning and Cybernetics, Baoding, China, pp.154-160, Jul. 2009.

[29] C. Vajiramedhin and J. Werapun, "EBPA: An Efficient Data Structure for Frequent Closed Itemset Mining," Applied Mathematical Sciences, vol. 7 no. 30, pp. 1483-1506, 2013.

ภาคผนวก

ภาคผนวก ก

ผลงานตีพิมพ์บทความวิจัย

1. Chakarin Vajiramedhin and Jeeraporn Werapun: " EBPA: An Efficient Data Structure for Frequent Closed Itemset Mining," Applied Mathematical Sciences, vol. 7 no. 30, pp. 1483-1506, 2013.
2. Jugkarin Wachiramethin and Jeeraporn Werapun, "BPA: A Bitmap-Prefix-tree Aarray data structure for frequent closed pattern mining," Proceedings of the Eighth International Conference on Machine Learning and Cybernetics, Baoding, China, pp.154-160, Jul. 2009.
3. Jugkarin Sribuaban, Veera Boonjing, and Jeeraporn Werapun, "Frequent Closed Multi-dimensional Multi-level pattern mining," Proceedings of the Fourth IASTED International Conference on Advances in Computer Science and Technology, Malaysia, pp. 201-206, Apr. 2008.

Applied Mathematical Sciences, Vol. 7, 2013, no. 30, 1483 - 1506
HIKARI Ltd, www.m-hikari.com

EBPA: An Efficient Data Structure for Frequent Closed Itemset Mining

Chakarin Vajiramedhin

Department of Computer Science, Faculty of Science
King Mongkut's Institute of Technology
Ladkrabang, Bangkok, 10520, Thailand
chakarin@live.com

Jeeraporn Werapun

Department of Computer Science, Faculty of Science
King Mongkut's Institute of Technology
Ladkrabang, Bangkok, 10520, Thailand
ksjeerap@kmitl.ac.th

Abstract

In closed itemset mining, the process of mining from a large transaction database directly often leads to inefficient space and time. Practically, many data structures were proposed to maintain valuable data for frequent closed itemset mining (FCIM), while each data structure has its own advantages and disadvantages. In recent study, a collaboration of array, bitmap, and prefix tree was proposed to gain advantages of those basic data structures by reducing the computing time of the FCIM. That collaboration can save space over that of the original prefix tree, which requires extra space for (m-1) parent-child pointers and its corresponding hashing table (in each tree-node). However, the extra sorting all transactions and merging repeated transactions are required before constructing the prefix tree. Therefore, this paper presents the improved collaboration data structure, called the EBPA (Efficient Bitmap-Prefix-tree Array), with the efficient (parent-child) access in $O(1)$ by using a (temporary) pointer array (without space for hashing) and not require the extra sorting transactions. In system performance evaluation, experimental results showed that the response time of our EBPA-based FCIM mining outperforms over that of the existing collaboration-based FCIM approach.

Keywords: data mining, closed itemset mining, bitmap-prefix-tree array data structure

1 Introduction

Frequent Closed Itemset Mining (FCIM) is the main important technique in several data mining applications (e.g., classifiers, association rules, etc.), for representing useful extracting patterns (or itemsets) from all of very large candidate patterns within a transaction database for solving the problem of a frequent itemset mining (FIM). The FIM approaches are possible to create an exponential number of output patterns, especially in case of the minimum support threshold is set to low value, while the transaction database is very large. Later research focuses on the complete closed itemsets, which can reduce a number of itemsets without information loss and can represent covering all results of the original FIM with saving memory space and time. Therefore, many FCIM approaches have been proposed [1] - [14].

Besides the best solution, an appropriate data structure and corresponding frequency computation functions are a key to improve the performance of each method. Recently, many data structures for computing the FCIM have been proposed ([4], [7], [10], [12], [14]).

In 2002, CHARM algorithm and IT-TREE data structure [4] were proposed for finding frequent closed itemsets. That approach is efficient in frequency counting over FCIM with using inverted lists data structure. However, it requires large memory space for storing node information ((m-1) pointers and its hashing). The frequency computation and the closure process of that prefix tree approach are still the heaviest task of the FCIM, especially the process of the longest path of the prefix tree ($\leq m$ items).

In 2003, the array list data structure was introduced to support LCM approach [7], by storing each transaction of the database in the array lists. That array list-based method is efficient in computation for a sparse transaction database, but it is still weak in a dense transaction database [10].

In 2006, the vertical bitmap data structure was improved over the original bitmap to support DCI-CLOSED approach [12]. That data structure is efficient in memory space for the dense transaction database, especially to save memory in storing data by using only one bit (0 or 1) for each item, represented in all itemsets. However, that method may not be efficient in time ($O(mT^2)$) in the sparse transaction database since its corresponding bitmap matrix containing many *0*s but the process is equal to mT^2 fixed steps, where m is a number of frequent 1-itemset and T is a number of transactions in the database.

The collaboration of array lists, vertical bitmap, and prefix-tree data structure [10] was proposed in 2005 to utilize the advantages of those basic data structures for making more efficient in computation time and saving more memory space. In particular, the (compact) prefix tree and (bucket) array lists are combined to

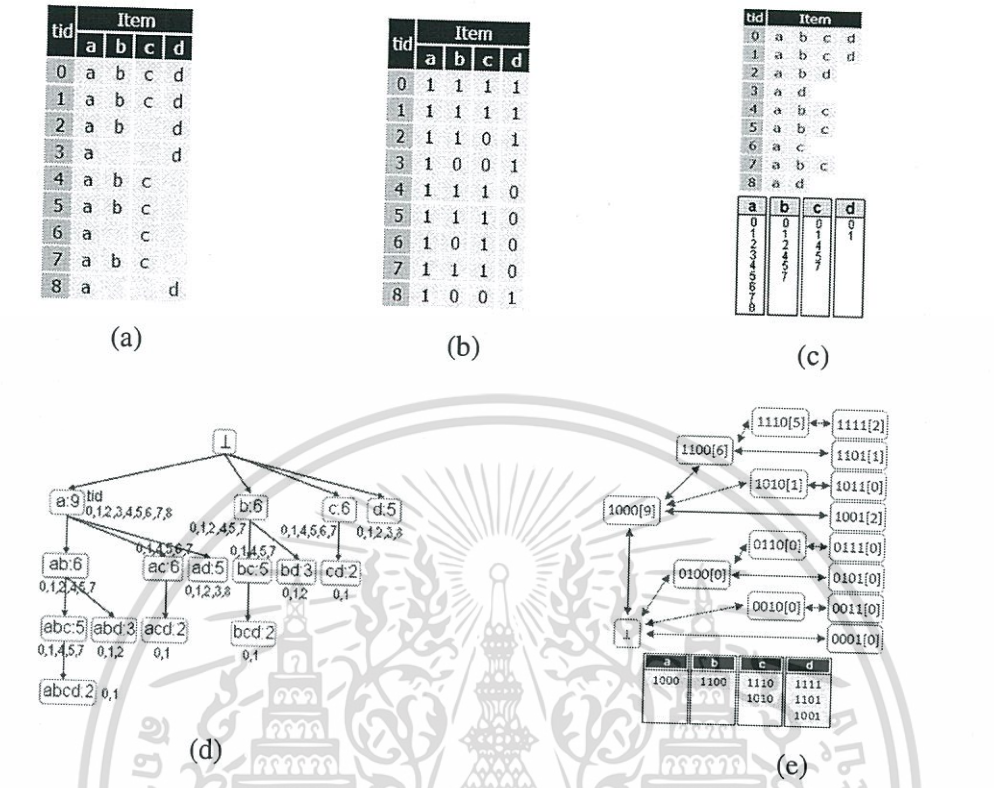


Fig. 1 a) an original transaction dataset, containing four 1-itemsets (*a, b, c* and *d*) and existing data structures represent nine transactions, b) Vertical Bitmap Matrix, c) Array List, d) Prefix tree, and e) Collaboration (Array, Bitmap, Prefix tree).

reduce the FCIM computing time over that of the (original) prefix tree in CHARM [4] and the array list in LCM [7]. In addition, using the bitmap (transaction) node and no extra space for (m-1) parent-child pointers and hashing (in each node). So far, the collaboration method is efficient but it requires time to sort all transactions and combine the repeated transactions before constructing the prefix tree (from the leaf nodes to the root), corresponding to the specific transaction order.

In this paper, we propose the improved collaboration data structure, called the EBPA (Efficient Bitmap-Prefix-tree Array) and the EBPA-CLOSED algorithm for frequent closed itemset mining. The EBPA data structure, the (top down) prefix-tree construction, is an improved version of our previous work, the (bottom up) prefix-tree BPA [14]. In order to gain the advantage of the collaboration approach, our EBPA maintains all features and results of the existing collaboration data structure [10] without extra sorting and merging transactions. In our EBPA data structure, the efficient (parent-child) access is introduced in *O(1)* time by using a temporary pointer array (in each node) but no hashing space and

no extra sorting all transactions. Our method provides not only saving space but also saving time that is faster and easier to access each (parent-child) node of the prefix-tree array for filling and counting the frequency. Finally, like the existing collaboration, the result of our EBPA provides the faster computing in item-based buckets (arrays) for the closure process of the FCIM mining. In addition to saving space, we design the EBPA that creates the compact prefix tree, containing only occurrence nodes from the transaction database (with storing 2 integers (*tid* and *w*) in each node) and uses the shared (bitmap) itemsets among corresponding nodes along the same path (of the prefix tree) for saving more space.

The remainder of this paper is organized as follows: Section 2 provides a concise survey of related work. Section 3 presents our efficient EBPA data structure and the EBPA-CLOSED algorithm for the FCIM mining. Section 4 displays the performance evaluation and experimental results. Finally, conclusion and future study are discussed in Section 5.

2 Related Work

The FCIM, first proposed by Pasquier et al.[1] in 1999, is an interesting alternative solution for representing useful extracting patterns from very large candidate patterns within transaction database. The FCIM is known as an efficient mining technique because of interesting only the frequent closed itemsets instead of mining the complete set of frequent itemsets. The complete itemsets derived from this method can reduce a number of itemsets without information loss, where as it can represent or cover all results of the original FIM. Therefore, the FCIM approach saves more time to search only the frequent closed itemsets without using a huge space for keeping all result patterns.

Let $D = \{t_1, t_2, t_3, \ldots, t_T\}$ be a transaction database. Each t_i is a transaction (i = 1, 2, 3, ..., T) in the database consisting of a transaction identifier (*tid*) and items ordered from 1 to k items ($i_1, i_2, i_3, \ldots, i_k$).

Definition 1: Let $I = (i_1, i_2, i_3, \ldots, i_k)$ be a set of items in transaction database that every subset P of I is called an itemset. The itemset P with k items is called a *k-itemset*. The number of transaction in D matching the itemset P is called the support of P, denoted as *supp(P)*. Given a minimum support threshold *min_supp*, the itemset P is called a frequent itemset if and only if $supp(P) \geq min\_supp$.

Example 1: Form input transaction database in Fig.1a, there are four 1-itemsets (*a*, *b*, *c*, and *d*)) and suppose minimum support threshold is 5. The frequent 1-itemsets are (*a*:9), (*b*:6), (*c*:6) and (*d*:5). Therefore a set of frequent itemsets is {(*a*:9), (*b*:6), (*c*:6), (*d*:5), (*ab*:6), (*ac*:6), (*ad*:5), (*abc*:5)}, because their occurrences (or support) are equal to or more than 5 (support $\geq$ 5) that pass the minimum support threshold.

Definition 2: The itemset P is a closed itemset if there is no superset and can represent all itemsets that belong to the same equivalence class with the same

support. A closed itemset P is frequent if its support passes the given support threshold ($supp(P) \geq min\_supp$).

Example 2: From input transaction database in Fig.1a, suppose minimum support threshold is 5. The itemsets *a:9*, *ab:6*, *ac:6*, *ad:5*, and *abc:5* are called the closed frequent itemsets (see Fig.2), because their occurrences are equal to or more than 5, passing the given support threshold, and can represent the belonging itemsets in the same equivalence class with the same support.

Fig.2 shows the lattice of frequent itemsets and closed frequent itemset derived from the input transaction database from Fig.1a. For example, the itemset *cab:5* is a closed frequent itemset, because it can represent itemsets *c:5*, *ca:5*, *cb:5*, and *cab:5*, etc. That approach collects five itemsets only, instead of storing all itemsets, and hence can save more space. The closed itemsets are the results of the frequent closed itemsets mining (FCIM) representation of all frequent itemsets in the same support of equivalence class extracted from the transaction database. In the past ten years, many FCIM techniques [4], [7], [10], [12] were proposed to solve the data storage problem with compacted data by storing a set of representative itemsets that can cover all other itemsets. Each technique has its specific function and data structure that have some advantages and disadvantages to tradeoff. The performance keys of each technique are the efficient data structure construction, including the fast frequency computation function.

The vertical bitmap data structure was developed to support DCI-CLOSED approach [12] (Fig.1b) and its efficacious FCIM traversed the search space in a depth-first manner. This data structure is a memory space efficient structure for the dense transaction database, especially in saving memory to store data in main memory that represents all itemsets in the input transaction database by using only one bit (0 or 1) for each item (or mT bits for all transactions (T)), where m represents a number of frequent 1-itemsets. However, this method may not be efficient in time ($O(mT^2)$) for the sparse transaction database since its corresponding bitmap matrix containing many 0s but the process is always equal to mT^2 iterations. Therefore, that data structure will take long time to compute the frequency of itemsets in case of there are a lot of transactions (T) in the database where as there are a few number of items in each transaction.

The array list data structure was used to support the original LCM [7]. In this (inverted) list-based approach (Fig.1c), each of T transactions in the database is scanned and stored in frequent 1-itemset buckets in $O(mT)$ (see Fig.1c). A number of array lists are equal to the number of the frequent 1-itemset (m) and the length of each array is equal to the frequency of each 1-itemset. The array list data structure computes the frequencies of itemsets by scanning from the lists of the 1-itemset in m buckets. Thus, the array list-based method yields an efficient computation for a sparse transaction database, but it is still weak in any dense transaction database.

The IT-TREE data structure (see Fig.1d) and CHARM algorithm [4] were proposed for finding frequent closed itemsets. The process of that approach is based on the prefix tree and (parent-child) hashing for support both dense and sparse transaction databases that store an itemset ($\leq m$ bytes) in each node of the prefix tree, where m is a number of frequent 1-itemsets. Each node of the tree contains an item identifier ($\leq m$ bytes), a support (or frequency), a parent-node pointer, child node indices ($\leq m$), and a hashing table. The prefix tree is constructed from the root and the first level contains 1-itemsets only and their (inversed-list) frequency counting for T transactions in $O(mT)$, where m is a number of levels (for m 1-itemsets). More time are required for adding all corresponding nodes (in other levels) of the prefix tree. The main advantage of using IT-TREE in CHARM is that frequent searching with the hashing (parent-to-child nodes) is efficient. However, that approach may require large memory space for storing node information (in bytes) in all nodes ($n \leq N = 2^0 + 2^1 + \ldots + 2^{m-1} = 2^m - 1$). The frequency computation and the closure process of that prefix tree are still the heaviest task of the FCIM mining, especially the process of the longest path of the prefix tree ($\leq m$ items).

Lately, the collaboration data structure [10] was introduced (see Fig.1e) to combine three basic data structures (array lists, vertical bitmap, and prefix tree) to utilize advantages of each data structure for making more efficient in computation

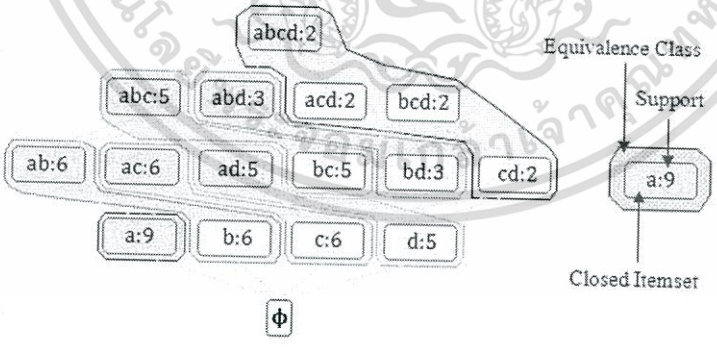


Fig.2 A lattice of the frequent closed itemset (*min_supp*=1) mine the input data from Fig.1a.

time and saving more memory space. Such a collaboration data structure was developed to support the efficient LCM (version 3) for mining large transaction

database, including dense and spare databases. In that collaboration, the (compact) prefix tree and (bucket) arrays are combined to save time of frequency computing and closed processing over that of the (original) prefix tree in CHARM [4] and the array list in original LCM [7]. In addition, the (bitmap) transaction in each tree-node is used to save space over the construction of the prefix tree in CHARM (since there is no extra space required for (m-1) parent-child pointers and hashing (in each node)). However, that collaboration approach requires the extra sorting of all T transactions (m elements per transaction), according to decreasing order of levels and prefix items in $O(mT)$, to merge the repeated transactions before constructing the (bottom up) prefix tree, corresponding to that transaction order. After merging the repeated transactions, the collaboration data structure is created, as follows: Each of T' ($\leq T$) unique transactions is scanned to fill its initial frequency into the corresponding node (of the prefix tree), containing an m-bitmap array and a weight) from leaf nodes to the root) in $O(T')$. During move to fill frequency of nodes in the next (lower) level (of m levels), the frequency of each child node is shared to corresponding parent nodes (for m levels (n_i nodes in each level i)). Therefore, time complexity of the collaboration to construct the (compact) prefix tree is $O(mT) + O(mn_iT')$, where $n = max(n_i)$, $n_i \leq 2^i$ and $i = 0, 1, 2, \ldots, m$-1. In addition, to save space, that data structure stores bitmap-itemsets in binary (0/1) format, like the vertical bitmap, in each node of the prefix tree. Next process is performed in separate array buckets, which store only occurrence (transaction) nodes and process the frequency counting of the remaining itemsets in the buckets for efficient FCIM mining. Finally, the efficient FCIM processing in LCM3 is computed by applying the ppc-extension (prefix preserving closure process) algorithm [11] and the collaboration data structure [10]. Practically, the collaboration approach is efficient for the FCIM mining since using bitmap-node (without (m-1)-pointers) to save memory space and using prefix-tree plus bucket arrays to save processing time of the FCIM mining. However, the response time of that collaboration in LCM3 can be improved if its process does not require extra sorting to order all T transactions before constructing the prefix tree.

3 The improved Collaboration (EBPA) Data Structure for FCIM Mining

In this section, we present the improved collaboration data structure, called "the EBPA (Efficient Bitmap Prefix-tree Array) data structure" based on the efficient (parent-child) access of the prefix-tree array in $O(1)$ and the EBPA-CLOSED algorithm to improve both time and space for the FCIM mining. The contribution of our FCIM mining includes the following functions:

1. Propose the (parent-child) access ($O(1)$) for the (compact) prefix tree (in Section 3.1).

2. Design the efficient EBPA data structure that utilizes the (compact) prefix tree and (bucket) arrays with no extra sorting before constructing the prefix tree (in Section 3.2).
3. Improve the ppc extension [11] for the FCIM mining with the pre-test technique to look over unnecessary closure sets and post sets and save the response time of the FCIM computing (in Section 3.3).

3.1 The (Temporary) Pointer Array for Efficient Parent-Child Access: *O(1)*

Fig.3a illustrates an example of the (compact) prefix tree (of four items (*a*, *b*, *c*, *d*)) that node indexing (in each level) are assigned corresponding to order of incoming transactions. Fig.3b depicts the (complete) prefix tree, where node indexing (in each level *i*) are assigned with prefix ordering (0, 1, 2, …, n_i-1, where n_i = a number of nodes in level i (= 2^i) and i = 0, 1, 2, … , m-1. Practically, the (compact) prefix tree requires a counter (c_i) in each level i to set index (for each bucket array without any fragment) and hence the number of occurrence nodes ($n_i \le 2^i$) in each level i can be linear up to exponential nodes.

Let $b_0b_1b_2 \ldots b_{m-2}b_{m-1}$) represent the (bitmap) transaction or itemsets.
ptr represent a (temporary) pointer array with *O(1)* access (equation (1)).
l_p represent the level of the parent node, where $l_p = p$ and $0 \le p \le m$-1.
l_i represent the level of the child node, where $l_i = i$ ($> p$) and $0 < i \le m$-1.
c_i represent the counter for setting (compact) node-index in each level i.

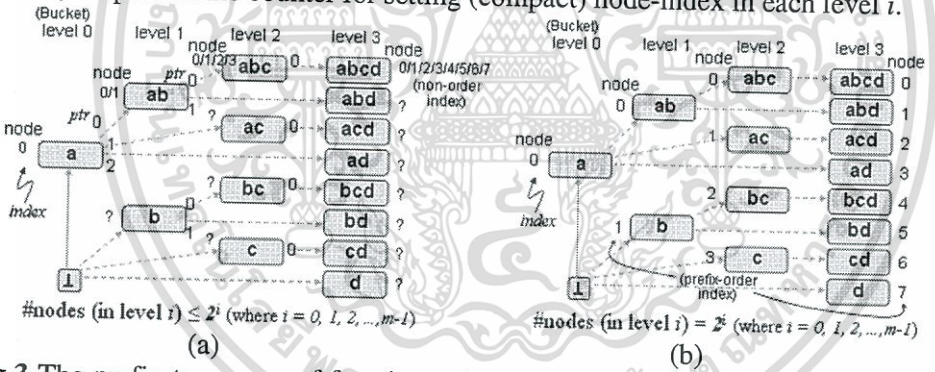


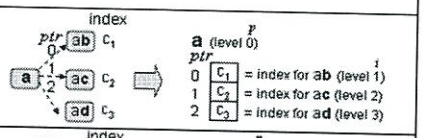
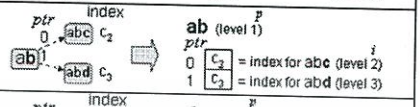
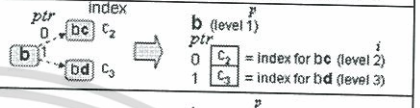
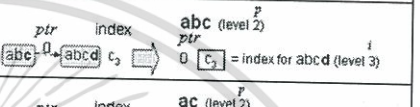
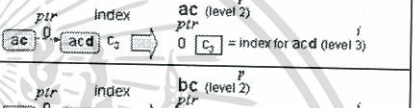
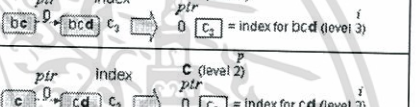
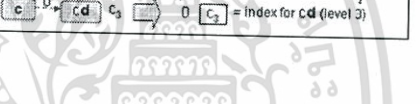
Fig.3 The prefix-tree array of four items (*a*, *b*, *c*, *d*): a) the (compact) prefix tree (with non-order index), b) the (complete) prefix tree (with prefix-order index).

During construction of the (compact) prefix tree, the (temporary) pointer array (*ptr*) is required for each (internal) occurrence node. A number of child nodes for a parent are m-p-1 nodes, depending on a number of items (m) and prefix-items upto the parent node (p). The location of that pointer array (m-p-1 elements) for each particular parent-child can be 0, 1, 2,…, or m-p-2, as defined in equation (1).

$$location\ of\ ptr = l_i - l_p - 1,\ \ where\ \#childs = m - p - 1 \qquad (1)$$

For each (bitmap) transaction, the *lptr* (in equation(1)) is computed in *O(1)* to point to corresponding child directly, related to levels of child ($l_i = i$) and parent ($l_p = p$), illustrated in Table 1 for all internal nodes of the prefix tree.

Table 1. The (temporary) pointer array for all (internal) nodes of the prefix tree.

node	*bitmap*	*level p*	l_i=i	location of *ptr*	
a	**1**000	0	i=1 i=2 i=3	*lptr* =1-0-1 = 0 *lptr* =2-0-1 = 1 *lptr* =3-0-1 = 2	
ab	**11**00	1	i=2 i=3	*lptr* =2-1-1 = 0 *lptr* =3-1-1 = 1	
b	**01**00	1	i=2 i=3	*lptr* =2-1-1 = 0 *lptr* =3-1-1 = 1	
abc	**111**0	2	i=3	*lptr* =3-2-1 = 0	
ac	**101**0	2	i=3	*lptr* =3-2-1 = 0	
bc	**011**0	2	i=3	*lptr* =3-2-1 = 0	
c	**001**0	2	i=3	*lptr* =3-2-1 = 0	

3.2 The Construction of the EBPA Structure

We design the efficient EBPA data structure as an array-based (compact) prefix tree for saving space (in practice) that concerns only occurrence transactions and their corresponding parent nodes, an improved version of our previous data structure (BPA) [14] that exponentially generates full nodes (N) of the complete prefix tree (Fig.3b), where $N = 2^0 + 2^1 + 2^2 + \ldots + 2^i + \ldots + 2^{m-1} = 2^m - 1$ nodes. In particular, there are two main steps in the collaboration based EBPA-FCIM mining: 1) create the (compact) prefix tree (Fig.4b) and the (bucket) arrays (Fig.4c) in Section 3.2 and 2) compute the closure sets and post sets of the FCIM mining (see Section 3.3). Our focus in this section is the first part, the EBPA (Efficient Bitmap-Prefix tree Array) data structure, which is introduced to improve space and time of the construction of the (compact) prefix tree array (from T transactions directly without extra sorting), while yielding similar results to the original collaboration in LCM3 [10]. In our EBPA structure, the (temporary) pointer array *ptr* (of m-p-1 elements with initially setting to -1) is introduced with efficiently accessed in $O(1)$ by using a specific location *lptr* (equation(1)) and a counter (c_i) in each level i to access parent-child nodes ($< m$) (without hashing). Note that the initial counter c_i is set to -1 ($c_i = -1$) and it is incremented by one ($++c_i$) before adding the new node (in level i) for the corresponding transaction. Therefore, time complexity of the frequency computing (along the same path) for each transaction is $O(m)$ for existing nodes and $O(m^2)$ for new nodes and for all T transactions is $O(m^2n) + O(mT)$, where $n = \max(n_i)$ and $n \le 2^i$, $i = 0, 1, 2, \ldots, m$-1.

Algorithm 1: Constructing of EBPA data structure.

Step 1: Initial process **[$O(kT)$]**

1.1 scan all transactions to find some parameters:
T (#transactions), *k* (#all itemsets), *f* (frequency of each item), *l* (level of each transaction), and set minimum support threshold to find *m* (#frequent itemsets).
1.2 sort *m* frequent itemsets in descending order of item- frequency.
1.3 convert all *T* transactions in a bitmap 0/1 format (*m*-bit).

Step 2: EBPA construction **[$O(m^2n) + O(mT)$]**

2.1 start with scanning (bitmap) transaction for finding the level (*i*) containing 1-bit.
2.2 check in level *i* of (compact) prefix array whether there exists that node or not. if yes (i.e., identifier ≥ 0), increment frequency; otherwise (i.e., identifier =-1) create node and set node information (*tid*, *w*, *m-p-1* (temporary) pointers).
2.3 set current node to be parent (level *p*) and scan next 1-bit (level *i*) and apply eq.(1) (location of *ptr* = *i-p*-1) to link to child node (level *i*) and repeat step 2.2 – 2.3 until the end of transaction.
2.4 move to the next transaction and repeat step 2.1 - 2.4 until the last transaction.

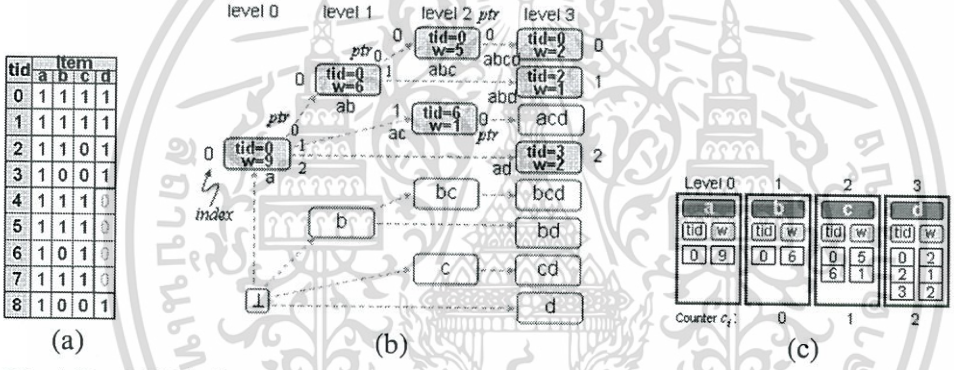


Fig.4 Our EBPA data structure: a) the input dataset; b) the (compact) prefix tree; and c) the (bucket) arrays.

The compact prefix-tree array (Fig.4b) is an associative array that a number of levels in prefix array are equal to a number of 1-itemsets (i.e., *a*, *b*, *c*, *d*). Each node contains only a support (*weight w*) and a transaction *id* (*tid*) to the shared bitmap array of the same branch of the prefix-tree array. In our approach, the bitmap-array table (Fig.4a) contains a number of shared bitmaps (0/1) of related transactions or nodes along the same path in the prefix tree (i.e., a shared bitmap 1111 for itemsets *abcd* (4 bits) upto level 3, *abc* (3 bits) upto level 2, *ab* (2 bits) upto level 1, and a (1 bit)) in level 0. The size of the bitmap array is $T \times m$, where T is a number of transactions in database and m is a number of 1-itemsets that their support values ≥ *min_supp*. The construction of our EBPA data structure is illustrated in Algorithm 1, which composes of two main steps:1) the initial process and 2) the EBPA construction.

First, step 1.1 loads the input file to initialize memory and scans all transactions (in the database) for finding frequency of each item (f_i), *m* 1-itemsets,

a number of transactions (T). Then, step 1.2 sorts m frequent itemsets in descending order of item-frequency. Next, step 1.3 allocates the bitmap array and set 0 to initialized data. Then, scan all transactions by m frequency order pattern and set 1 into bitmap array in the frequent items. For example, Fig.4a illustrates the bitmap array, the results of step 1 ready to generate the EBPA data structure.

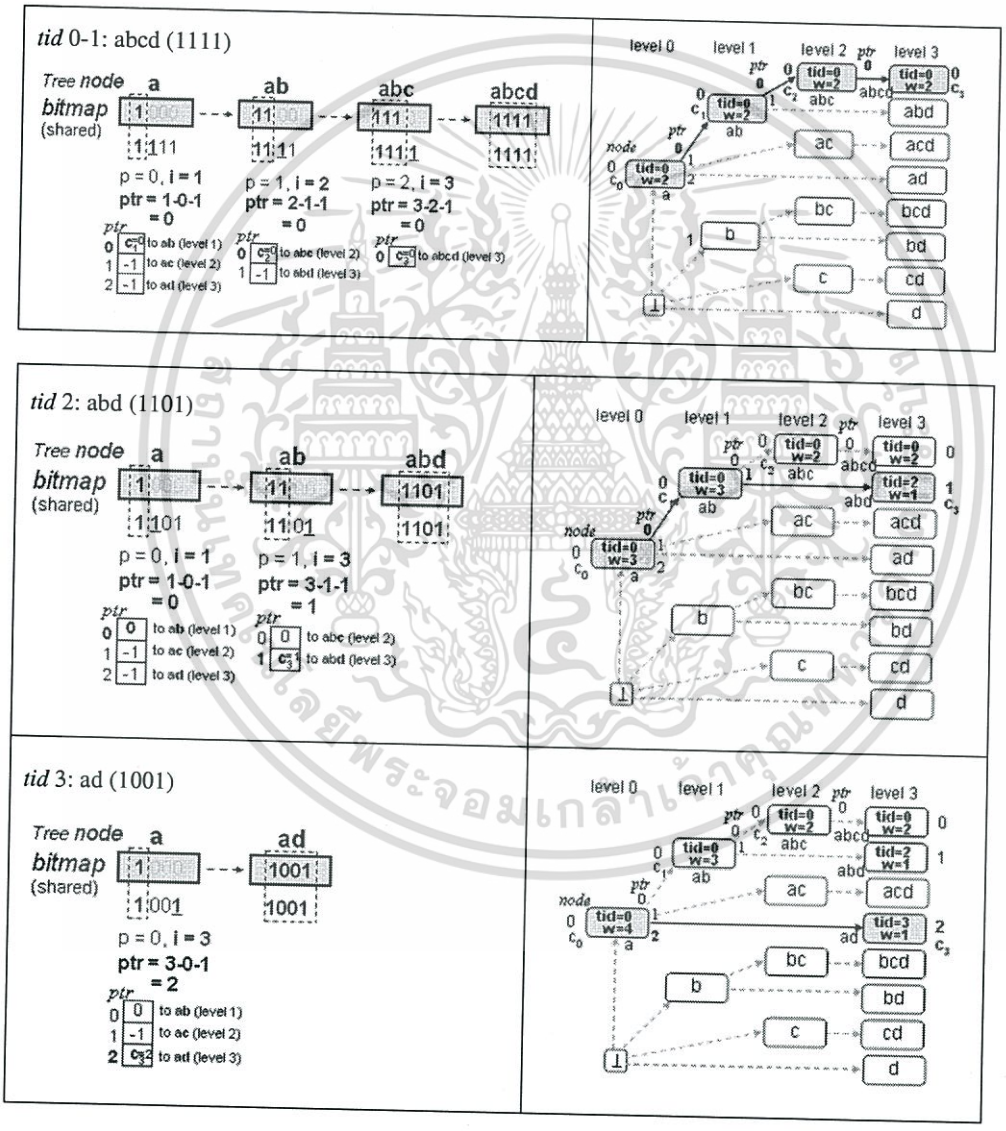
In step 2, the existing collaboration [10] requires extra sorting T transactions (m-bit itemset per transaction) and their merging the same repeated transactions. Its prefix-tree arrays are constructed (i.e., fills the frequency back) from level m-1 (leaf nodes) to level 0 (the root), according to sorted (T') unique transactions. Applying the compact prefix-tree array of that collaboration may confront with many difficulties (i.e., requiring extra sorting, finding child-parent relation, etc.) and take long time in cases of a large amount of 1-itemsets to build the prefix-tree array. In our top-down EBPA approach (with our efficient parent-child access in $O(1)$), we construct the compact prefix-tree array (without extra sorting) by storing only occurrence transactions and their parent nodes. In our approach, fill frequency processing time depends on a number of occurrence items ($\leq m$) in each transaction. Thus, time complexity of filling frequency for each transaction is $O(m)$ for existing nodes or $O(m^2)$ for new nodes (with m-p-1 temporary pointers per node) and $O(m^2n) + O(mT)$ for all T transactions and there exist n nodes in the (compact) prefix tree. In step 2.1, we start with scanning the bitmap transaction for finding the level containing the first 1-bit, representing the first item of that transaction (i.e., assume level i). Next, step 2.2 checks in level i of the compact prefix array whether the current node was created in the level i or not. If that node does not exist, increment the counter of level i ($++c_i$), create that node in level i (at index = c_i) and set information (i.e., a transaction *id* number (*tid*), the support (*weight*=1)), and allocated the (temporary) pointer array (*ptr*) to link to its (m-p-1) child nodes with initial setting to (-1). In case of that node was created already, increment frequency (or weight) in that child node. Then, step 2.3 sets the current node to be the next parent (in level p) and scans the same transaction to find out the level of the next 1-bit (i.e., level $i > p$). From the parent node (in level p) and the particular child (in level i), apply equation (1) to jump to that child node directly. Then, repeat step 2.2–2.3 for the next item (or 1- bit) until the end of transaction. Next, move to the next transaction and repeat the same process (step 2.1–2.4) until the end of dataset (with the last transaction). Lastly, free the (temporary) bucket array for returning the memory space.

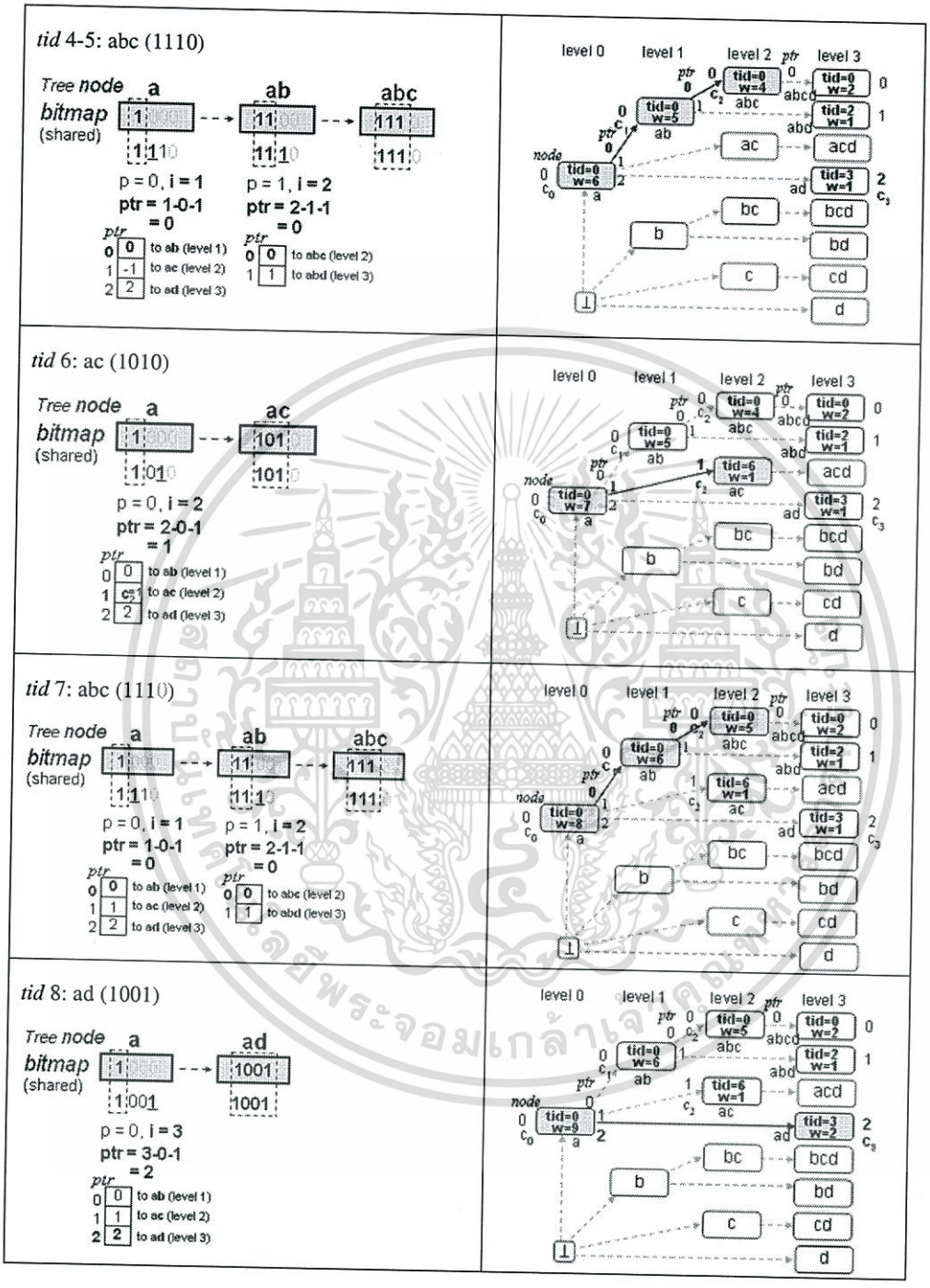
In our EBPA data structure, the (compact) prefix tree array is constructed for each of nine transactions (*tid*0 – *tid*8) of the input dataset (Fig.4a), as follows:

For each *tid* 0 – 1 (*abcd*: 1111), the 1st and 2nd items are a (parent, level p=0) and b (child, level i=1) and hence the location of $ptr = i - p - 1$ = 1-0-1 = 0 (that is $ptr[0] = ++c_1 = 0$) and then weight w of node[c_1=0] in level 1 is incremented. Next child of that itemset is c (level i=2) with parent b (level p=1) and the location of $ptr = i - p - 1$ = 2-1-1 = 0 (that is $ptr[0] = ++c_2 = 0$) and then weight w of node[c_2=0] in level 2 is increased. Finally, last child of that itemset is d (level i=3) with parent c (level p=2) and the location of $ptr = i - p - 1$ = 3-2-1 = 0 (that is $ptr[0] = ++c_3 = 0$) and then weight w of node[c_3=0] in level 3 is increased.

Next, for *tid* 2 (*abd*: 1101), the first and second items are *a* (parent, level p=0) and *b* (child, level i=1) and hence the location of $ptr = i - p - 1$ = 1-0-1 = 0 (and $ptr[0]$ = 0) and then weight w of node[0] in level 1 is added. Lastly, next child of that itemset is *d* (level i=3) with parent *b* (level p=1) and the location of $ptr = i - p - 1$ = 3-1-1 = 1 (that is $ptr[1] = ++c_3 = 1$) and then weight w of node[c_3=1] in level 3 is added.

For *tid* 3 (*ad*: 1001), the first and second items are *a* (parent, level p=0) and *d* (child, level i=3) and hence the location of $ptr = i - p - 1$ = 3-0-1 = 2 (that is $ptr[2] = ++c_3 = 2$) and then weight w of node[c_3=2] in level 3 is incremented.





For each *tid* 4 – 5 (*abc*: 1110), the first and second items are *a* (parent, level p=0) and *b* (child, level i=1) and hence the location of $ptr = i - p - 1$ = 1-0-1 = 0 (and $ptr[0] = 0$) and then weight w of node[0] in level 1 is incremented. Lastly, next child of that itemset is *c* (level i=2) with parent *b* (level p=1) and the location of $ptr = i - p - 1$ = 2-1-1 = 0 (and $ptr[0] = 0$) and then weight w of node[0] in level 2 is incremented. Similar processes are repeated for *tid* 6 (ac or 1010), *tid* 7 (abc or 1110), and *tid* 8 (ad or 1001), as illustrated in the above figures.

Note: the shared-bitmap array is introduced in our approach in order to save more space in each node since only transaction *id* number (*tid*), referring to the bitmap of each itemset, is stored and the corresponding bitmap among nodes along the same path are shared (upto level i of the last item), as illustrated in the above example.

After completing (fill/increment frequency) all T transactions, our efficient (compact) prefix-tree array reduces the frequency searching area for the FCIM process by decomposing the large transaction database (T transactions) into the compact prefix arrays (without any fragment) for making short lists collected only related data without generated the complete prefix tree (including some fragments). In our (compact) prefix tree array, we apply "(temporary) pointer" array (in each occurrence node) for faster accessing child node and use "shared bitmap" for saving more space. Finally, we will free the memory storing the temporary pointer array before processing closed mining step of the FCIM in (bucket) arrays (in Section 3.3). Compared to the prefix-tree array of the original collaboration [10], time complexity of that prefix tree is $O(mT)+O(mn_iT')$, depended on T' (a number of unique transactions), while our time complexity is $O(m^2n)+O(mT)$, based on n (a number of occurrence nodes in the compact prefix-tree array). Practically, for most of transaction databases, a number of occurrence nodes are less than a number of all unique transactions.

After finishing the compact prefix-tree array, the array buckets are ready for the FCIM mining step, containing accumulated frequency in each prefix path. The next step of the EBPA-based FCIM computing (in Section 3.3) requires the frequency counting (of some remaining itemsets). Compared to some data structures (i.e., array list, bitmap, and prefix tree), that frequency computing in the FCIM mining is a heaviest task and time consuming. In the collabolation approach, the remaining process is performed efficiently in level-based buckets (of the prefix tree) to simplify the process because processing with a number of nodes in each level (n_i) by the collaboration structure are less than processing with a number of transactions (T), as required in other data structures.

3.3 The EBPA-CLOSED for the FCIM Mining

The efficient FCIM computing (Algorithm 2) is processed faster from the (bucket) arrays of our EBPA data structure. Our EBPA-CLOSED Algorithm consists of two main steps: 1) the initial data and 2) the closed itemsets. The idea of computing the closure process in Algorithm 2 is improved from the ppc-extension (prefix preserving closure extension) [11] by searching all closed itemsets in an

efficient depth-first search manner. Our focus is using the pre-test technique for the frequent closure sets and the frequent (root sub-tree) post sets to reduce uncessary repeated steps of the closure operation. Note: see definitions of the closure set and the post set in Definition 3 and 4, defined in [11].

In step 1, the initial data are prepared for each frequent 1-itemset generator from our EBPA data structure to efficiently counting frequency in the FCIM to be ready for computing suffix closed itemsets of the 1-itemset generators (in step 2). For a generator item, we compute the corresponding (frequent) closure sets and frequent (root sub-tree) post sets, the related data generated from transaction including the generator only, for quickly computing the (level-based) frequency.

Definition 3: Given a generator itemset P_l, the closure sets of P_l are the corresponding itemset of P_l in the lower level than P_l level, denoted as *closure*(P_l). The *closure*(P_l) are the itemsets $i_0, i_1, i_2, \ldots, i_{l-1}$, where, l is the level of a generator itemset.

Definition 4: Given a generator itemset P_l, the post sets of P_l are the corresponding itemset of P_l in the higher level than P_l level, denoted as *post*(P_l). The *post*(P_l) are the itemsets $i_{l+1}, i_{l+2}, i_{l+3}, \ldots, i_{m-1}$, where, l is the level of a generator itemset.

Apply definition 3 (in Algorithm 2) with pre-test for the (frequent) closure sets can reduce the number of closure sets of the generator and reduce the number of suffix generators. Apply definition 4 with pre-test for the frequent (root sub-tree) post set can reduce the number of post sets of the generator. Note that in our approach, the frequent closure set is the closure set that its frequency counting $\geq$ *min_supp* and the frequent (root sub-tree) post set is the post set that is a child of the generator and its frequency counting $\geq$ *min_supp*. For example, Fig.5 shows

Algorithm 2: The EBPA-CLOSED Algorithm.

Step 1: Create Initial data (from 1-itemset generator) **[O(mn)]**

1.1 for each of generator item, scan and count weight of existing itemsets ($n_i \leq n$) in its bucket by go directly in that bucket, where n_i is a number of nodes in level i ($\leq 2^i$) and $n = max(n_i)$, $0 \leq i \leq m\text{-}1$.

1.2 from that generator, find its (frequent) closure set (prefix items of gen-item) from the generator level and frequent (root sub-tre) post set (suffix items of gen-item).

Step 2: Find closed itemsets **[O(mn)]**

2.1 compare frequency of gen-item with (frequent root sub-tree) post sets to find a closed itemset.

2.2 extend intermediate results of step 2.1 by generated new generator from closed itemset and their closure results.

2.3 repeat step 2.1-2.3 for find the new closed itemset (residing in higher level than the generator item) until closure=0 (no closure result).

Step 3: repeat Step 1-2 for other (m 1-itemset) generators. **[O(m^2n)]**

the prefix-tree lattice (with the minimum support=1). For the generator itemset (*a*:12), its post sets are (*b*:9), (*c*:8), (*d*:6), (*e*:4) and (*f*:3). After applying the

pre-test technique, the frequent (root sub-tree) post sets of the generator are (*b*:9) and (*c*:8). Clearly, this case can reduce repeating process for five post sets to two post sets only. Because of the nature of the prefix-tree, the suffix node will have the frequency less than or equal to its prefix node (or its root sub-tree). Therefore, time of repeated check with some (unnecessary) post sets can be skipped.

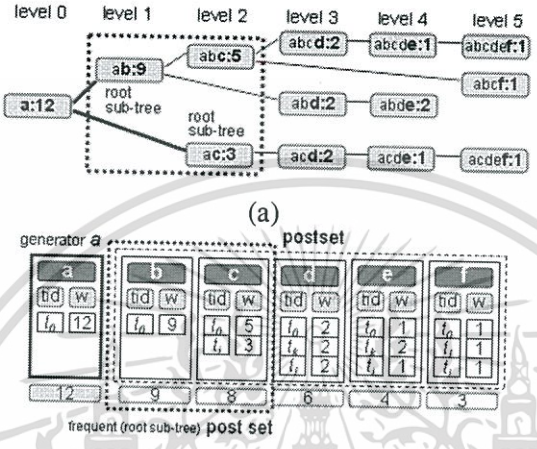


Fig.5 a) An example of the prefix-tree lattice and b) The frequent (root sub-tree) post sets of the generator *a*

Next, we will apply the pre-test technique in the initial data (step 1) from our EBPA data structure to compute the (frequent) closure sets and the frequent (root sub-tree) post sets of the generator efficiently, Like the collaboration data structure in LCM [10] with ppc-extension [11], the initial data are created from the related data of the 1-itemset generator only and are reused for corresponding *k*-itemset generators to find all suffix closed itemsets of the 1-itemset. Thus, the frequency from the initial data will be processed faster (or saving more time) than those of the existing data structures (such as the original bitmap [12], the array list [7], and the original prefix tree [4]). In our approach, the initial data can be created step by step, as follows:

First, we start with setting the generator list to the level of the generator. The generator list contains index links (*tid*) to the bitmap array and the weight (*w*) of each node form the prefix-tree array. Fig.6 illustrates an example of the initial data (of four generators) that are generators of items *a* (in level 0), *b* (in level 1), *c* (in level 2), and *d* (in level 3)).

Next, we scan the lower (position) items than the generator in the bitmap array of the generator to find the frequent closure sets (only their weights $\geq$ *min_supp*) and create lists of the closure sets. Lastly, we scan all nodes for the frequent (root sub-tree) post sets (only childs of the generator with their weights $\geq$ *min_supp*). Time complexity of this process is $O(mn)$ for each generator ($\leq n$ nodes in each level, m elements of closure set and postset) and $O(m^2n)$ for all m generators.

Fig.6c shows the initial data of a generator item *c*, which can be computed step by step, as follows: The generator *c* showed in level 2 of the prefix array having 2 nodes. Each node of level 2 links to index 0 and 1 of the prefix array and (bitmap) *tid* 0 and 6 of the bitmap array, respectively. Then, data of itemset *b* is set for defining closure sets and an itemset *d* for defining post set. Next, we add all *tid* (a link to (shared) bitmap of itemsets) and *w* (their corresponding weights) into the generator *ca* and its corresponding closure (*b*) and postset (*d*), for the initial data. The generator itemset *ca* (in level 2) occurs at location 0 and 6 of the bitmap array with weights 5 and 1. At level 2, the itemset *b* occurs at location 0 with weights is 5. The itemset *d* data is defined (in the next level 3) for computing the postset, which is weight 2 at location 0. Similar processes are performed to define the initial data for other generator *a* (Fig.6a), *b* (Fig.6b), and *d* (Fig.6d), including the frequency counting.

In the frequency computing, the searching area is the main issue affecting the performance of the FCIM process, such as the bitmap matrix [12] always computes the frequency in all transactions and the array list [7] finds the frequency by searching in bucket arrays, which are time consuming, especially on large databases. For example, if we compute the frequency of the generator itemset *c* in the bitmap format (see Fig.1b) and the array list (see Fig.1c) directly, that process takes 9-step and 6-step loops, respectively, while our EBPA-based approach takes a shorter computation time (2 steps only), see Fig.6c. Thus, our processing time is efficient, especially in case of the long transaction database.

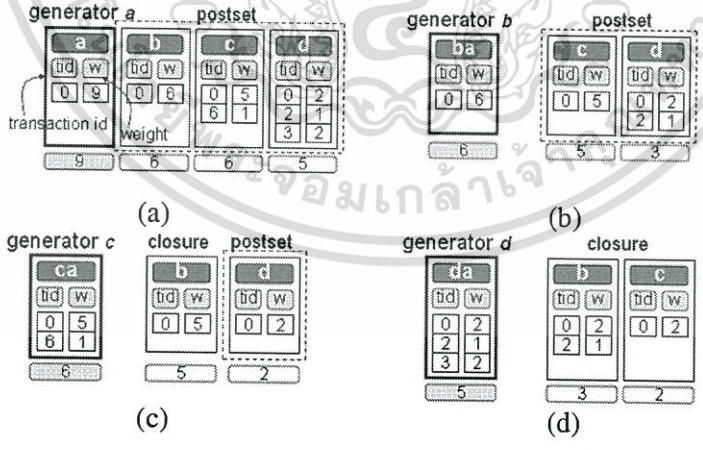


Fig.6 Initial data of 1-itemset generator (*a*, *b*, *c* and *d*).

Like the original collaboration in LCM-3 with ppc-extension [11] time complexity of our EBPA-based FCIM for creating a generator and its frequent closure sets and frequent (root sub-tree) post sets for frequency counting is $O(mn)$ and hence $O(m^2n)$ for all m generators since there are $\leq n$ nodes in each level, $\leq m$ elements of (frequent) closure sets and post sets. Note: applying the pre-test for efficient closure sets and post sets often provides the best case process in O(mn).

According to our initial data (with pre-test for efficient closure sets and post sets for each generator) and shared bitmap of corresponding itemsets, we design a more efficient computing and memory saving technique. In our approach, we generate the initial data of each 1-itemset generator first (in step 1) and then reused them for finding all suffix closed itemsets (or k-itemset generators in higher levels) of the 1-itemset generator (in step 2). For example, the result in Fig.6c shows the initial data that are prepared for the generator (itemset ca) from the EBPA data structure, including the data of the generator, the closure set (itemset b) and the post set (itemset d). Form this initial data, we can find all suffix closed itemsets of the 1-itemset c (up to level 2).

For the generator itemset ca (with one closure set (b) and one postset (d)), we can find total frequency of the generator and each closure set and postset (i.e., ca:6, b:5, d:2). Then (see Fig.7c), find intersection of ca:6 (in level 2) and

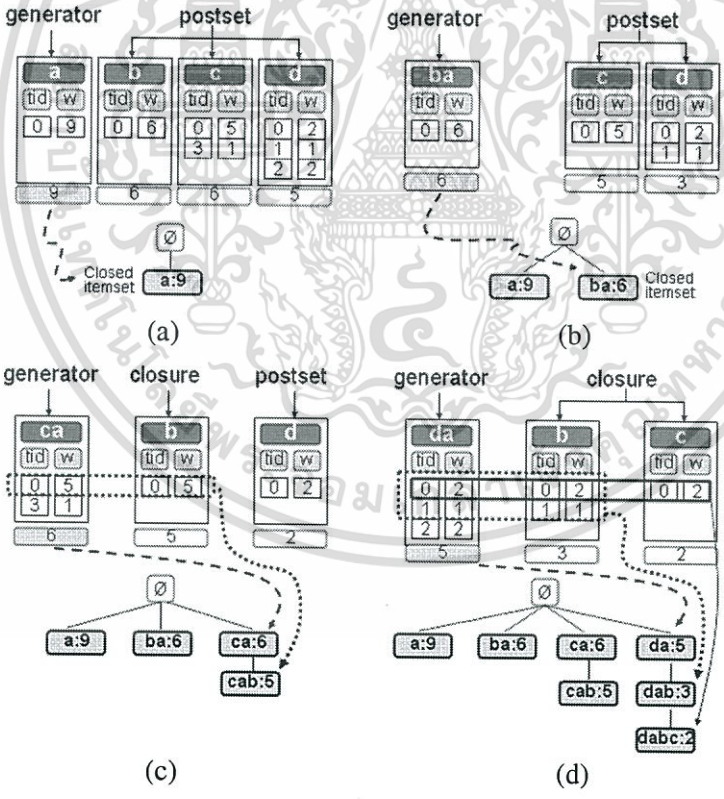


Fig.7 Our EBPA-based closed itemsets from 4 single itemset generators.

compare to *ca*:6 with their postset for duplication checking. In this case, itemset *c* is not closed (*c*, *ca* belong to the equivalence class with the same support 6), and only *ca* is called closed. Since *cb*:5 is not equal to *c*:6 then *cb* is not closed. Next, in level 3, find intersection of *cab*: 5 and compare with the postset and hence itemset *cab* is called closed. From the example illustrated in Fig.7a-d, we will find that our EBPA-CLOSED can generate closed itemsets from another prefix closed itemset from the previous (lower) level (by reusing intermediate results) and reduce a number of non-generators without storing previously enumerated itemsets.

4 Performance Evaluation

In performance evaluation, we implemented our EBPA data structure and the EBPA-CLOSED Algorithm, compared with the existing approach. Our program code was written in C language. Experiments have been performed on a Windows XP notebook PC, equipped with a 2.5 GHz Intel Core i5 and 2048MB of RAM memory. A number of experiments were investigated by using a four-data tested set of dense transactions, which are *connect*, *chess*, *pumsb** and, *pumsb* respectively. These existing datasets have been used for testing in many FCIM approaches. The *chess* dataset composes of 3196 tuples with 75 items, the *connect* dataset composes of 67437 tuples with 129 items, the *pumsb** dataset composes of 34776 tuples with 2001 items and the *pumsb* dataset composes of 49046 tuples with 2113 items.

The performance results were recorded in terms of response time (in seconds). In each experiment, evaluated results of the EBPA-CLOSED mining (ebpa) have been investigated and compared to those of the existing LCM version 3 (lcm3) with original collaboration data structure [10] and FCIM ppc-extension [11]. For each data tested set, threshold of minimum support are set varying for investigating the response time of a number of limited frequent 1-itemsets. The (response time) results of our experiments were reported in Fig.8 - Fig.11, compared between our "ebpa" and the "lcm3", the best of existing FCIM mining. In each figure, the y-axis represents the response time (of FCIM mining) with varying support thresholds (in x-axis) on four datasets (*chess*, *connect*, *pumsb** and *pumsb*).

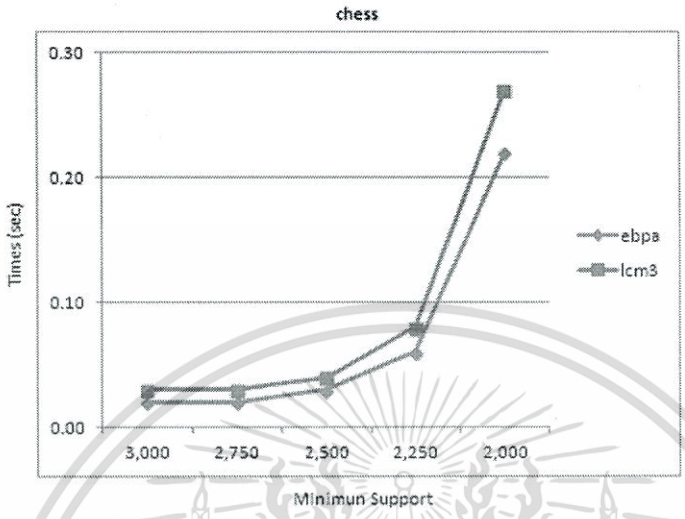


Fig.8 The comparison of our ebpa-closed and lcm3 mining of *chess* data set with different minimum support.

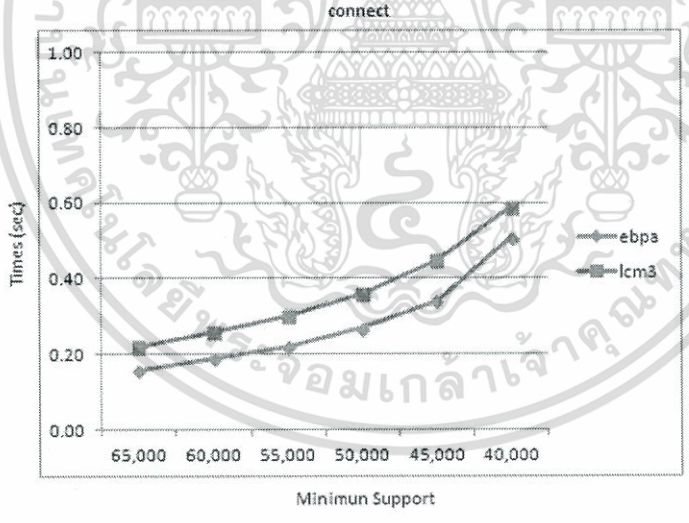


Fig.9 The comparison of our ebpa-closed and lcm3 mining of *connect* data set with different minimum support.

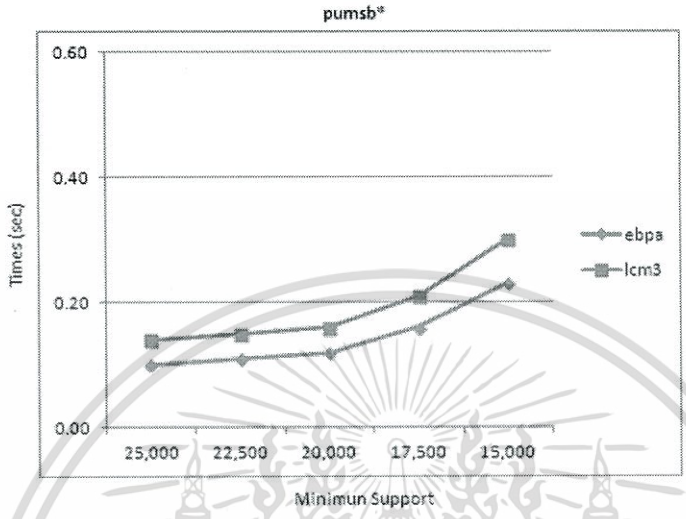


Fig.10 The comparison of our ebpa-closed and lcm3 mining of pumsb* data set with different minimum support.

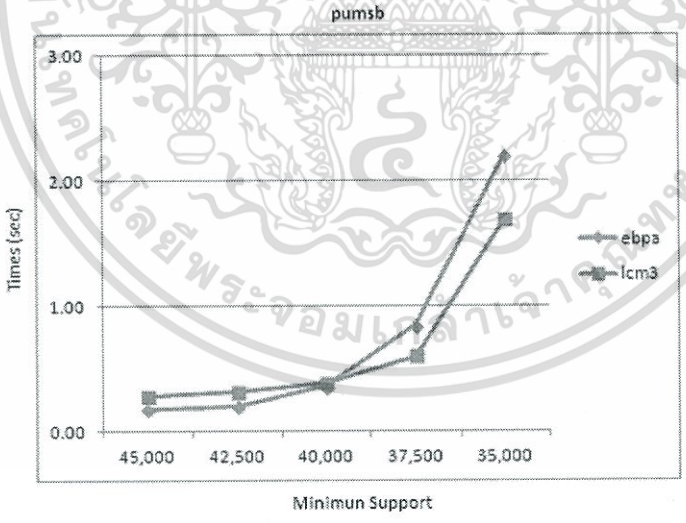


Fig.11 The comparison of our ebpa-closed and lcm3 mining of *pumsb* data set with different minimum support.

The results showed that our ebpa approach performed very well on three datasets (*chess*, *connect*, *pumsb**), which outperformed over those of the lcm3 approach. However, the percentage of improved results also depended on each type of datasets such as improved up to 22% on *chess*, 15% on *connect*, and 30% on *pumsb** respectively. In all datasets, when decreasing the minimum support values (in the dataset), the response time increased in both approaches (ebpa, lcm3) because of the less minimum support values, the more frequent 1-itemsets and more time needed to process. For three of four datasets, our ebpa outperformed over those of the lcm3. The reason is that we introduce the improved collaboration data structure that provides the efficient FCIM mining, as illustrated in their time complexity (in Section 3.2 and 3.3). Time complexity of the FCIM algorithm (i.e., time for creating the initial data plus time for processing FCIM) in the lcm3 (using the original collaboration data structure [10]) is $[O(mT) + O(mn_iT')] + O(m^2n)$, where as that of our EBPA-based FCIM algorithm is $[O(m^2n) + O(mT)] + O(m^2n)$. Cleary, the response time of constructing our (compact) prefix tree is improved over that of the existing collaboration if the unique transactions (T') are more than the number of occurrence nodes (n) because (in practice) most of transaction databases and available datasets composts of $T' > n$. However, when $n > T'$ (in some datasets i.e., *pumsb* for *min_supp* < 40000 (Fig.11)), which there exist a lot of repeated transactions (less T') and a variety of itemsets (more n), the response time of the lcm3 is less than our ebpa approach.

5 Conclusion

This paper presents an improved collaboration data structure, called the EBPA data structure, and its corresponding algorithm (the EBPA-CLOSED algorithm) for frequent closed itemset mining (FCIM). Our proposed EBPA (Efficient Bitmap Prefix-tree Array) data structure for efficient FCIM algorithm can improve both space and time with $O(1)$ parent-child access (into the prefix-tree array). The performance evaluation was performed on four available datasets, which were *pumsb*, *pumsb**, *chess*, and *connect*. Experimental results showed that our EBPA-based closed itemset mining (ebpa) can reduce response time up to 15-30% over that of the lcm3 in *pumsb**, *chess* and *connect* datasets. Finally, our future research will focus on designing an efficient parallel EBPA-CLOSED algorithm with the EBPA data structure on multi-core systems.

Acknowledgement

We would like to thank the Office of the Higher Education Commission of Thailand and Ubonratchathani University for the financial support. We are also grateful to the owner of the datasets for providing the available datasets and the LCM editors for posting the best programming code.

References

[1] N. Pasquier, Y. Bastide, R. Touil and L. Lakhal, Discovering frequent closed itemsets for association rules, Proceedings of 7th International Conference on Database Theory (ICDT 1999), LNCS, vol.1540, Springer-Verlag, Jerusalem, Israel, 1999.

[2] N. Pasquier, Y. Bastide, R. Taouil and L. Lakhal, Efficient Mining of Association Rules Using Closed Itemset Lattices, Information Systems, vol. 24, no. 1, 1999, 25-46.

[3] J. Pei, J. Han and R. Mao, CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets, Proceedings of ACM SIGMOD Int'l Workshop Data Mining and Knowledge Discovery, 2000.

[4] M.J. Zaki and C.-J. Hsiao, CHARM: An Efficient Algorithm for Closed Itemsets Mining, Proceedings of Second SIAM Int'l Conf. Data Mining, 2002.

[5] G. Grahne and J. Zhu, Efficiently Using Prefix-Trees in Mining Frequent Itemsets, Proceedings of ICDM Workshop Frequent Itemset Mining Implementations, 2003.

[6] J. Pei, J. Han and J. Wang, CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets, Proceedings of Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, 2003.

[7] T. Uno, T. Asai, Y. Uchida and H. Arimura, LCM: An Efficient Algorithm for Enumerating Frequent Closed Item Sets, Proceedings of IEEE ICDM'03 Workshop FIMI'03, 2003.

[8] T. Uno, M. Kiyomi and H. Arimura, LCM ver.2: Efficient Mining Algorithm for Frequent Closed Maximal Itemsets, Proceedings of IEEE ICDM'04 Workshop FIMI'04, 2004.

[9] C. Lucchesse, S. Orlando and R. Perego, DCI-Closed: a fast and memory efficient algorithm to mine frequent closed itemsets, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04), volume 126 of CEUR Workshop Proceedings, Brighton, UK, 2004.

[10] T. Uno, M. Kiyomi and H. Arimura, LCM ver.3: Collaboration of Array, Bitmap and Prefix Tree for Frequent Itemset Mining, Proceedings of Open Source Data Mining Workshop on Frequent Pattern Mining Implementations 2005, 2005.

[11] T. Uno, T. Asai, Y. Uchida and H. Arimura, An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases, IEEE ICDM workshop FIMI'04, Zaki & Goethals, 2004.

[12] C. Lucchese, S. Orlando and R. Perego, Fast and Memory Efficient Mining of Frequent Closed Itemsets, IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 1, 2006, 21-36.

[13] J. Sribuaban, V. Boonjing and J.Werapun, Frequent Closed Multi dimensional Multi-level pattern mining, Proceedings of the Fourth IASTED International Conference on Advances in Computer Science and Technology, Malaysia, 2008, 201-206.

[14] J.Wachiramethin and J.Werapun, BPA: A Bitmap-Prefix-tree Aarray data structure for frequent closed pattern mining, Proceedings of the Eighth International Conference on Machine Learning and Cybernetics, Baoding, China, 2009, 154-160.

Received: January, 2013

Proceedings of the Eighth International Conference on Machine Learning and Cybernetics, Baoding, 12-15 July 2009

BPA: A BITMAP-PREFIX-TREE ARRAY DATA STRUCTURE FOR FREQUENT CLOSED PATTERN MINING

JUGKARIN WACHIRAMETHIN, JEERAPORN WERAPUN

Department of Computer Science, Faculty of Science,King Mongkut's Institute of Technology, Ladkrabang (KMITL), Bangkok 10520, THAILAND
E-MAIL: s9062951@kmitl.ac.th and ksjeerap@kmitl.ac.th

Abstract:

This paper presents a new efficient data structure, called "a *BPA* (Bitmap-Prefix-tree Array)" for discovering frequent closed itemset in large transaction database. Recently, most studies have been focused on using an efficient data structure with preprocessing data for the frequent closed itemset mining. Existing prefix-tree-based approach presented the *IT-Tree* data structure in its complete preprocessing data for the efficient frequent searching but used large memory space and time consuming in the preprocessing step. Lately, another approach introduced the efficient data structure, called "a collaboration of array, bitmap, and prefix tree", to improve storage and time in preprocessing data. However, its preprocessing step was not complete and hence its frequent searching for the frequent closed itemset mining may take more time than that of the *IT-Tree*-based approach. In this paper, we propose the efficient *BPA* data structure to enhance not only computation-time and memory-space in the complete preprocessing data but also in those in the frequent searching.

Keywords:

Data mining; Closed itemset mining; Multi-dimensional multi-level pattern mining; Bitmap; Prefix tree; Array lists

1. Introduction

Frequent Itemset Mining (or *FIM*) is a major important technique in several data mining applications that is used for discovering interesting patterns within transaction database by considering frequent of itemsets. In some types of databases, the *FIM* is possible to create an exponentially large number of patterns when the minimum support threshold is low, and the transaction is very large because of containing strongly correlated items and long frequent patterns. In that case, it takes a long time for finding interesting patterns and uses a lot of memory space. Therefore, many approaches ([4], [5], [9], [10], [11]) for frequent closed itemset mining (or *CFIM*) were proposed and have been improved for solving those problems. The *CFIM* is a method for representing patterns finding from all candidate patterns by considering itemsets with the same support in equivalence class. The complete set derived from this method can reduce a number of itemsets without information loss and cover all results of the original *FIM*. However, the frequent searching of the (backtracking-based) *CFIM* is still time consuming for a large number of items.

One solution for improving the computation time of that *CFIM* is "preprocessing data", stored in a special data structure. The suitable data structure and efficient preprocessing can enhance computation-time and save more memory-space. In the different *CFIM* with preprocessing data, three basic data structures were applied: 1) bitmap matrix ([1], [2]); 2) array lists [12], [13]; and 3) prefix tree ([3], [8])). Each data structure was introduced for either dense database or spare database with different advantage and disadvantage.

In 2002, an *IT-Tree* data structure [8] was introduced in an efficient algorithm for closed itemsets mining for both dense and space databases. The *IT-Tree* uses the prefix-tree and hashing data structures in the complete preprocessing data for the *CFIM*. The frequent searching in the *IT-Tree* is $O(n)$ time, where $n = \max(n_i)$ and $n_i = i$ be a number of frequent single-items in a node at level i of the tree ($1 \le i \le n$). However, that approach may require a large memory-space for all nodes (storing in bytes) of the IT-Tree and time consuming for the complete preprocessing data.

In 2005, an efficient data structure, called a collaboration of array lists, bitmap, and prefix trees [14], was introduced with the efficient preprocessing data for the *CFIM* to save memory-space and computation-time. That data structure combined the advantages of three basic data structures for providing the efficient time and improving memory-space in all nodes (storing in bits) for mining very large dataset. However, the preprocessing data of that approach performs the partial frequent counting. Thus, the frequent searching of that approach has to include the final frequent counting and hence its frequent searching may take more time than that of the *IT-Tree*-based approach.

In real world, collected data in the transaction database

978-1-4244-3703-0/09/$25.00 ©2009 IEEE

usually contain more interesting useful information than the traditional pattern mining, such as dimension of consumers and hierarchy of products. Therefore, in our previous study [6], we proposed the frequent closed multi-dimensional multi-level pattern mining (or *CMDML*) that is a combination of frequent multi-dimensional multi-level pattern mining (or *MDML*) [7] and the *CFIM*. Such significant information can be used for analysis on knowledge discovery in database systems. That study was introduced for mining frequent patterns in real life information by including dimensional and hierarchical information. However, the *CMDML* approach will generate more patterns than the methods concerning only item information, and hence that approach requires more space to store data during mining and needs more time to compute the support of each item.

Applying the IT-Tree data structure [8] for dimensional and hierarchical information will generate more patterns and require more spaces (storing in bytes) in its prefix-tree structure but yield the efficient frequent searching. Applying the existing collaboration data structure [14] for such information, its frequent searching will take more time but require less spaces (storing in bits) than those of the prefix-tree-based approach.

In this paper, we propose a new efficient data structure, called "BPA: a Bitmap Prefix-tree *Array data structure*", for the preprocessing data and the frequent searching, applied for both the *CFIM* (frequent Closed Itemset Mining) and the *CMDML* (frequent Closed Multi-Dimensional Multi-Level pattern mining). Our proposed *BPA* data structure and its corresponding computation are efficient in computation-time and memory-space for the frequent closed itemset mining. In particular, we focus on constructing the efficient *BPA* structure by combining the prefix tree (fast searching), the array list (fast computing), and vertical bitmap matrix (space reducing) to enhance the preprocessing time and save memory-space in the preprocessing data over that of the collaboration-based approach [14], while maintain the efficient frequent searching ($O(n)$) similar to that of the prefix-tree-based approach [8]. In addition, our new data structure (*BPA*) can be applicable to existing frequent closed pattern mining algorithms, such as in CHARM [8] (using *prefix-tree*–based structure) and LCM3 [14] (using collaboration (prefix-tree, array-list, bitmap)-based structure).

The remainder of this paper is organized as follows: Section 2 presents related work. Section 3 presents our new combined data structure (*BPA*) and an efficient technique for the frequent computation for the *CFIM* and *CMDML*. Finally, conclusion and future study are discussed in Section 4.

2. Related Work

For efficient *CFIM* in existing studies, three basic data structures have been applied: 1) the vertical-bitmap [1], [2]; 2) the array-list structure [12], [13]; and 3) the prefix-tree structure [3], [8]. Figure 1(a) illustrates an example of representing a transaction dataset (of eight 1-items (a, b, c, d, e, f, g, h) and 9 transactions) and Figure 1(b)-(c)) display results after applying those three basic data structures with minimum support = 5.

The vertical bitmap structure (Figure 1(b)) provides a memory-space efficient method for the dense transaction database because this approach represents all itemsets in transaction database by using only one bit (*0* or *1*) for each item (or nT bits for all transactions (T)), where $T \leq 2^n$) and n represents a number of frequent 1-items. However, this method may not be efficient in time ($O(nT^2)$) in sparse transaction database since its corresponding bitmap matrix containing many *0*s but the process is always equal to nT^2.

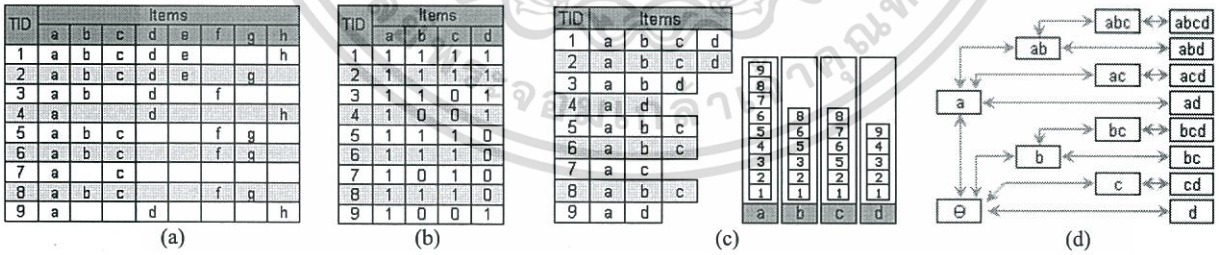


Figure 1. An example of three data structures for a transaction dataset of eight 1-items (a, b, c, d, e, f, g and h): (a) An original transaction database, (b) Vertical Bitmap Matrix represents the transaction dataset with preprocessing, (c) Array List data structure with buckets represents the transaction dataset with preprocessing, and (d) Complete Prefix tree data structure represents four 1–items (*a, b, c, d*) of the transaction dataset with preprocessing.

The array list data structure represents n frequent 1-items and T transactions by using T array lists (see Figure

1(c)) and computes the frequencies by scanning the nine lists (T=9) of four 1-items (n=4) in buckets. That array-list-based method yields an efficient computation for sparse transaction database [13], but it is still weak in quite dense transaction database [14].

For both dense and sparse databases, the *IT-Tree* [8] was introduced by using the prefix tree structure to store items of transaction in each node of the tree (see Figure 1(d)). The *IT-Tree* was constructed from the root (or top-down) step by step, according to all T transactions (one at a time). The advantage of using the prefix tree is that accessing to any node for computing or searching in the *IT-tree* with hashing (between parent and its children nodes) is efficient ($O(n)$ time). However, the preprocessing step of the prefix-tree structure for each transaction takes long time, especially in dense transaction database, because it always starts computing from the root node. For example (in Figure 1(d)), the frequent counting for itemset "*abcd*" covers the computing in all nodes ($\leq 1+2+2^2+\ldots+2^{n-1} = 2^n-1 = N$) in the tree and hence its computation time is $O(N)$ and $O(NT)$ for all T transactions, where $N = \max(N_i)$ is either small or large, according to sparse or dense transaction database ($N_i \leq 1+2+2^2+\ldots+2^{i-1} = 2^i-1$; $i = 1, 2, \ldots, n$). Moreover, that approach may require large memory-space for storing node information (in bytes) in all nodes ($\leq 2^n-1$) in the tree, where each node contains an item identifier ($\leq n$ bytes), a support (or frequency), a parent-node pointer, and n_i children-node indices ($\leq n$). Specially, in case of long patterns, it uses many bytes ($\leq 2n+2$ bytes) per pattern (or $\leq 2(n+1)2^n$ bytes) for all possible transactions ($T \leq 2^n$). Note: that prefix-tree contains only appearing transactions to save space and time in the sparse database.

The collaboration of array, bitmap and prefix-tree data structure [14] was introduced to utilize the advantage of using each of three basic data structures for making more efficient in computation-time (using "array list" and "prefix tree") and saving more memory-space (using "bitmap") for mining large transaction database, including dense and spare databases. In that approach, the complete prefix-tree (2^n-1 nodes) for n 1-items is generated first. Then scan each transaction to fill its initial frequency into its corresponding node. Each node contains a node label (in bits), a support (or frequency), a parent-node pointer, and n_i children-node mapping ($\leq n$ pointers). The node label represents items of transaction in binary (n bits) in order to save memory-space. For the efficient computation time, that approach performs the complete preprocessing data (or frequent counting) in two separate parts: 1) the frequent counting in depth for

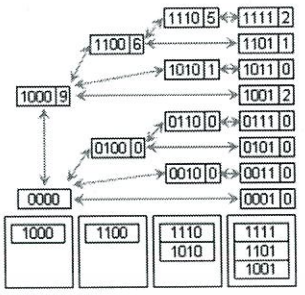


Figure 2. Collaboration of Array, Bitmap, and Prefix Tree structure represents the transaction dataset with preprocessing.

supporting the prefix relation (using the "prefix tree") plus 2) the frequent counting in each level for supporting the superset relation (using the "array lists" or "buckets").

- The frequent counting (in depth) is accumulated from all right nodes to the root (or bottom up). In order to avoid scanning some nodes many times, initial frequencies are filled in nodes first from T transactions. Then, frequencies of leaf nodes (in level L) are forwarded to accumulate in their parent nodes (in previous level) before starting the process in level L-1 and so on (to the root). For example (see Figure 2), the frequent counting in depth of nodes 1111 and 1101 are 2 and 1. Hence, that of node 1110 is 5 (3 from initial setting add 2 from 1111) and that of node 1100 is 6 (5+1 from 1110 and 1101). Therefore, time complexity for this step is $O(nT+N)$, where n=max(n_i).
- In each level of the tree, a bucket is used to store only appearing transaction(s), where m_i = 0, 1, 2, 3,..., or 2^i nodes in level i of the tree ($1 \leq i \leq n$). However, the frequent counting (in each bucket) will be computed in the frequent searching for a transaction. That process starts finding all of its subsets (by scanning all nodes in the bucket) and then computes the summation of their corresponding frequencies. Therefore, time complexity of that frequent counting (and searching) is $O(m)$ and hence $O(mT)$ for T transactions, where $m = \max(m_i)$.

However, n_i links (of 1-parent and n_i-children) in each tree-node [14] cannot be represented in bits and hence requires n_i bytes per node. Moreover, all n-bit-nodes in each bucket (m_i) are not sorted and hence the frequency counting (in each level) needs to scan all m_i nodes in the bucket for finding the superset of the generated itemset. Therefore, next in Section 3 we introduce the more efficient data structure to be the solution for those problems.

Proceedings of the Eighth International Conference on Machine Learning and Cybernetics, Baoding, 12-15 July 2009

3. The Proposed "BPA" Data Structure

In this section, we present a new efficient data structure, called "*BPA: a Bitmap Prefix tree Array data structure*", for the efficient preprocessing data and the efficient frequent searching for itemset mining.

Our new *BPA* data structure combines the prefix tree with multiple arrays (for fast computing and searching) and bitmap (for space reducing). In this efficient structure, we uses multiple arrays to emulate the complete prefix-tree in order to achieve the fast access to index of any node directly in such a prefix-tree array by computing the "addressing index", corresponding to its n_i-bit-node label (introduced in Section 3.2). Therefore, n_i links (of a parent and its n_i-children) do not required in our *BPA* data structure and hence save more memory-space n_i bytes per node over those of the *IT-Tree* [8] and the collaboration [14]. Each node of our prefix-tree arrays contains a node label (n_i bits), a support (1 byte), and a parent node pointer (1 byte).

3.1 The Construction of the *BPA* Data Structure

The construction of the *BPA* data structure can be divided into three main steps, illustrated as follows:

Step 1	**Transform Data into Bitmap** Find the frequent of 1-items by scanning the transaction dataset to find the set of frequent items, sorting by descending order from the support of itemsets and convert to binary bitmap (itemsets>=*min_supp* only).
Step 2	**Construct Complete Prefix-Tree Array Layout** Generate complete prefix-tree array layout of ($N = 2^n$-1 nodes) from n frequent 1-items and set initial frequency = 0 into every node.
Step 3	**Counting the Support (or Frequent Counting), based on the "addressing index"** Scan a bitmap transaction dataset to store all itemsets in our data structure and compute the frequent counting (in both depth and level).

To complete the construction of our data structure (*BPA*), we illustrate this construction (Step 1-3) starting from a transaction database by using a simple example,

Item	Support	Transaction
a	9	1,2,3,4,5,6,7,8,9
b	6	1,2,3,5,6,8
c	6	1,2,5,6,7,8
d	5	1,2,3,4,9
g	4	2,5,6,8
f	4	3,5,6,8
h	3	1,4,9
e	2	1,2

(a)

Bitmap	weight
1111	2
1101	1
1001	2
111	3
101	1

(b)

Figure 3. (a) Descending reorder transaction database and (b) A bitmap transaction database with weight.

shown in Figure 3 (with all 1-items) and Figure 4 (with four frequent 1-items (*a, b, c, d*), $n = 4$).

First (in Step 1), find all 1-items (*a, b, c, d, e, f, g, h*) (see Figure 1(a)) by scanning dataset and create item-index table, and reorder dataset for frequent 1-items (items ≥ *min_sup*) (see Figure 3(a)). Suppose minimum support threshold is 5, and hence frequent 1-items (n=4) are *a(9), b(6), c(6), d(5)*. Then convert frequent itemsets to bitmap, and sum frequency of the same transaction (see Figure 3(b)). Next (in Step 2), generate 4 index tables (T1, T01, T001, T0001) and 4 complete prefix-tree arrays (a1, a01, a001, a0001) with n_i-bitmap-data in each node (see Figure 4(a)) from frequent 1-items (n) with setting initial frequency = 0 .

Lastly (in Step 3), first scan bitmap transaction dataset (see Figure 3(b)) to fill itemsets and initial frequent counting (Step 3.1) in corresponding prefix-tree-array nodes (see Figure 4(b)). Then perform the complete preprocessing data (or frequent counting) in Step 3.2-3.3: 1) the frequent counting in depth (using "prefix-tree array") for the prefix relation (see Figure 4(c)-(e); and 2) the frequent counting in (each) level (using "buckets") for the superset relation (see Figure 4(f)-(k)).

- First (in Step 3.1: "Initial frequent counting"), scan the bitmap transaction database and add initial frequency of each itemset into its array node (see Figure 4(b)). For example, the result of the transaction *1111* is performed as follows: compute "addressing index" of 1111=0, and add frequency (=2) into the prefix-tree array a0001[0] and store index (=0) into the corresponding index table T0001. Similarly for *1101*, add frequency (=1) into a0001[1] and store index (=1) into T0001. For *1001*, add frequency (=2) into a0001[4] and store index (=3) into T0001. For *111*, add frequency (=3) into a001[0] and store index (=0) into T001, and so on. Note: Time complexity of this step for each transaction is *O(n)* and *O(nT)* for all transactions (*T*) since node's addressing index is computed in n_i iterations (see in Section 3.2).
- Next (in Step 3.2: "Frequent counting in depth"), add (or accumulate) the frequency of parent node from its children node arrays (a0001, a001, a01) to the root (a1). For example, a0001 has three elements (at 0, 1, and 3) whose frequency > 0. For each element of a0001, its frequency is forwarded and added (=>) into its parent, and hence *1111* => a001[0], *1101* => a01[0], and *1001* => a1[0] (see Figure 4(c)). Similarly for each element of a001, *111* => a01[0], and *101* => a1[0] (see Figure 4(d)). Lastly for a01, *11* => a1[0] (see Figure 4(e)). Note: Time complexity of this step for all nodes ($m_1 + m_2 + \ldots + m_n \le N = 2^n$-1) is *O(N)*, where $0 \le m_i \le 2^i$ (size of array at level i of the prefix-tree arrays ($1 \le i \le n$).

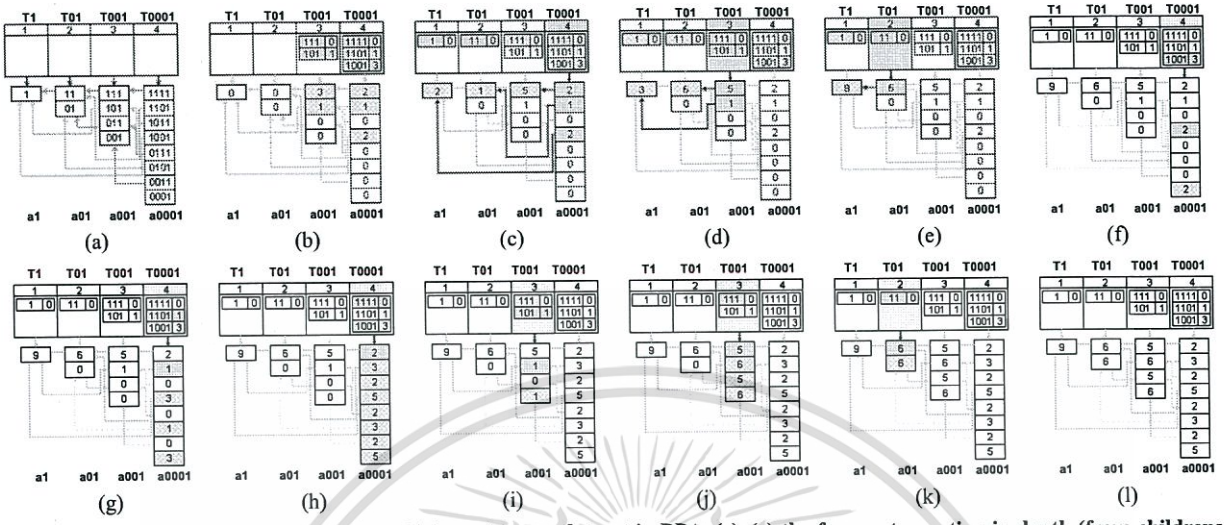


Figure 4. (a) BPA data structure outline, (b) input bitmap dataset in BPA, (c)–(e) the frequent counting in depth (from childrens node to parent node), (f)-(k) the frequent counting in level (spread the frequent step), and (l) the complete BPA data structure.

- Finally (in Step 3.3: "Frequent counting in (each) level"), add the frequency of subset node in the same array or the same level of the prefix-tree. For example, the array a0001 has 3 elements (*1111, 1101, 1001*) at indices (0, 1, 3). First, the subset node of a0001[4] (=*1001*) is node *0001*. So, add the support of *1001* (=2) into node *0001* (0+2=2) (see Figure 4(f)). Next, add that (=1) of a0001[1] (=*1101*) into its subset nodes *1001*:3, *0101*:1 and *0001*:3 (see Figure 4(g)). Then, add that (=2) of a0001[0] (=*1111*) into nodes *1101, 1011, 1001, 0111, 0101, 0011, 0001* (see Figure 4(h)). For the array a001 with 2 elements, add that (=1) of a001[1] (=*101*) into node *001* and add that (=5) of a001[0] (=*111*) into nodes *101, 011, 001* (see Figure 4(i)-(j)). For the array a01 with 1 element, add that (=6) of a01[0] (=*11*) into node *01* (see Figure 4(k)). Time complexity of the frequent counting for each node is $O(m)$ and $O(mT)$ for T transactions (or appearing nodes), where $m = \max(m_i)$.

Note: In order to reduce the frequent counting time (in each level in Step 3.3) of our *BPA* approach, we need the sorted results in each bucket (or index table T0…01) before starting Step 3.3. The sorting can avoid wasting time in scanning some nodes in the same bucket ($O(m_i^2)$) by sorting ($O(m_i \log_2 m_i)$), $i = 1, 2, 3, \ldots, n$. Then, to find all subsets of each node in the same bucket, we can scan from the first index up to the destination index only (with 1 index for best case, $(m_i+1)/2$ indices for average case, and m_i indices for worst case). Therefore, in practical executing this step can be performed faster than those ($O(m_i^2)$) of the collaboration [14], which each of m_i nodes in the same bucket needs to scan through all m_i nodes in all (best, average, worst) cases.

3.2 An Efficient Frequent Counting and Frequent Searching Method for Closed Itemsets

Closed itemset is closure of others that have the same support in the same equivalence class. Thus, the frequency counting for the support value is the main important part in closed itemset mining. Finally, the support (of frequent) searching performance of each data structure applied is different, depending on its data structure and support searching technique. For closed itemset mining, the intersection, presented in [1] and [2], is one popular technique that uses to perform closures, frequent counts and duplicate detections.

The frequency counting is the heaviest part of frequent itemset mining and frequent closed itemset mining. In particular, some databases contain many transactions and long patterns that take long time. Applied our *BPA* data structure for the frequent counting (see Section 3.1) in the preprocessing step can provide the efficient frequent searching in $O(n)$ for each transaction and hence $O(nT)$ for all T transactions, where $n = \max(n_i)$ and $n_i = i$ be a number of frequent single-items in a node at level i of the tree ($1 \le i \le n$). The derived time complexity is illustrated as follows:

After preprocessing data (complete frequent counting), our *BPA* data structure contains the support of each itemset within each node in the prefix-tree arrays (a1, a01, a001, …, a000…01). Then to find a support of each itemset (for frequent closed itemset mining), we can access directly to any node (to read its support) by computing its "addressing index" in the corresponding prefix-tree array.

978-1-4244-3703-0/09/$25.00 ©2009 IEEE

According to the prefix-bitmap node ($b_0b_1b_2\ldots b_j\ldots b_{n_i-1}$) in our prefix-tree arrays (where $n_i = i$ and $1 \le i \le n$), the node's addressing index can be computed, as follows:

$$\text{index} = \sum_{j=1}^{n_i} \text{sum}_j \quad \text{where} \quad \text{sum}_j = \begin{cases} 2^{(n_i-j)-1}, & \text{if } b_{j-1} = 0, \\ 0, & \text{otherwise.} \end{cases}$$

This equation convert the prefix-bitmap node to its "addressing index" in the corresponding prefix-tree array (a0…01) by accumulating the weight ($=2^{(n_i-j)-1}$) of all 0 bits ($b_j = 0$), where j represents the position of bit j ($j = 1, 2, \ldots, n_i$). For example, the support searching of itemset *"abd"* (or *1101*) from the equation ($n_4 = 4, j = 3$) can be performed by computing the itemset's index = 1 (or $2^{(4-3)-1}$) and hence read its support from a0001[1] (see Figure 4(l)). Thus, time complexity of the support searching is $O(n)$, $n = \max(n_i)$.

Compared time and space among the *IT-Tree* [8], the collaboration [14], and our *BPA*, time complexity of the preprocessing data are $O(NT)$, $O((n+m)T+N)$, and $O((n+m)T+N)$ respectively, where n=max(n_i), m=max(m_i), $m_i \le 2^i$, and N, $T \le 2^n$. The required space for each of N nodes are $\le 2(n+1)$ bytes of the *IT-Tree* and $\le$ ($2n$ bits + (n+2) bytes) of the collaboration and $\le$ (n bits + 2 bytes) of our *BPA*. Therefore, computing time in the preprocessing of our *BPA* is comparable to that of the *collaboration* and memory space is improved over those approaches. In addition, the searching time for each itemset of the *IT-Tree* and our *BPA* is $O(n)$, while that of the collaboration is $O(m)$, where $0 \le m_i \le 2^i$ and $i = 1, 2, 3, \ldots, n$-1.

4. Conclusions

This paper presents the combination data structure (*BPA*) of prefix tree, bitmap and array list data structures for the *CFIM* and the *CMDML*. Our study focuses on improving both computation-time and memory-space in the preprocessing data (for the frequent counting) and the frequent searching (for the closed itemset mining). Our new *BPA* data structure can reduce the frequency counting time and space over those of the existing approaches, while maintaining the efficient frequent search. Finally, in our future study, we will modify our *BPA* data structure for designing an efficient parallel algorithm for the *CFIM* and the *CMDML*.

References

[1] C. Lucchesse, S. Orlando, and R. Perego, "DCI-Closed: a fast and memory efficient algorithm to mine frequent closed itemsets", Proc. the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04), volume 126 of CEUR Workshop Proceedings, Brighton, UK, Nov. 2004.

[2] C. Lucchese, S. Orlando, and R. Perego, "Fast and Memory Efficient Mining of Frequent Closed Itemsets", IEEE Transaction on Knowledge and Data Engineering, Vol. 18, No. 1, pp. 21-36, Jan. 2006.

[3] G. Grahne and J. Zhu, "Efficiently Using Prefix-Trees in Mining Frequent Itemsets," Proc. ICDM 2003 Workshop Frequent Itemset Mining Implementations, Dec. 2003.

[4] J. Pei, J. Han, and R. Mao, "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets," In Proceedings of ACM SIGMOD Int'l Workshop Data Mining and Knowledge Discovery, May. 2000.

[5] J. Pei, J. Han, and J. Wang, "CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets," Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, Aug. 2003.

[6] J. Sribuaban, V. Boonjing, and J. Werapun, "Frequent Closed Multi-dimensional Multi-level pattern mining," Proc. the Fourth IASTED International Conference on Advances in Computer Science and Technology, Malaysia, pp. 201-206, Apr. 2008.

[7] M. Runying, "Adaptive-FP: An Efficient and Effective Method for Multi-Level Multi-Dimensional Frequent Pattern Mining", Apr. 2001.

[8] M.J. Zaki and C.-J. Hsiao, "CHARM: An Efficient Algorithm for Closed Itemsets Mining," Proc. Second SIAM Int'l Conf. Data Mining, Apr. 2002.

[9] M.J. Zaki, "Mining Non-Redundant Association Rules," Proc. Data Mining and Knowledge Discovery, vol. 9, no. 3, pp. 223-248, 2004.

[10] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Efficient Mining of Association Rules Using Closed Itemset Lattices," Proc. Information Systems, vol. 24, no. 1, pp. 25-46, 1999.

[11] N. Pasquier, Y. Bastide, R. Touil, and L. Lakhal, "Discovering frequent closed itemsets for association rules", Proc. 7th International Conference on Database Theory (ICDT 1999), LNCS, v.1540, Springer-Verlag, Jerusalem, Israel, pp. 398–416, Jan. 1999.

[12] T. Uno, T. Asai, Y. Uchida, H. Arimura, "LCM: An Efficient Algorithm for Enumerating Frequent Closed Item Sets", Proc. IEEE ICDM'03 Workshop FIMI'03, 2003.

[13] T. Uno, M. Kiyomi, H. Arimura, "LCM ver. 2: Efficient Mining Algorithm for Frequent Closed Maximal Itemsets", Proc. IEEE ICDM'04 Workshop FIMI'04, 2004.

[14] T. Uno, M. Kiyomi, H. Arimura, "LCM ver.3: Collaboration of Array, Bitmap and Prefix Tree for Frequent Itemset Mining", Proc. Open Source Data Mining Workshop on Frequent Pattern Mining Implementations 2005. Aug. 2005.

Proceedings of the Fourth IASTED International Conference
Advances in Computer Science and Technology (ACST 2008)
April 2-4, 2008 Langkawi, Malaysia
ISBN CD: 978-0-88986-730-7

FREQUENT CLOSED MULTI-DIMENSIONAL MULTI-LEVEL PATTERN MINING

Jugkarin Sribuaban, Veera Boonjing and Jeeraporn Werapun
Department of Mathematics and Computer Science, Faculty of Science,
King Mongkut's Institute of Technology at Ladkrabang (KMITL), Ladkrabang, Bangkok 10520, THAILAND
E-mail: S9062951@kmitl.ac.th, kbveera@kmitl.ac.th, ksjeerap@kmitl.ac.th

ABSTRACT
Recently, several efficient frequent closed itemset mining methods have been proposed. Those methods are only able to mine relationship among item information. However, real life colleted data in transaction database usually contain many interesting useful information not only item information but also dimension and hierarchal information. Such information can be used for analysis on knowledge discovery in database system. In this paper, we propose a new method, called frequent closed multi-dimensional multi-level pattern mining, which is suitable for mining frequent patterns in real life information. In additions, we show that 1) our completed frequent closed multi-dimensional multi-level patterns are smaller than the number of multi-dimensional multi-level frequent patterns and 2) our closed multi-dimensional multi-level patterns represent all patterns of multi-dimensional multi-level in equivalence class with the same support.

KEY WORDS
data mining , multi-dimensional mult-ilevel pattern mining, closed itemset mining

1. Introduction

Nowadays, computer technologies become fundamental tools for storing data that quickly growth every day. Data mining is a popular technology used to discover the knowledge from database. Association rule in data mining is a technique used for discovering itemsets from transaction database by considering frequent of items. Frequent itemsets mining is a main important technique in data mining for mined frequent itemsets from transaction database [1]. However, the problem of that method is the possibility in creating an exponentially large number of patterns when the minimal support threshold is low, and the transaction is very large because of containing strongly correlated items and long frequent patterns.

Therefore, many efficient methods for frequent closed itemsets mining have been proposed for solving that problem [2], [3], [4], [5], [10], [11], [12].

Those methods only consider the item information for mining frequent closed itemsets. However, real life colleted data in transaction database usually contain many interesting useful information not only item information but also dimension and hierarchal information. The more efficient method for mining frequent itemsets in real life information showed include dimension and hierarchal information, because it can produce patterns with more useful knowledge. However, the method mined frequent itemset by considering the multi-dimensional multi-level information will generate more patterns than those of the method mined frequent itemset by concerning only item information. In order to reduce the result of multi-dimensional multi-level patterns mining we need to use closed itemset mining.

This paper focuses on a combining of multi-dimensional multi-level pattern mining and closed itemset mining. Our new efficient method can reduce a number of multi-dimensional multi-level patterns. Moreover, the complete set of closed multi-dimensional multi-level patterns (or *CMDML*) derived from our new method covers all result of multi-dimensional multi-level patterns (or *MDML*).

This paper is organized as follows: section 2 illustrates basic definitions and properties applied in itemsets mining. Section 3 presents our new method called *CMDML* and its correctness. Finally, shows the conclusion and future study.

2. Basic definitions and Properties

Let a schema $(TID, D_1,D_2,...,D_m ,\{L_1, L_2,..., L_n\})$ be a multi-dimensional multi-level transaction database; where *TID* is a transaction identifier, $D_1,D_2,...,D_m$ are multi-dimensional information order 1 to m dimension and $\{L_1,L_2...,L_n\}$ is set of hierarchy information order 1 to n hierarchy. Let * be any value which belong to any domain of $D_1, D_2,...,D_m$ or $\{L_1, L_2,...,L_n\}$. A multi-dimensional multi-level pattern takes the form of $(d_1, d_2, ..., d_m ,\{l_1,l_2,...,l_n\})$, where $d_i \in (D_i \cup \{*\})$, $l_i \in (L_j \cup \{*\})$ for $(1 \le i \le m)$ and $(1 \le j \le n)$.

The definitions of frequent multi-dimensional multi-level patterns mining are illustrated as follows:

Definition 1: Let $P = (d_1,d_2,...,d_m,\{l_1,l_2,...,l_n\})$ be a multi-dimensional multi-level patterns of transaction database. The number of transaction in the database matching a multi-dimensional multi-level pattern P is called the support of P, denoted as $supp(P)$. Given a minimum support threshold min_supp, an pattern P is called a frequent multi-dimensional multi-level pattern if and only if $supp(P) \geq min\_supp$.

Table 1: Example multi-dimensional multi-level database

T ID	Age	Income	List of Item
001	>35	High	(Mobile, Nokia Mobile, Nokia 2630); (Memory Stick, Nokia Memory Stick, 1024 Mb Memory Stick)
002	>35	High	(Mobile, Nokia Mobile, Nokia 2630); (Battery, Battery Nokia, Nokia Li ION Battery); (Memory Stick, Nokia Memory Stick, 1024 Mb Memory Stick)
003	<35	Medium	(Mobile, Sony-Ericsson Mobile, Sony-Ericsson P910); (Memory Stick, Sony Memory Stick, 256Mb Memory Stick)
004	<35	Medium	(Mobile, Sony-Ericsson Mobile, Sony-Ericsson P910); (Memory Stick, Sony Memory Stick, 256Mb Memory Stick)
005	<35	Low	(Mobile, Nokia Mobile, Nokia 2630); (Memory Stick, Nokia Memory Stick, 512 Mb Memory Stick)
006	<35	Low	(Mobile, Nokia Mobile, Nokia 2630)

The example multi-dimensional multi-level database showed in Table 1 are real life data stored in transaction database that store dimension information such as age and income of customer and hierarchal information such as type of product, brand of product and model of product.

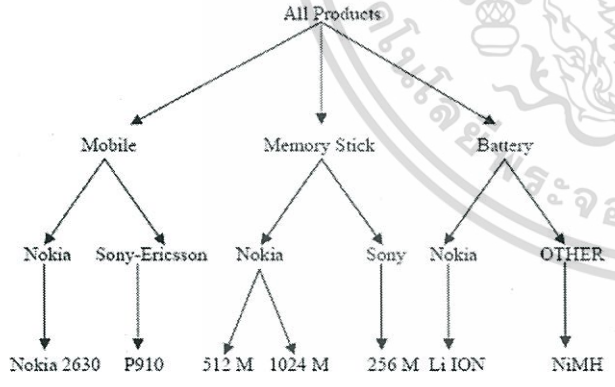


Fig. 1 An example hierarchy

The corresponding tree structure of hierarchal information from Table 1 is depicted in Fig. 1. Then, Table 2 shows the encoded transaction databases form Table 1 that is easy to process.

Table 2: Encoded transaction database

T ID	Age	Income	Items
T1	B	E	{111, 311}
T2	B	E	{111, 211, 311}
T3	A	D	{121, 321}
T4	A	D	{121, 321}
T5	A	C	{111, 312}
T6	A	C	{111}

Example 1: Form encoded transaction database in Table 2, the character *A* and *B* in age dimension represent age lower than 35 and age higher than 35, respectively, and the character *C*, *D* and *E* in income dimension represent low income, medium income and high income, respectively. Suppose minimum support threshold is 2. Multi-dimensional multi-level patterns *(B,E,{1**, 3**})*, *(A,D,{1**,3**})*, *(A,C,{1**})* are called the frequent multi-dimensional multi-level patterns, because they occurred 2 times in the example multi-dimensional multi-level database and passes the minimum support threshold.

Definition 2: Given multi-level information x, $P(x)$ is a set of all non-empty sub multi-levels information of x.

Example 2: From table 2, multi-level information *{321}* has seven sub multi-levels that they are *{3}*, *{2}*, *{1}*, *{3, 2}*, *{3, 1}*, *{2, 1}* and *{3, 2, 1}*.

Definition 3: Given multi-dimensional information y, $P(y)$ is a set of all non-empty sub multi-dimensionals of y.

Example 3: From table 2, multi-dimensional information *(B, E)* has tree sub multi-dimensionals that they are *(B)*, *(E)* and *(B, E)*.

Definition 4: Given a multi-dimensional multi-level pattern $\alpha = (y, \{x\})$, $P(\alpha)$ is a set of cross product of $P(y)$ and $P(x)$.

Example 4: From Table 2, a multi-dimensional multi-level pattern *(B, E, {1**, 3**})* has nine sub multi-dimensional multi-level patterns that they are *(B, {1**})*, *(E, {1**})*, *(B, E, {1**})*, *(B, {3**})*, *(E, {3**})*, *(B, E, {3**})*, *(B, {1**, 3**})*, *(E, {1**, 3**})* and *(B, E, {1**, 3**})*.

Definition 5: A *MDML* pattern is a *CMDML* pattern if there can be represented all *MDML* patterns that belong to the same equivalence class with the same support.

Example 5: From Table 1, suppose minimum support is 2. Set of *CMDML* patterns are *(A, D, {1**, 3**}):2, (B, E, {1**, 3**}):2, (A, C, {1**}):2, (A, {1**}):4, (A, {3**}):3, ({1**, 3**}):5, ({1**}):6, (A, D, {12*, 32*}):2, (B, E, {11*, 31*}):2, (A, C, {11*}):2, ({11*}):4, ({31*}):3, (A):4, (A, D,{121,321}):2, (B,E,{111,311}):2, (A, C, {111}):2, (A):4* and *({111}):4*.

Next, we redefine some properties of itemset have been used to find closed itemsets in [6], [7], [8] for finding frequent closed multi-dimensional multi-level pattern as follows:

Property 1: The support of *MDML* pattern in level 1 (same support in every level) is lower than the minimum support than the support of *MDML* pattern in the next level is lower than the minimum support too.

Example 6: From Table 2, the support of a *MDML* pattern *(B, E, {1**, 2**,3**})* in level 1 is not frequent *MDML* pattern. Therefore, the support of *MDML* pattern *(B, E, {11*, 21*,31*})* in level 2 and *MDML* pattern *(B, E, {111,211,311})* in level 3 are not *MDML* frequent pattern because their supports are lower than minimum support.

Property 2: Two itemsets belong to the same equivalence class they have the same closure.

Example 7: From Table 2, pattern *(A, C, {111}):2* is closure and closed itemset of *(A, C):2, (A, {111}):2, (C, 111):2* and *(C):2*.

Property 3: All sub multi-dimensional multi-level pattern of a *MDML* pattern are *MDML* pattern.

Example 8: From Table 2, all sub multi-dimensional multi-level pattern of a multi-dimensional multi-level pattern *(B, E, {1**, 3**}):2* consist of *(B, E, {11*, 31*}):2* and *(B, E, {111,311}:2)* that the supports of these itemsets are not lower than minimum support, so they are multi-dimensional multi-level pattern.

Property 4: The support of a *MDML* pattern α is equal to the support of the smallest *CMDML* pattern containing α.

Example 9: From Table 2, the support of a *MDML* pattern *(B, E, {1**, 3**})* is equal to the support of a *CMDML* pattern *(B, E, (111, 311)*, which is the smallest *CMDML* pattern containing *(B, E, {1**, 3**})*.

3. The method of closed multi-dimensional multi-level itemsets mining

This section, we presented our new method called closed multi-dimensional multi-level pattern mining or *CMDML*.

3.1 Multi-dimensional frequent pattern mining

The real life transaction database usually contains more than one dimension [9]. Therefore, we may be included dimension information into the mining process. From Table1, they have two dimensions beside the transaction identity dimension.

Example 10: From Table 1, we can be generated many rules by using multi-dimensional information.

Age(x, ">35") ∧ Income(x, "high") =>
Bye(x, "Nokia 2630") ∧ Bye(x, "1024 Mb Memory Stick")

Income(x, "high") ∧ Bye(x, "Nokia 2630") =>
Bye(x, "1024 Mb Memory Stick")

We can use dimension information for generating useful rules for using in the knowledge discovery in database system.

3.2 Multi-level frequent pattern mining

Hierarchal information tells us more useful information such as a type of product, a group of product or a class of product that there were colleted in real life transaction database. See an Example 10.

Example 11: From Table 1, we explore multi-level frequent pattern mining by using the example multi-dimensional multi-level database. Suppose the minimum support threshold is 2.

Step 1: Find frequent itemsets at level 1. (*min_supp=2*)
Level-1 *min_supp = 2* Level-1 Frequent 1-itemset
Level-1 candidate 1-itemset

Itemset	Support
{1**}	6
{2**}	1
{3**}	5

Itemset	Support
{1**}	6
{3**}	5

Level-1 candidate 2-itemset Level-1 Frequent 2-itemset

Itemset	Support
{1**, 2**}	1
{1**, 3**}	5
{2**, 3**}	1

Itemset	Support
{1**, 3**}	5

Step 2: Find frequent itemsets at level 2. (min_supp=2)
Level-2 min_supp = 2 Level-2 Frequent 1-itemset
Level-2 candidate 1-itemset

Itemset	Support
{11*}	4
{12*}	2
{21*}	1
{31*}	3
{32*}	2

Itemset	Support
{11*}	4
{12*}	2
{31*}	3
{32*}	2

Level-2 candidate 2-itemset

Itemset	Support
{11*, 21*}	1
{11*, 31*}	3
{12*, 32*}	2
{21*, 31*}	1

Level-2 Frequent 2-itemset

Itemset	Support
{11*, 31*}	3
{12*, 32*}	2

Step 3: Find frequent itemsets at level 3. (min_supp=2)

Level-3 min_supp = 2

Level-3 candidate 1-itemset

Itemset	Support
{111}	4
{121}	2
{211}	1
{311}	2
{312}	1
{321}	2

Level-3 Frequent 1-itemset

Itemset	Support
{111}	4
{121}	2
{311}	2
{321}	2

Level-3 candidate 2-itemset

Itemset	Support
{111, 211}	1
{111, 311}	2
{111, 312}	1
{121, 321}	2
{211, 311}	1

Level-3 Frequent 2-itemset

Itemset	Support
{111, 311}	2
{121, 321}	2

Therefore, the multi-level frequent itemsets found are:

Level-1: *(1**:6), (3**:5), (1**, 3**:5);*

Level-2: *(11*:4), (12*:2), (31*:3), (32*:2), (11*, 31*:3), (12*, 32*:2);*

Level-3: *(111:4), (121:2), (311:2), (321:2), (111, 311:2), (121, 321:2);.*

3.3 Closed multi-dimensional multi-level frequent pattern mining

Our new method is a combining of multi-dimensional multi-level pattern mining and closed itemset mining that consist of two major steps:

In the first step, we scan database once to get counting of every single item and every single dimension. The frequent 1-items or frequent 1-dimensions are those whose counting passes their corresponding minimum support threshold.

Example 12: From Table 1, we get the count of every single item and every single dimension from the example multi-dimensional multi-level database that shows in Table 3.

Table 3 Item and dimension counts (support=2)

Item and Dimension	Encode	Count
Age < 35	A	4
Age > 35	B	2
Income low	C	2
Income medium	D	2
Income high	E	2
Mobile	1	6
Memory Stick	3	5
Nokia Mobile	11	4
Sony-Ericsson Mobile	12	2
Nokia Memory Stick	31	3
Sony Memory Stick	32	2
Nokia 2630	111	4
Sony-Ericsson P910	121	2
Nokia 1024 Mb Memory Stick	311	2
Sony 256 Mb Memory Stick	321	2

In the second step, Computing Closures: To compute the closure of a candidate set of single multi-dimensional multi-level itemset (or C-MDML) derived form first step in every level and eliminate redundant itemsets from the result.

$C\text{-}MDML_1 = (MD_1, \{ML_1\})$

$C\text{-}MDML_2 = (MD_1, \{ML_1, ML_2\})$

…

…

$C\text{-}MDML_n = (MD_1, MD_2, \ldots, MD_d, \{ML_1, ML_2, \ldots, ML_l\})$

Given itemset x, $y \in C\text{-}MDML$, if $y \subseteq x$ and supp(y) = supp(x), then y is redundant itemset of x and can be eliminated. Frequent closed multi-dimensional multi-level pattern can appear in one of the three forms:

1. $(d_1, d_2, \ldots, d_m)$;
2. $(l_1, l_2, \ldots, l_n)$;
3. $(d_1, d_2, \ldots, d_m, \{l_1, l_2, \ldots, l_n\})$.

Example 13: Form Table 2, the frequent closed multi-dimensional multi-level itemsets found are:

Level-1: *(A, D,{1,3}):2, (B,E,{1,3}):2, (A,C,{1}):2, (A,{1}):4, (A, {3}):3, ({1,3}):5, ({1}):6;*

Level-2: *(A,D,{12,32}):2, (B,E,{11,31}):2, (A,C,{11}):2, ({11}):4, ({31}):3,(A):4;*

Level-3: *(A,D,{121,321}):2, (B,E,{111,311}):2, (A,C,{111}):2, (A):4, ({111}):4.*

Table 4 illustrates the evaluation results of our proposed method (frequent closed multi-dimensional multi-level pattern mining), compared with existing methods (such as Frequent Itemset Mining, Frequent Closed Itemset Mining, Frequent Multi-Dimensional Itemset Mining, Frequent Multi-level Itemset Mining, Frequent Multi-Dimensional Multi-Level Itemset Mining) by using the example with the minimum support = 2.

Table 4 the result of each method (support=2)

Method	Frequent Itemsets	Number of Itemsets
Frequent Itemset Mining	({111}):4, (121):2, ({311}):2, ({321}):2, ({111,311}):2, ({121,321}):2	6
Frequent Closed Itemset Mining	({111,311}):2, ({121,321}):2, ({111}):4	3
Frequent Multi-Dimensional Itemset Mining	(A):4, (B):2, (C):2, (D):2, (E):2, ({111}):4, ({121}):2, ({311}):2, ({321}):2, (A,C):2, (A,D):2, (A,{111}):2, (A,{121}):2, (A,{321}):2, (B,E):2, (B,{111}):2, (B,{311}):2, (C,{111}):2, (D,{121}):2, (D,{321}):2, (E,{111}):2, (E,{311}):2, ({111,311}):2, ({121,312}):2, (A,C,{111}):2, (A,D,{121}):2, (A,D,{321}):2, (B,E,{111}):2, (B,E,{311}):2, (B,E,{311}):2, (A,D,{121,321}):2, (B,E,{111,311}):2	31
Frequent Multi-level Itemset Mining	**Level 1 :** ({1**}):6, ({3**}):5, ({1**, 3**}):5 **Level 2 :** ({11*}):4, ({12*}):2, ({31*}):3, ({32*}):2, ({11*,31*}):3, ({12*,32*}):2 **Level 3 :** ({111}):4, ({121}):2, ({311}):2, ({321}):2, ({111, 311}):2, ({121, 321}):2	15
Frequent Multi-Dimensional Multi-Level Itemset Mining	**Level 1 :** (A):4, (B):2, (C):2, (D):2, (E):2, ({1**}):6, ({3**}):5, (A,C):2, (A,D):2, (A,{1**}):4, (A,{3**}):3, (B,E):2, (B,{1**}):2, (B,{3**}):2, (C,{1**}):2, (D,{1**}):2, (D,{3**}):2, (E,{1**}):2, (E,{3**}):2, ({1**,3**}):5, (A,C,{1**}):2, (A,D,{1**}):2, (A,D,{3**}):2, (A,{1**,3**}):2, (B,E,{1**}):2, (B,E,{3**}):2, (B,{1**,3**}):2, (D,{1**,3**}):2, (E,{1**,3**}):2, (A,D,{1**,3**}):2, (B,E,{1**,3**}):2 **Level 2 :** (A):4, (B):2, (C):2, (D):2, (E):2, ({11*}):4, ({12*}):2, ({31*}):3, ({32*}):2, (A,C):2, (A,D):2, (A,{11*}):2, (A,{12*}):2, (A,{32*}):2, (B,E):2, (B,{11*}):2, (B,{31*}):2, (C,{11*}):2, (D,{12*}):2, (D,{32*}):2, (E,{11*}):2, (E,{31*}):2, ({11*,31*}):3, ({12*,32*}):2, (A,C,{11*}):2, (A,D,{12*}):2, (A,D,{32*}):2, (A,{12*,32*}):2, (B,E,{11*}):2, (B,E,{31*}):2, (D,{12*,32*}):2, (E,{11*,31*}):2, (B,{11*,31*}):2, (A,D,{12*,32*}):2, (B,E,{11*,31*}):2 **Level 3 :** (A):4, (B):2, (C):2, (D):2, (E):2, ({111}):4, ({121}):2, ({311}):2, ({321}):2, (A,C):2, (A,D):2, (A,{111}):2, (A,{121}):2, (A,{321}):2, (B,E):2, (B,{111}):2, (B,{311}):2, (C,{111}):2, (D,{121}):2, (D,{321}):2, (E,{111}):2, (E,{311}):2, ({111,311}):2, ({121,312}):2, (A,C,{111}):2, (A,D,{121}):2, (A,D,{321}):2, (A,{121,321}):2, (B,E,{111}):2, (B,E,{311}):2, (B,{111,311}):2, (D,{121,321}):2, (E,{111,311}):2, (A,D,{121,321}):2, (B,E,{111,311}):2	101
Frequent Closed Multi-Dimension Multi-Level Itemset Mining	**Level 1 :** ({1**}):6, (A,{1**}):4, (A, {3**}):3, ({1**,3**}):5, (A,C,{1**}):2, (A, D,{1**,3**}):2, (B,E,{1**,3**}):2 **Level 2 :** (A):4 , ({11*}):4, ({31*}):3, (A,C,{11*}):2, (A,D,{12*,32*}):2, (B,E,{11*,31*}):2 **Level 3 :** (A):4, ({111}):4, (A,C,{111}):2, (A,D,{121,321}):2, (B,E,{111,311}):2	18

From Table 4, the frequent multi-dimensional multi-level itemset mining without closed itemset mining generated 101 more patterns than other itemset mining. Our new method (the frequent multi-dimensional multi-level itemset mining with closed itemset mining) can reduce the generated results, which can represent all required patterns, as proved in section 3.4.

3.4 Correctness

The theorems show correctness of our new method of *CMDML* pattern mining as below.

Theorem 1: $CI \subseteq I$, where CI and I are set of closed itemset patterns and itemset patterns, respectively.

Proof: Based on definition 5 and property 2, we imply that *CMDML* represent every itemset of *MDML* in the same equivalence class sharing the same supporting and the number of complete closed multi-dimensional multi-level pattern derived form theorem 1 is not larger than the number of complete multi-dimensional multi-level pattern because of *CMDML* is subset of *MDML*.

Theorem 2: The set of *MDML* patterns can be generated from the set of *CMDML* pattern.

Proof: Based on property 3 and 4, the support of *MDML* pattern can be derived from the support of *CMDML* pattern. Therefore, we can conclude that the set of *MDML* patterns can be generated from the set of *CMDML* pattern.

4. Conclusion

The propose of this paper is to present the new method called *CMDML* that is a combining of multi-dimensional multi-level pattern mining and closed itemset mining to reduce a number of multi-dimensional multi-level patterns. Moreover, the closed multi-dimensional multi-level patterns derived from our new method cover all the result of multi-dimensional multi-level patterns and represent all patterns of multi-dimensional multi-level in equivalence class with the same support. Finally, the future research of effort will focus on an efficient parallel algorithm of closed multi-dimensional multi-level pattern mining.

References

[1] J. Han, J. Pei, and Y. Yin: "Mining frequent patterns without candidate generation", In Proceedings of ACM SIGMOD'00, page 1–12, May 2000.

[2] C. Lucchesse, S. Orlando, and R. Perego, "DCI-Closed: a fast and memory efficient algorithm to mine frequent closed itemsets", In B. Goethals, M. J. Zaki, and R. Bayardo, editors, *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, volume 126 of *CEUR Workshop Proceedings, Brighton, UK*, 1 November 2004.

[3] C. Lucchese, S. Orlando, and R. Perego, "Fast and Memory Efficient Mining of Frequent Closed Itemsets", IEEE Transaction on Knowledge and Data Engineering, Vol. 18, No. 1, pp. 21-36, Jan. 2006.

[4] J. Pei, J. Han, and R. Mao, "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets," Proc. ACM SIGMOD Int'l Workshop Data Mining and Knowledge Discovery, May 2000.

[5] J. Pei, J. Han, and J. Wang, "CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets," Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, Aug. 2003.

[6] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Efficient Mining of Association Rules Using Closed Itemset Lattices," Information Systems, vol. 24, no. 1, page. 25-46, 1999.

[7] N. Pasquier, Y. Bastide, R. Touil, and L. Lakhal, "Discovering frequent closed itemsets for association rules", In C. Beeri and P. Buneman, editors, Proceedings of 7th International Conference on Database Theory (ICDT 1999), LNCS, v.1540, Springer-Verlag, Jerusalem, Israel, pages 398–416, Jan. 1999.

[8] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Mining Based for Associate Rules using Galois Closed Sets," In Research Report, Computer Science Dept., Nice Sophia Antipolis University, 2000.

[9] M. Runying, "Adaptive-FP: An Efficient and Effective Method for Multi-Level Multi-Dimensional Frequent Pattern Mining", Apr. 2001

[10] T. Uno, T. Asai, Y. Uchida, H. Arimura, "LCM: An Efficient Algorithm for Enumerating Frequent Closed Item Sets", In Proc. IEEE ICDM'03 Workshop FIMI'03, 2003.

[11] M.J. Zaki and C.-J. Hsiao, "CHARM: An Efficient Algorithm for Closed Itemsets Mining," Proc. Second SIAM Int'l Conf. Data Mining, Apr. 2002.

[12] M.J. Zaki, "Mining Non-Redundant Association Rules," Data Mining and Knowledge Discovery, vol. 9, no. 3, pp. 223-248, 2004.

ประวัติผู้เขียน

ข้อมูลผู้เขียน

ชื่อ-นามสกุลไทย: จักริน วชิรเมธิน

ชื่อ-นามสกุลอังกฤษ: Chakarin Vajiramedhin

วันเดือนปีเกิด: 4 มกราคม พ.ศ.2519

ที่อยู่: 450 หมู่ที่ 7 ตำบลกำแพง อำเภออุทุมพรพิสัย จังหวัดศรีสะเกษ 33120

อีเมลล์: s9062951@kmitl.ac.th